# SUPPLEMENTARY TECHNICAL REFERENCE MANUAL

Version 1.1

for the

SIRIUS 1 Microcomputer

Originally produced by

Barson Computers P/L.,
335 Johnston Street,
ABBOTSFORD VIC 3067
AUSTRALIA
(03) 419 3033

This copy was
printed and supplied by

The VICTORIAN CONNECTION Bulletin Board

P.O. Box 6761

Silver Spring, MD 20906

The **VICTORIAN CONNECTION** Bulletin Board

(301) 460-7159

1200 baud


Capital Area Victor Users' Group,

CAVUG

P.O. Box 6255
Washington, DC 20015-0255



NOTICE:

The Capital Area Users' Group (CAVUG) and the VICTORIAN CONNECTION Bulletin Board (VCBB) assume no liability related to the use of this document or any of the information contained therein. The CAVUG and the VCBB do not make any representations or recommendations with respect to the contents of this document.



This document was printed on a **Hewlett-Packard LaserJet** printer using the **HP 92286D, Prestige Elite** font cartridge, and was driven by a **Victor 9000** microcomputer running **MicroPro Wordstar 3.3**.

SUPPLEMENTARY TECHNICAL REFERENCE MANUAL

Version 1.1

for the

SIRIUS 1 Microcomputer

Originally produced by

Barson Computers P/L.,
335 Johnston Street,
ABBOTSFORD VIC 3067
AUSTRALIA
(03) 419 3033

This copy was
printed and supplied by

The VICTORIAN CONNECTION Bulletin Board

P.O. Box 6761

Silver Spring, MD 20906

The VICTORIAN CONNECTION Bulletin Board

(301) 460-7159

1200 baud

Capital Area Victor Users' Group,

CAVUG

P.O. Box 6255
Washington, DC 20015-0255

NOTICE:

The Capital Area Users' Group (CAVUG) and the VICTORIAN CONNECTION Bulletin Board (VCBB) assume no liability related to the use of this document or any of the information contained therein. The CAVUG and the VCBB do not make any representations or recommendations with respect to the contents of this document.

This document was printed on a Hewlett-Packard LaserJet printer using the HP 92286D, Prestige Elite font cartridge, and was driven by a Victor 9000 microcomputer running MicroPro Wordstar 3.3.

# Supplementary Technical Reference Manual

Version 1.1

12-1-84

The following manual contains much general technical information on the Sirius 1 microcomputer. It is intended to be used as both a sales and support aid. It should be read in conjunction with the Hardware Reference Manual and the Operators Manual.

If you have any suggestions as to how this document could be improved, please fill in and return the Reader Comment Form you will find at the rear of the manual.

There are several sample software programs contained within this manual, most have been carefully tested; one program, the Transmit Page program written in MS-BASIC, is correct, but a bug in the latest release of MS-BASIC from Microsoft prevents it from working; the program will work once the bug has been fixed. The Pascal and Macro-86 examples of this program do work properly.

If you find any bugs in any other software program, or have any other problems or questions, please use the Reader Comment Form.

The following people and organisations are known to have contributed to the production of this manual and I wish to thank them for their efforts: ACT (Sirius) P/L, Victor Technologies Incorporated, Anne O'Hara, Stephen Page, Keith Pickup and Keith Rea.

Greg Johnstone
Barson Computers P/L.,
335 Johnston Street,
ABBOTSFORD VIC 3067
AUSTRALIA
(03)419 3033

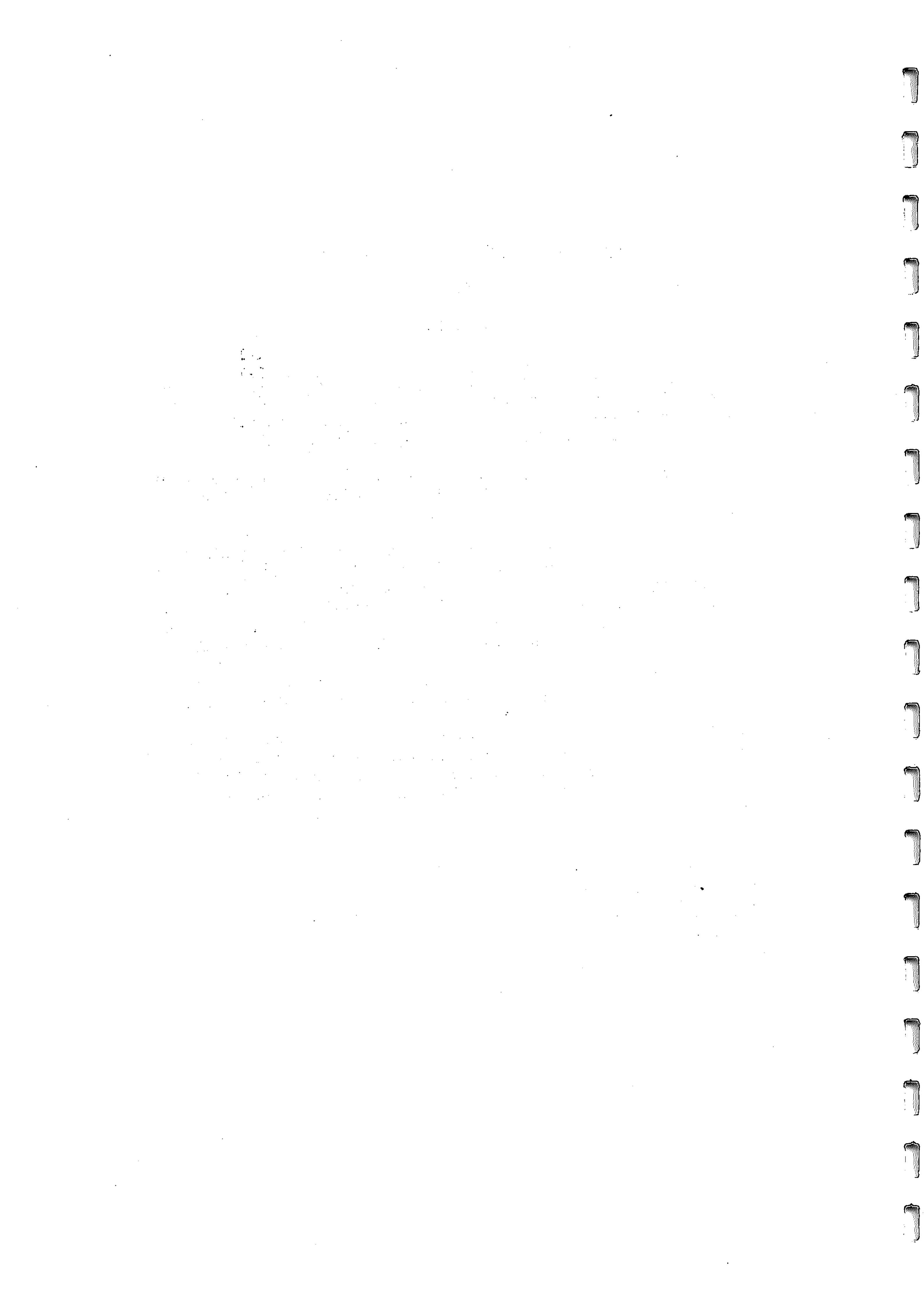


TABLE OF CONTENTS


| | | Page |
|---|---|---|
| 1. | Sirius 1 System Overview | |
| | 1.1 Computer | 1-1 |
| | 1.2 Memory | 1-1 |
| | 1.3 Disk System | 1-1 |
| | 1.4 Display System | 1-2 |
| | 1.5 Keyboard | 1-2 |
| | 1.6 Memory Map | 1-3 |
| | 1.6.1 MS-DOS | 1-4 |
| | 1.6.2 CP/M-86 | 1-5 |
| | 1.7 Memory Expansion & Requirements | 1-6 |
| | 1.7.1 Memory Organisation | 1-6 |
| | 1.7.2 Installation | 1-6 |
| | 1.7.3 Address Selection | 1-7 |
| | 1.7.4 Testing | 1-8 |
| | 1.7.5 Memory Requirements | 1-9 |
| 2. | Display Driver Specifications | |
| | 2.1 Overview | 2-1 |
| | 2.2 Screen Control Sequences | 2-1 |
| | 2.3 Multi-Character Escape Sequences | 2-2 |
| | 2.3.1 Cursor Functions | 2-2 |
| | 2.3.2 Editing Functions | 2-4 |
| | 2.3.3 Configuration Functions | 2-6 |
| | 2.3.4 Operation Mode Functions | 2-7 |
| | 2.3.5 Special Functions | 2-7 |
| | 2.3.6 VT52,Z19 commands | 2-10 |
| | 2.4 Direct Cursor Addressing - Examples | 2-10 |
| | 2.4.1 Microsoft MS-BASIC | 2-10 |
| | 2.4.2 Microsoft MACRO-86 | 2-11 |
| | 2.4.3 Microsoft MS-Pascal | 2-12 |
| | 2.5 Transmit Page - Examples | 2-12 |
| | 2.5.1 Microsoft MS-BASIC | 2-13 |
| | 2.5.2 Microsoft MACRO-86 | 2-13 |
| | 2.5.3 Microsoft MS-Pascal | 2-14 |
| | 2.6 25th Line Display Examples of use | 2-15 |
| | 2.6.1 Microsoft MACRO-86 Assembler | 2-15 |
| | 2.6.2 Microsoft MS-BASIC | 2-17 |
| | 2.7 132-column Display | 2-18 |
| 3. | Input/Output Port Specifications | |
| | 3.1 Device Connection | 3-1 |
| | 3.2 Parallel Port Signals | 3-2 |
| | 3.3 Parallel Printer Connection | 3-2 |
| | 3.3.1 Parallel Cable Requirements | 3-3 |

CONTENTS
continued


3.4 Serial Port Signals.................... 3-4
3.5 Serial Printer Connection.............. 3-7
3.5.1 Serial Cable Requirements........ 3-7
3.6 Operating System Port Utilities ....... 3-9
3.6.1 SETIO - List Device Selection ... 3-9
3.6.2 STAT - List Device Selection .... 3-12
3.6.3 PORTSET - Baud Rate Selection ... 3-14
3.6.4 PORTCONF - Baud Rate Selection .. 3-14
3.7 Serial Input/Output Port Addresses..... 3-14
3.8 Baud Rate / Transmission - Examples ... 3-15
3.8.1 Microsoft MS-BASIC .............. 3-15
3.8.2 Microsoft MACRO-86 .............. 3-17
3.9 Transferring Files to & from Computers. 3-19
3.10 IEEE-488 Port.......................... 3-21
3.11 Control Port (internal port)........... 3-22
3.12 MS-DOS Logical devices................. 3-23
3.13 Sample program for Initialising printers 3-25

4. MS-DOS Notes
4.1 MS-DOS Program Load.................... 4-1
4.1.1 MS-DOS Base Page Structure....... 4-1
4.2 The Command Processor.................. 4-5
4.2.1 Introduction..................... 4-5
4.2.2 Replacing the Command Processor.. 4-6
4.2.3 Available MS-DOS functions....... 4-6
4.2.4 Diskette/File Management Notes... 4-7
4.2.5 The Disc Transfer Area........... 4-7
4.2.6 Error Trapping................... 4-8
4.2.7 General Guidelines............... 4-8
4.2.8 Examples of using MS-DOS Functions 4-9
4.2.9 To Create File FILE1............. 4-10
4.3 MS-DOS Diskette Directory.............. 4-12
4.4 MS-DOS Program Segment................. 4-13

5. Miscellaneous Programming Notes
5.1 Rounding Numbers in Basic-86........... 5-1
5.2 Undocumented Commands of the Interpreter 5-1
5.2.1 Date$............................ 5-1
5.2.2 Time$............................ 5-2
5.2.3 Date............................. 5-2
5.2.4 Time............................. 5-3
5.2.5 Bload............................ 5-3
5.2.6 Bsave............................ 5-4
5.2.7 Open............................. 5-5

CONTENTS
continued


5.3 Calling Assembler from Basic Compiler.. 5-6
5.4 Program Size Limitations.............. 5-9
5.4.1 Memory Usage in Pascal........... 5-10
5.4.2 Memory Usage in Fortran.......... 5-10
5.4.3 Memory Usage Outside the default 64k Segment in MS-FORTRAN........ 5-12
5.5 Fix for ASYNC to load default ASYN.IEM. 5-12
5.6 Creating ASCII Text Files............. 5-13
5.7 Codec Programming..................... 5-14
5.7.1 Volume.......................... 5-14
5.7.2 Codec Clock..................... 5-15
5.7.3 Codec Mode Control.............. 5-16
5.7.4 SDA Initialisation.............. 5-16
5.7.5 SDA Data Transfer............... 5-17
5.8 Data Security......................... 5-18
5.9 MS-PASCAL Date & Time (input/output)... 5-21
5.10 Accessing System Time in dBASE II...... 5-22
5.11 Programming the 8253 Timer............ 5-26
5.12 Manipulating a Batch File............. 5-26
5.13 CALC (or UDCCALC) - Calculator Function 5-28
5.14 Directory Entries..................... 5-29
5.15 MS-DOS File Sizes and Disc Structure... 5-30

6. Wordprocessing Notes
6.1 Install for WordStar.................. 6-1
6.2 Summary of WordStar Patch Locations.... 6-2
6.3 Summary of Keyboard Table AUSWP4.KB.... 6-5
6.4 Convert CP/M WordStar to MS-DOS........ 6-8
6.5 Using C.Itoh F10 Printer with WordStar. 6-8
6.6 Benchmark............................ 6-9
6.7 XON/XOFF Printer Driver for Wordstar... 6-9

7. CP/M-80 System - Z-80 Card
7.1 Z-80 CPU Card......................... 7-1

8. Hard Disc
8.1 Hard Disc Introduction................ 8-1
8.2 Disc Drive Functional Characters....... 8-1
8.2.1 Disc Rotation................... 8-1
8.2.2 Head Positioning................ 8-1
8.2.3 Start/Stop...................... 8-2
8.2.4 Air Filtration.................. 8-2
8.2.5 Media........................... 8-2
8.2.6 Storage Capacity................ 8-2
8.3 Winchester Drive Handling Precautions.. 8-3

8.3.1 Do's and Don'ts................. 8-3
8.4 Hard Disc System Diagnostics........... 8-3
8.5 Hard Disc Problems..................... 8-4

9. **Local Area Network**
9.1 Introduction.......................... 9-1
9.1.1 Introduction to LAN............ 9-1
9.1.2 ISO Seven Layer Network Model... 9-1
9.2 Local Area Network Overview........... 9-3
9.3 Network Software Overview............. 9-5

10. **High Resolution Graphics**
10.1 Introduction.......................... 10-1
10.2 Clearing a Hi-Res area................ 10-1
10.3 Setting Screen Buffer Pointers......... 10-2
10.4 Reprogramming the CRT Controller....... 10-2
10.5 Examples.............................. 10-3
10.5.1 Microsoft MACRO-86 Assembler.... 10-3
10.5.2 Microsoft MS-BASIC Interpreter.. 10-9
10.6 Printer Configuration in Grafix Kernel. 10-10
10.7 Patching Grafix Kernel for MT-180...... 10-14
10.8 Character Printing.................... 10-15
10.9 Patching CHRPRINT for the MT-180....... 10-16

11. **Assembly to High Level Interface**
11.1 Interfacing Basic with Assembler....... 11-1
11.1.1 Calling Assembler Subroutines... 11-1
11.1.2 Basic Data Types............... 11-2
11.1.3 Passing Parameters............. 11-3
11.1.4 Example........................ 11-4
11.2 Interfacing Compiled Basic with Assembler 11-6
11.2.1 Calling Assembler Subroutines... 11-7
11.2.2 Compiled Basic Data Types....... 11-7
11.2.3 Passing parameters............. 11-8
11.2.4 Example........................ 11-9
11.3 Interfacing GWBasic with Assembler..... 11-11
11.3.1 GWBasic Data Types............. 11-12
11.3.2 Passing Parameters............. 11-13
11.3.3 Example........................ 11-13
11.4 Interfacing MS-COBOL with Assembler.... 11-16
11.4.1 Calling assembler Subroutines... 11-16
11.4.2 Cobol Dtata Types.............. 11-17
11.4.3 Passing Parameters............. 11-19
11.4.4 Example........................ 11-19

CONTENTS
continued

11.5 Interfacing MS-Pascal with Assembler... 11-21
11.5.1 Calling external Subroutines.... 11-21
11.5.2 Passing Parameters.............. 11-22
11.5.3 Pascal Data Types............... 11-23
11.5.4 Returned Values................. 11-25
11.5.5 Example 1 - Sum Function........ 11-26
11.5.6 Example 2 - String Concatenation 11-28
11.5.7 Linking......................... 11-30

Appendices

Appendix A: ASCII Codes
A.1 ASCII Codes used in the Sirius 1 A-1
A.2 ASCII/Hex/Decimal Chart ........... A-2

Appendix B: Keyboard
B.1 Sirius 1 Keyboard Layout .......... B-1

Appendix C: Input/Output Ports
C.1 Parallel & Serial Cable Requirement C-1
C.2 Parallel Printer Cables............ C-2
C.3 Serial Printer Cables.............. C-6

Appendix D: Assembler Examples
D.1 MACRO-86 Assembler Shell .......... D-1
D.2 ASM-86 Assembler Shell ............ D-2

Appendix E: MS-DOS EXE File Header Structure
.................................. E-1

Appendix F: Sirius 1 Specifications
F.1 Technical Specifications .......... F-1
F.2 Physical Specifications ........... F-2

Appendix G: Glossary
.................................. G-1

Appendix H: Dealers Demonstration Package
H.1 Disc 1 Latest Graphics Demo........ H-1
H.2 Disc 2 Sliding Picture Show........ H-5
H.3 Disc 3 1550 (C.Itoh) Graphics...... H-6
H.4 Disc 4 Arabic Demonstration........ H-6

CONTENTS
continued

Appendix I: Interrupt Driven Serial Input/Output
I.1 Introduction...................... I-1
I.2 Interrupt Vectors.................. I-1
I.2.1 Vectors Available on Sirius.. I-1
I.2.2 Location of Vectors.......... I-2
I.2.3 Set Vector - Assembler....... I-2
I.3 Enabling Internal & External clocks I-3
I.3.1 Providing Clocks............ I-3
I.4 Initialising the SIO............... I-4
I.4.1 Baud Rate for SIO........... I-5
I.4.2 Set PIC to enable SIO Interr. I-5
I.5 Interrupt Service Routine - ISR.... I-6
I.5.1 Sample ISR.................. I-6
I.6 Setting Direction Bits............. I-7

Appendix J: File Header Information
J.1 Character Set Header............... J-1
J.1.1 Sample Character Set Table File Header................. J-3
J.2 Prop. Character Set Trailer Info... J-3
J.3 Keyboard Table Header.............. J-4
J.4 Banner Skeleton Files.............. J-5
J.5 Banner Customisation............... J-5
J.6 Logo Creation...................... J-6
J.7 Normal File Control Block.......... J-6
J.8 Extended File Control Block........ J-8

Appendix K: Comparisons Between MS-DOS & CP/M-86 .................................. K-1

Appendix L: Features included in MS-DOS Version 2 .................................. L-1

Appendix M: SIRIUS I Dealer Spare Parts Kit .................................. M-1

Appendix N: Double Sided Diskettes
N.1 Double sided diskettes............. N-1
N.2 Boot Disc Label format............. N-3

Appendix O: Functional Specifications of Boot ROM
O.1 Diagnostic ROM Board Support....... O-1
O.2 ICONS for boot ROM version P1...... O-2
O.3 Exception Displays................. O-3

CONTENTS
continued


O.4 Universal Boot EPROMS.............. O-3

Appendix P: Transferring files from Commodore to Sirius
.................................. P-1

Appendix Q: Unprotecting Discs
.................................. Q-1

Appendix R: ASYNC Protocol
R.1 Data Block......................... R-1
R.2 File name Blocks................... R-2

Appendix S: Communications
S.1 IBM Remote Batch Emulation......... S-1
S.2 IBM 3270 Emulation Package......... S-2
S.3 Asynchronous Communications Package S-4
S.4 ASYNC Package - Remote Terminal.... S-6

# SIRIUS 1 SYSTEM OVERVIEW

## 1.1 Computer

The Sirius 1 computer is based upon the Intel 8088 16-bit microprocessor. This processor chip is directly related to the Intel 8086 16-bit microprocessor, but with two subtle differences:

| 8088 | 8086 |
| --- | --- |
| 8-bit data bus | 16-bit data bus |
| 4 instruction look-ahead | 6 instruction look-ahead |

The major difference, the 8-bit data bus, has some effect on the relative abilities of the two chips; the main difference is that while the 8086 can load an entire 16-bit word of data directly, the 8088 has to load two 8-bit bytes to achieve the same result - the outcome of which being that the 8088 processor is a little slower than the 8086. The loss of speed, however, is balanced by the fact that the cost of the main circuit board and add-on boards are lower than for the wider 8086 requirement. This means that the end-user will have the best cost/performance ratio for a 16-bit computer.

## 1.2 Memory

The Sirius 1 has a maximum memory capacity of 896 kilobytes of Random Access Memory or "RAM" (a measure of a computer's internal storage capacity; a "kilobyte" is 1,024 bytes). A byte is able to store one character of data - thus the Sirius 1, with full 896K memory capacity is able to hold, internally, nearly 1 million characters - compare this figure with the older Z80 or 6502 computers that have a maximum memory capacity of less than 70,000 characters or 64k bytes of RAM.

## 1.3 Disk System

The Sirius 1 has several integral disc configurations available; these are:

- o Twin single-sided 600k bytes per drive 130mm minifloppies, giving a total capacity of 1.2Mbytes (1,200Kbytes) available on-line.
- o Twin double-sided 1.2M bytes per drive 130mm minifloppies, giving a total capacity of 2.4Mbytes (2,400Kbytes) available on-line.

- Single 10M byte hard disc (Winchester) plus a single double-sided 1.2M byte 130mm floppy, giving a total capacity of 11.2Mbytes (11,200Kbytes) available on-line.

Future disc systems will include an external 10Mbyte hard disc (Winchester) that will allow expansion of any of the above systems by a further 10,000K bytes.

Although the Sirius 1 uses 130mm minifloppies of a similar type to those used in other computers, the floppy discs themselves are not readable on other machines, nor can the Sirius 1 read a disc from another manufacturers machine. The Sirius 1 uses a unique recording method to allow the data to be packed as densely as 600Kbytes on a single-sided single-density minifloppy; this recording method involves the regulation of the speed at which the floppy rotates, explaining the fact that the noise from the drive sometimes changes frequency.

## 1.4 Display System

The display unit swivels and tilts to permit optimum adjustment of the viewing angle, and the unit incorporates a 300mm antiglare screen to prevent eye strain. The display, in normal mode, is 25 lines, each line having 80 columns. Characters are formed, in normal mode, in a 10-x-16 font cell, providing a highly-readable display. The screen may be used in high-resolution mode, providing a bit-mapped screen with 800-x-400 dot matrix resolution. The high-resolution mode is available only under software control, there is no means of simply "switching" in to high-resolution. Victor Technologies has provided software to allow full use of the screen in high-resolution mode in the Graphics Tool Kit.

Character sets are "soft" - that is they may be substituted for alternative character sets of the users choice, or creation. Only one 256-character character set may be displayed on the screen at one time - multiple character sets cannot, currently, be displayed simultaneously - but this feature may well become available in the future. Character set manipulation software is available in both the Graphics and Programmers Tool Kits.

## 1.5 Keyboard

Every key is programmable, permitting the offering of a National keyboard in each country in which it is marketed. As a result, the keyboard can be customised to satisfy the requirements of foreign languages and so that striking a key enters a character or predetermined set of commands.

Keyboards are as soft as the character sets - this allows a keyboard to be generated to match a newly created or special character set. Each key on the keyboard has three potential states; the unshifted, shifted and alternate. The unshifted mode is accessed when the desired key is depressed; the shifted mode is accessed when the shift key is depressed along with the desired key; and the alternate mode is accessed when the ALT key is depressed along with the desired key. Keyboard manipulation software is available in both the Graphics and Programmers Tool Kits.

## 1.6 Memory Map

The Sirius 1 is currently supplied with two major disc operating systems; CP/M-86 from Digital Research, and MS-DOS from Microsoft. Athough these two operating systems appear superficially similar, they are quite different in their operation, program interfacing techniques, and their memory structure. The following diagrams are the memory maps for CP/M-86 and MS-DOS; you will notice that some aspects of the machine never change, such as the screen RAM and interrupt vector locations, these areas are hardware defined, and as such never alter. The memory maps for MS-DOS and CP/M-86 are not fixed in the Sirius 1, thus some of the elements of the map will not be specific; this is not to be deliberately vague, but improvements to the performance aspects of the software do take place forcing the diagrams to be unspecific to some degree.

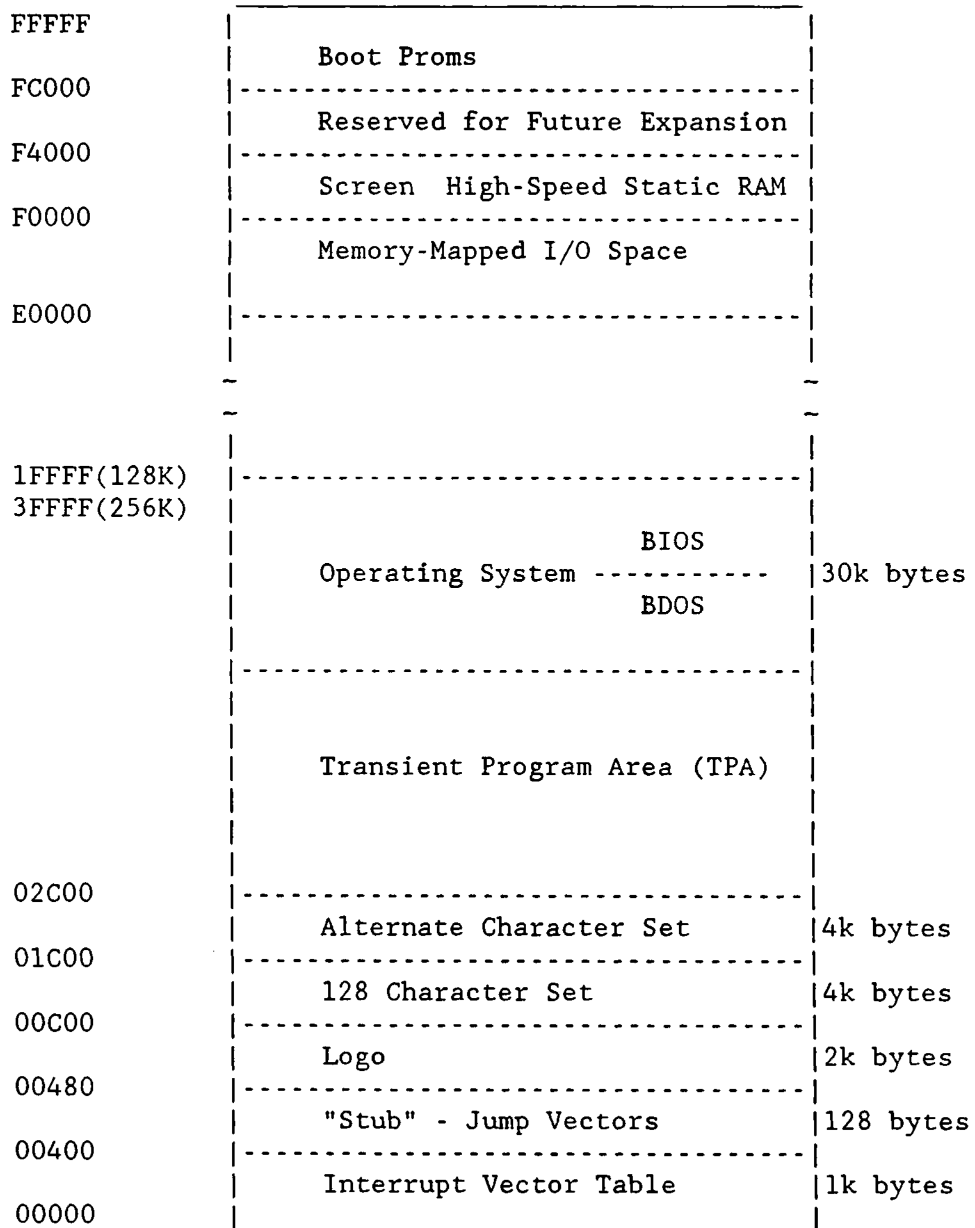
Note that in CP/M-86 user programs load in the Transient Program Area (TPA) from the top down, whereas in MS-DOS, they load from the bottom up.

## 1.6.1 Memory Map -- MS-DOS Operating System

| Address | Region | Size |
| --- | --- | --- |
| FFFFF | | |
| | Boot Proms | |
| FC000 | | |
| | Reserved for Future Expansion | |
| F4000 | | |
| | Screen High-Speed Static RAM | |
| F0000 | | |
| | Memory-Mapped I/O Space | |
| E0000 | | |
| | ~ ~ | |
| etc.<br>256k=3FFF0<br>128k=1FFF0 | Operating System -- BIOS / MS-DOS | |
| | Command - Resident Portion | |
| | Command - Transient Portion | |
| | Transient Program Area (TPA) | |
| 02C00 | | |
| | Alternate Character Set | 4k bytes |
| 01C00 | | |
| | 128 Character Set | 4k bytes |
| 00C00 | | |
| | Logo | 2k bytes |
| 00480 | | |
| | "Stub" - Jump Vectors | 128 bytes |
| 00400 | | |
| | Interrupt Vector Table | 1k bytes |
| 00000 | | |

Programs load from the bottom of TPA

## 1.6.2 Memory Map -- CP/M-86 Operating System




Programs load from the top of TPA

## 1.7 Memory Expansion and Memory Requirements

### 1.7.1 Memory Organisation

The total address space of the Sirius 1 is 1Mbyte (or 64K paragraphs; a paragraph being 16 bytes). For convenience this can be considered as 8 blocks of 128K, as follows:

```
Address
(paragraphs)

FFFF     |----------|      Memory mapped I/O,
         |    7     | }    screen RAM, ROM, etc.
E000     |----------|
         |    6     | \
C000     |----------|  |
         |    5     |  |
A000     |----------|  |
         |    4     |  |
8000     |----------|  >  Expansion
         |    3     |  |
6000     |----------|  |
         |    2     |  |
4000     |----------|  |
         |    1     |  |
2000     |----------| /
         |    0     | }   RAM supplied with standard
0000     |----------|     128K machine
```

RAM expansion boards are 128K, 256K and 384K (3 x 128K) capacity. The 128K and 256K boards can be addressed at any 128K boundary in the area of memory reserved for expansion (ie. addresses starting at blocks 1 to 6 in the diagram). The 384K boards can (currently) only be addressed at blocks 1 or 4. Later versions will allow the 384K board to be addressed similarly to the other two boards.

RAM must be contiguous, thus if two 128K RAM cards are used in a standard 128K machine, for example, they must be addressed at blocks 1 and 2. If a 384K RAM card and a 128K RAM card are used, the 384K RAM card must be addressed at blocks 1-3 and the 128K RAM card must be addressed at block 4.

### 1.7.2 Installation

A RAM card may occupy any of the four expansion slots on the motherboard. A plastic retainer is supplied to hold the top of the board in place. When installed, the component side of the

board should face away from the disc drives.

### 1.7.3 Address Selection

128K and 256K RAM card:

The group of DIP switches on the card is used to select which blocks the RAM should address. Switches 1 to 6 select blocks 1 to 6 respectively. If a 256K board is to address blocks 3 and 4 then switches 3 and 4 should be on. If a 128K board is to address block 1, switch 1 should be on.

Switches 7 and 8 are used to select refresh rates. These should be left as set.

384K RAM card:

Older versions of this board are addressed by two jumpers. They are supplied with jumpers E2-E3 and E4-E6 soldered in. This configuration addresses blocks 1 to 3. The jumpers should be changed to E1-E2 and E5-E6 to address blocks 4 to 6. Later versions have a single jumper which can be used to select 'upper' or 'lower' banks of 384K. Yet-to-be-released versions of the 384K RAM card will use similar addressing methods to the 128K and 256K RAM cards.

NOTE: A possible problem exists when using current or older builds of the 384K RAM card with a 256K CPU board. It is necessary to modify the RAM card so that the first 128K bank of RAM is relocated to reside in the range 40000-5FFFFh. This is done by modifying the address decoding on the 384K RAM card. There are at present two 384K RAM cards supported. They have the part numbers "101070-01 C 10663" and "101070-01 B D 10870". The modifications for the two boards are shown below:-

For board number 101070-01 B D 10870

1. Locate device 1D (74LS11) on the 384K PCB.

2. Cut the track going to pin 1 of this device on the bottom of the board (coming from pin 14 of device 1B (74LS138)).

3. Link pins 1 and 3 of device 1D (best done on the bottom of the board with a small piece of wire wrap wire).

4. Inspect the modification, checking to make sure that the track is cut and that there are no solder shorts.

5. Ensure that J1 (next to device 1F) is linked to the 'LOWER' position.

6. Proceed to "testing" section.

For board number 101070-01 C 10663

1. Carry out the modification as per 1 to 5 above.

2. Locate the jumper configuration E1,E2 and E3 (to the left of device 2B, next to R1 and C10). Remove the wire link connecting either E2 to E1 or E2 to E3 and discard.

3. Link device 2B (74LS86) pin 4 to device 1B (74LS138) pin 3.

4. Inspect the modification, checking to make sure that the link is open and that there are no solder shorts.

5. Proceed to "testing" section.

Testing:

Install the RAM card in the machine and turn on. If all is well, the machine will display "A000" paragraphs of memory when booting from a drive instead of the normal "4000" paragraphs message. The machine now has 640K of RAM from 00000h to 9FFFFh.

NOTE: 1 paragraph=16 bytes and Sirius displays the number of paragraphs in hexadecimal.

1.7.4 **Testing**

When installed, the board should be tested by booting the system. During boot, RAM is tested and the amount of RAM (in paragraphs) is displayed at the bottom of the screen. For example, the standard 128K machine displays **M** 2000, a 256K machine displays **M** 4000. If an error is found in RAM, Sirius will not boot. Machines with extra RAM take longer to boot. (See also, Appendix O).

The latest 'Universal' boot ROMs (see Appendix O) display memory in kilobytes rather than paragraphs. Thus the display on a 256K machine will be **M** 256K.

### 1.7.5 Memory Requirements

The operating system always relocates itself to take account of all available memory.

At this moment, Microsoft Fortran, Pascal and Cobol compilers require 256K to compile, although the compiled programs may run in 128K. The graphics package requires 256K minimum and will use up to 512K.

Microsoft Basic interpreter will not allow the user more than about 62K of free space.

Supercalc, Autocad and Scientex use all available memory.

SIRIUS SYSTEM OVERVIEW 1-10

# DISPLAY DRIVER SPECIFICATIONS

## 2.1 Overview

The display system in the Sirius 1 is, like so much of the machine, soft. The operating system BIOS contains the Zenith H-19 video terminal emulator, which is an enhanced control set of the DEC VT52 crt. The BIOS takes all ASCII characters received and either displays them or uses their control characteristics. The control characters 00hex (00decimal) through 1Fhex (31decimal) and 7Fhex (127decimal) are not displayed under normal circumstances. The non-display characters previously discussed, plus those characters having the high-bit set, being 80hex (128decimal) through FFhex (255decimal), may be displayed on the screen under program control, but extensive use of these characters is easier with the character graphics utilities.

Most of the control characters act by themselves; for example, the TAB key (Control I, 09hex, 09decimal) will cause the cursor to move to the right to the next tab position. For more complex cursor/screen control the multiple character escape sequences should be used. The control characters, and the escape sequences are fully described below.

## 2.2 Screen Control Sequences

### Single Control Characters

Bell (Control G, 07hex, 07decimal - ASCII BEL)
This ASCII character is not truly a displaying character, but causes the loudspeaker to make a beep.

Backspace (Control H, 08hex, 08decimal - ASCII BS)
Causes the cursor to be positioned one column to the left of its current position. If at column 1, it causes the cursor to be placed at column 80 of the previous line; if the cursor is at column 1, line 1, then the cursor moves to column 80 of line 1.

Horizontal Tab (Control I, 09hex, 09decimal - ASCII HT)
Positions the cursor at the next tab stop to the right. Tab stops are fixed, and are at columns 9, 17, 25, 33, 41, 49, 57, 65, and 72 through 80. If the cursor is at column 80, it remains there.

Line Feed (Control J, 0Ahex, 10decimal - ASCII LF)
Positions the cursor down one line. If at line 24, then

the display scrolls up one line. This key may be treated as a carriage return -- see ESC x9.

Carriage Return (Control M, 0Dhex, 13decimal - ASCII CR)
Positions the cursor at column 1 of the current line. This key may be treated as a line feed -- see ESC x8.

Shift Out (Control N, 0Ehex, 14decimal - ASCII SO)
Shift out of the standard system character set, and shift into the alternative system character set (Character set 1, G1). This gives the ability to access and display those characters having the high-bit set - being those characters from 80hex (128decimal) through FFhex (255decimal).

Shift In (Control O, 0Fhex, 15decimal - ASCII SI)
Shift into the standard system character set (Character set 0, G0). This gives the ability to access and display the standard ASCII character set - being those characters from 00hex (00decimal) through 7Fhex (127decimal).

Escape (Control [, 1Bhex, 27decimal - ASCII ESC)
Tells the video driver that a command of one or more characters follows. See section 2.3.

## 2.3 Multi-Character Escape Sequences

### 2.3.1 Cursor Functions

As well as the above control characters, the video driver has a large vocabulary of commands which are several characters long. The first character of these commands is the control character called Escape (ESC), which has the value 27 in the ASCII character set. When the video driver is sent an ESC character it performs whatever function is specified by the following characters. This kind of command is called an "escape sequence".

To make it easier for programs written on other computers to be run on the Sirius, the set of escape sequences is designed to be very similar to a DEC VT52 terminal. In addition, some of the more fancy features are borrowed from a Heath Z19 terminal.

To send an escape sequence from a Basic program, use the following statement sequence (the example shows how to send an ESC-A and ESC-1):

```
10 E$ = CHR$(27)   'put the ESC character in E$
20 PRINT E$;"A";   'ESC-A moves up a line
30 PRINT E$;"1";   'ESC-1 clears the line
```

| Escape Sequence/Function | ASCII Code | Performed Function |
|---|---|---|
| ESC A | 1B, 41hex<br>27, 65dec | Move cursor up one line without changing column. |
| ESC B | 1B, 42hex<br>27, 66dec | Move cursor down one line without changing column. |
| ESC C | 1B, 43hex<br>27, 67dec | Move cursor forward one character position. |
| ESC D | 1B, 44hex<br>27, 68dec | Move cursor backward one character position. |
| ESC H | 1B, 48hex<br>27, 72dec | Move cursor to the home position. Cursor moves to line 1, column 1. |
| ESC I | 1B, 49hex<br>27, 73dec | Reverse index. Move cursor up to previous line at current column position. |
| ESC Y l c | 1B, 59hex<br>27, 89dec | Moves the cursor via direct (absolute) addressing to the line and column location described by 'l' and 'c'. The line ('l') and column ('c') coordinates are binary values offset from 20hex (32decimal). Thus, to move to the end of the top line, we use (in Basic):<br>PRINT CHR$(27);"Y";CHR$(32+0);CHR$(32+79)<br>(For further information on the use of direct addressing see section 2.4). |
| ESC j | 1B, 6Ahex<br>27, 106dec | Store the current cursor position. The cursor location |

| | | |
|---|---|---|
| | | is saved for later restoration (see ESC k). |
| ESC k | 1B, 6Bhex<br>27, 107dec | Returns cursor to the previously saved location (see ESC j). |
| ESC n | 1B, 6Ehex<br>27, 110dec | Return the current cursor position. The current cursor location is returned as line and column, offset from 20hex (32decimal), in the next character input request. |

### 2.3.2 Editing Functions

| Escape Sequence/Function | ASCII Code | Performed Function |
|---|---|---|
| ESC @ | 1B, 40hex<br>27, 64dec | Enter the character insert mode. Characters may be added at the current cursor position, as each new character is added, the character at the end of the line is lost. |
| ESC E | 1B, 45hex<br>27, 69dec | Erase the entire screen. |
| ESC J | 1B, 4Ahex<br>27, 74dec | Erase from the current cursor position to the end of the screen. |
| ESC K | 1B, 4Bhex<br>27, 75dec | Erase the screen from the current cursor position to the end of the line. |
| ESC L | 1B, 4Chex<br>27, 76dec | Insert a blank line on the current cursor line. The current line, and all following lines are moved down one, and the cursor is placed at the beginning of the blank line. |
| ESC M | 1B, 4Dhex<br>27, 77dec | Delete the line containing the cursor, place the cursor at |

| | | |
|---|---|---|
| | | the start of the line, and move all following lines up one - a blank line is inserted at line 24. |
| ESC N | 1B, 4Ehex<br>27, 78dec | Delete the character at the cursor position, and move all other characters on the line after the cursor to the left one character position. |
| ESC O | 1B, 4Fhex<br>27, 79dec | Exit from the character insert mode (see ESC @). |
| ESC X | 1B, 58hex<br>27, 88dec | Exchanges the current line for the contents of an internal buffer. To swap two lines, do the following:<br><br>move cursor to first line ESC X (puts line into internal buffer)<br><br>move cursor to other line ESC X (swaps first line for this)<br><br>move cursor to first line ESC X (puts second line where first was) |
| ESC b | 1B, 62hex<br>27, 98dec | Erase the screen from the start of the screen up to, and including, the current cursor position. |
| ESC 1 | 1B, 6Chex<br>27, 108dec | Erase entire current cursor line. |
| ESC o | 1B, 6Fhex<br>27, 111dec | Erase the beginning of the line up to, and including, the current cursor position. |

### 2.3.3 Configuration Functions

| Escape Sequence/Function | ASCII Code | Performed Function |
|---|---|---|
| ESC x Ps | 1B, 78hex<br>27, 120dec | Sets mode(s) as follows: |
| | | **Ps — Mode** |
| | 31hex, 49dec | 1 Enable 25th line (see section 5.5) |
| | 33hex, 51dec | 3 Hold screen mode on |
| | 34hex, 52dec | 4 Block cursor |
| | 35hex, 53dec | 5 Cursor off |
| | 38hex, 56dec | 8 Auto line feed on receipt of a carriage return. |
| | 39hex, 57dec | 9 Auto carriage return on receipt of line feed |
| | 41hex, 65dec | A Increase audio volume |
| | 42hex, 66dec | B Increase CRT brightness |
| | 43hex, 67dec | C Increase CRT contrast |
| | | For example, to disable the cursor, use (in Basic):<br>PRINT CHR$(27);"x5" |
| ESC y Ps | 1B, 79hex<br>27, 120dec | Resets mode(s) as follows: |
| | | **Ps — Mode** |
| | 31hex, 49dec | 1 Disable 25th line |
| | 33hex, 51dec | 3 Hold screen mode off |
| | 34hex, 52dec | 4 Underscore cursor |
| | 35hex, 53dec | 5 Cursor on |
| | 38hex, 56dec | 8 No auto line feed on receipt of a carriage return. |
| | 39hex, 57dec | 9 No auto carriage return on receipt of line feed |
| | 41hex, 65dec | A Decrease audio volume |
| | 42hex, 66dec | B Decrease CRT brightness |
| | 43hex, 67dec | C Decrease CRT contrast |
| ESC [ | 1B, 5Bhex<br>27, 91dec | Set hold mode |
| ESC \ | 1B, 5Chex<br>27, 92dec | Clear hold mode |
| ESC ^ | 1B, 5Ehex<br>27, 94dec | Toggle hold mode on/off. |

### 2.3.4 Operation Mode Functions

| Escape Sequence/Function | ASCII Code | Performed Function |
|---|---|---|
| ESC ( | 1B, 28hex<br>27, 40dec | Enter high intensity mode. All characters displayed after this point will be displayed in high-intensity. |
| ESC ) | 1B, 29hex<br>1B, 41dec | Exit high intensity mode. |
| ESC 0 | 1B, 30hex<br>27, 48dec | Enter underline mode. All characters displayed after this point will be underlined. |
| ESC 1 | 1B, 31hex<br>27, 49dec | Exit underline mode. |
| ESC p | 1B, 70hex<br>27, 112dec | Enter reverse video mode. All characters displayed after this point will be displayed in reverse video. |
| ESC q | 1B, 71hex<br>27, 113dec | Exit reverse video mode. |

### 2.3.5 Special Functions

| Escape Sequence/Function | ASCII Code | Performed Function |
|---|---|---|
| ESC # | 1B, 23hex<br>27, 35dec | Return the current contents of the page. The entire contents of the screen are made available at the next character input request(s). (For further information on the use of this function, see section 2.5). |
| ESC $ | 1B, 24hex<br>27, 36dec | Return the value of the character at the current cursor position. The character is returned in the next character input request. |

| | | |
|---|---|---|
| ESC + | 1B, 2Bhex<br>27, 43dec | Clear the foreground. Clear all high-intensity displayed characters. |
| ESC 2 | 1B, 32hex<br>27, 50dec | Make cursor blink. |
| ESC 3 | 1B, 33hex<br>27, 51dec | Stop cursor blink. |
| ESC 4 | 1B, 34hex<br>27, 52dec | Temporarily generate different characters on the keyboard. Though the escape sequence is listed under 132C, it can be used under MBASIC on MS-DOS. Eg. change function key no. 1 to backslash, escape sequence - Esc 4 m lk kv. Where m=character 1, 2 or 3, 1 = unshift, 2 = shift, 3 = alternate. lk = logical key number (00 -7Fhex), kv = hexadecimal ASCII keycode of the new key value. |
| ESC 8 | 1B, 38hex<br>27, 56dec | Set the text (literally) mode for the next single character. This allows the display of characters from 01hex (01dec) through 1Fhex (31dec) on the screen. Thus the BELL character (07hex, 07dec) will not cause the bleep, but a character will appear on the screen. For example,<br><br>PRINT CHR$(27);"8";CHR$(12); will print whatever graphic character occupies position 12 in the current set. |
| ESC F | 1B 40hex<br>27 70dec | The other method of accessing the graphics characters stored in positions 0-31 of the current set. This escape code maps the graphics characters into codes 94-127, i.e. it replaces the lower case letter |

by graphics characters.

| | | |
|---|---|---|
| ESC G | 1B 41hex<br>27 71hex | Clear graphics mode. |
| ESC Z | 1B, 5Ahex<br>27, 90dec | Identify terminal type. The VT52 emulator will return ESC\Z in the next character input request. |
| ESC ] | 1B, 5Dhex<br>27, 93dec | Return the value of the 25th line. The next series of character input requests will receive the current contents of the 25th line. |
| ESC v | 1B, 76hex<br>27, 118dec | Enable wrap-around at the end of each screen line. A character placed after column 80 of a line will be placed on the next line at column 1. |
| ESC w | 1B, 77hex<br>27, 119dec | Disable wrap-around at the end of each line. |
| ESC z | 1B, 7Ahex<br>27, 122dec | Reset terminal emulator to the power-on state. This clears all user selected modes, clears the screen, and homes the cursor. |
| ESC { | 1B, 7Bhex<br>27, 123dec | Enable keyboard input. (see ESC }). |
| ESC } | 1B, 7Dhex<br>27, 125dec | Disable keyboard input. This locks the keyboard. Any character(s) typed are ignored until an ESC { is issued. |
| ESC \| | 1B 7Chex<br>27 124dec | Activate user-defined console. When the 132 column utility is in memory, this escape sequence transfers control to 132C. |
| ESC i Ps | 1B, 69hex<br>27, 105dec | Displays banner as follows:<br><br>Ps Mode |

| | | |
|---|---|---|
| 30hex, 48dec | 0 | Display entire banner |
| 31hex, 49dec | 1 | Display company logo |
| 32hex, 50dec | 2 | Display operating system |
| 33hex, 51dec | 3 | Display configuration |

**2.3.6 The following VT52/Z19 commands are accepted but do nothing at present:**

ESC x2 Disable key click.
ESC y2 Enable key click.

ESC x6 Enable keypad shift.
ESC y6 Disable keypad shift.
ESC t Enable keypad shift.
ESC u Disable keypad shift.

ESC x7 Enter alternate keypad mode.
ESC y7 Exit alternate keypad mode.
ESC = Enter alternate keypad mode.
ESC > Exit alternate keypad mode.

## 2.4 Direct Cursor Addressing -- Examples of Use

The direct cursor addressing function is accessed by sending the ESC Y l c sequence to the screen (see section 2.3.1). "l" is the line number required, whose valid coordinates are between 1 and 24. An offset of 1Fhex (31decimal) must be added to the location required in order to correctly locate the cursor. "c" is the column number required, whose valid coordinates are between 1 and 80. An offset of 1Fhex (31decimal) must be added to the location required in order to correctly locate the cursor.

Note that the true offset requirement of 20hex (32decimal) for line and column may only be used accurately when the line number is viewed 0 to 23, and the column number 0 to 79.

The line/column number requested must be handled as a binary digit, examples of this follow:

### 2.4.1 Microsoft MS-BASIC -- Direct Cursor Positioning

The following method uses offsets from line 1, column 1:

```
10 PRINT CHR$(27)+"E"    :REM CLEAR THE SCREEN
20 DEF FNM$(LIN,COL)=CHR$(27)+"Y"+CHR$(31+LIN)+CHR$(31+COL)
30 PRINT "Enter line (1-24) and column (1-80), as LINE,COL ";
40 INPUT LIN, COL
50 PRINT FNM$(LIN,COL);
```

```
60 FOR I = 1 TO 1000   :REM PAUSE BEFORE OK MESSAGE DISPLAYED
70 NEXT I
```

The alternative method, using offsets from zero is shown below:

```
10 PRINT CHR$(27)+"E"   :REM CLEAR THE SCREEN
20 DEF FNM$(LIN,COL)=CHR$(27)+"Y"+CHR$(32+LIN)+CHR$(32+COL)
30 PRINT "Enter line (0-23) and column (0-79), as LINE,COL ";
40 INPUT LIN, COL
50 PRINT FNM$(LIN,COL);
60 FOR I = 1 TO 1000   :REM PAUSE BEFORE OK MESSAGE DISPLAYED
70 NEXT I
```

### 2.4.2 Microsoft MACRO-86 Assembler -- Direct Cursor Positioning

```
line_off   equ  20h                   ;line position offset from 0
col_off    equ  20h                   ;column position offset from 0
esc        equ  1bh                   ;escape character
msdos      equ  21h                   ;interrupt to MS-DOS

clear_screen        db   esc,'E$'  ;clear screen request
dir_cur_pos_lead    db   esc,'Y$'  ;cursor positioning lead-in

; the cursor position required is handed down in BX
;    where BH = line (0-23 binary), BL = column (0-79 binary)

clear_and_locate:
     mov  ah,9h                       ;string output up to $
     mov  dx,offset clear_screen      ;get the clear screen string
     int  msdos                       ;and output it up to the $
;
; the cursor pcsition required is in BX
     add  bh,line_off                 ;normalise line for output
     add  bl,col_off                  ;normalise column for output
;
; send the direct cursor positioning lead-in
;
     mov  ah,9h                       ;select screen output up to $
     mov  dx,offset dir_cur_pos_lead ;select the lead in ESC Y
     int  msdos                       ;and output it up to $
;
; now the contents of BX must be sent to the terminal emulator
;
     mov  dl,bh                       ;ready the line number
     mov  ah,6h                       ;direct console output of DL
     int  msdos                       ;output the line coordinate
;
     mov  dl,bl                       ;ready the column number
```

```
    mov  ah,6h                     ;direct console output of DL
    int  msdos                     ;send the column coordinate
; the cursor is now at the location selected in BX
```

**2.4.3 Microsoft Pascal Compiler -- Direct Cursor Positioning**

```
program position (input,output);
{This method uses offsets from line 0, column 0.}

const
   clear_screen = chr(27) * chr(69);

var
   result : array[1..4] of char;
   i, line, column : integer
   row, col : char;

begin
   result[1] := chr(27);        {RESULT = ESC}
   result[2] := chr(89);        {RESULT = "Y"}
   write (clear_screen);
   write ('  Enter line (0-23) and column (0-79), as LINE COLUMN: ');
   readln (line, column);
   writeln (clear_screen);
   row := chr(32 + line);
   col := chr(32 + column);
   result[3] := row;            {RESULT = ROW}
   result[4] := col;            {RESULT = COL}
   for i := 1 to 4 do
      write (result[i]);        {PRINT CURSOR TO SCREEN}
   for i := 1 to 32000 do       {PAUSE}
end.
```

**2.5 Transmit Page -- Examples of Use**

The transmit page function is accessed by sending the ESC # sequence to the screen (see section 2.3.5). The result of this sequence is that all characters on the screen, as well as the cursor positioning sequences required to re-create the screen, are sent to the keyboard buffer. Reading the keyboard via a normal keyboard input request will return the entire screen of data to the program. The screen buffer within the program should be at least 1920 bytes (80x24) long to accommodate the entire screen - the program will need to perform 1920 single character inputs to empty the keyboard buffer. Note that the character input requests must be done rapidly to prevent the keyboard buffer overflowing and causing loss of data - note, too, that on a keyboard buffer overflow, the bell sounds.

The following sample programs demonstrate the use for this function request:

### 2.5.1 Microsoft MS-BASIC -- Transmit Page

```
10 DIM A$(1920)
20 PRINT CHR$(27)+"#";
30 FOR I = 1 TO 1920
40 A$(I)=INKEY$
50 NEXT I
60 PRINT CHR$(27)+"E";
70 FOR I = 1 TO 1920
80 PRINT A$(I);
90 NEXT I
```

### 2.5.2 Microsoft MACRO-86 Assembler -- Transmit Page

```
coniof        equ    6h                  ;direct console i/o function
conin         equ    0ffh                ;console input request
printf        equ    9h                  ;screen o/p up to $
msdos         equ    21h                 ;interrupt operating system
buffer_length    equ    1920             ;entire screen count

read_screen  db    1bh,'#$'              ;read entire screen
clear_screen db    1bh,'E$'              ;clear screen/home cursor
buffer       db    buffer_length dup (?) ;main buffer region

     mov     ax,DS                       ;get buffer data segment
     mov     ES,ax                       ;ready for store
     mov     di,offset buffer            ;get storage buffer
     mov     si,di                       ;init for later use
     mov     dx,offset read_screen       ;read entire screen string
     mov     ah,printf                   ;o/p it up to $
     int     msdos                       ;call the OS
;
; now read entire screen in to BUFFER
;
     mov     ah,coniof                   ;read from keyboard buffer
     mov     dl,conin                    ;ready to read
     mov     cx,buffer_length            ;count of chars to read
;
in_loop:
     int     msdos                       ;get a char in AL
     stosb                               ;save the char in BUFFER
     loop    in_loop                     ; and loop till buffer full
;
     mov     ah,printf                   ;ready to clear the screen
     mov     dx,offset clear_screen      ;get the string
```

```
    int     msdos                   ; and o/p it up to $
;
; now replace the screen data
;
    mov     cx,buffer_length        ;get the count
    mov     ah,coniof               ;get the o/p char function
;
out_loop:
    lodsb                           ;get a char
    mov     dl,al                   ;  ready to go
    int     msdos                   ;o/p it
    loop    out_loop                ;loop till buffer empty
    ret                             ;
```

### 2.5.3 Microsoft Pascal Compiler -- Transmit Page

```
PROGRAM Scrnbuf;

  CONST
    clear_screen   = CHR(27)*CHR(69)*CHR(36);
    transmit_page  = CHR(27)*CHR(35)*CHR(36);
    err_msg        = 'ERROR$';
    direct_conio   = #6;
    conin          = #0FF;
    print_string   = #9;

  VAR
    screen_dump : ARRAY [1..1920] OF CHAR;
    ch : CHAR;
    i : INTEGER;
    param : WORD;
    status : BYTE;

FUNCTION DOSXQQ( command, parameter : WORD ) : BYTE; EXTERNAL;

BEGIN
  EVAL(DOSXQQ(print_string,WRD(ADR(transmit_page) ) ) );
  param:= BYWORD( 0, conin );
  status:= DOSXQQ( direct_conio, param );
  IF status <> 0 THEN
    BEGIN
      i:= 1;
      WHILE status <> 0 DO
        BEGIN
          ch:= CHR(status);
          screen_dump[i]:= ch;
          i:= i + 1;
          status:= DOSXQQ( direct_conio, param );
```

```
        END;
      i:= i - 1;
      EVAL(DOSXQQ(print_string,WRD(ADR(clear_screen) ) ) );
      FOR VAR J:= 1 TO i DO
        EVAL(DOSXQQ( direct_conio, WRD(screen_dump[J]) ) );
    END
  ELSE
    EVAL(DOSXQQ(print_string,WRD(ADR(err_msg) ) ) );
END.
```

## 2.6 25th Line Display - Examples of Use

### 2.6.1 Microsoft MACRO-86 Assembler

```
; program name  SETKEY
;
; written by Greg Johnstone
;           Barson Computers
;           335 Johnston St
;           Abbotsford, 3067
;
; This  program displays function key tokens in the 25th line  of
; the  display  on the SIRIUS.  The program does NOT  affect  the
; codes attached to these keys (you must use KEYGEN in the Grafix
; or  Programmers  Toolkits  to do this);  NOR does it  read  the
; keyboard table to find out what the current code is.  All  this
; program  does,  is  to  display the data  contained  in  BUFFER
; (below) on the screen.  Ten characters are allowed for each key
; label.  You will need the programmers toolkit to  proceed.  Use
; MACRO86  to assemble this program then use LINK to link it,  as
; follows:
;
;
;              MACRO86 SETKEY;
;              LINK SETKEY;
;
; LINK  will  produce the message "warning:  no  stack  segment",
; ignore  it.  LINK produces a file ;  SETKEY.EXE which will  not
; run.  What  you must do is produce a .COM file using DEBUG,  as
; follows:
;              DEBUG SETKEY.EXE
;              N SETKEY.COM
;              W
;              Q
;
; The resulting program SETKEY.COM will run.
;
page
```

```
code    segment
        assume cs:code, ds:code

boot    equ     0               ;system reboot function
listout equ     5               ;list output function
conout  equ     2               ;console output function
;
cr      equ     0dh             ;carriage return
lf      equ     0ah             ;line feed
esc     equ     1bh             ;escape
;
        org     100h
start:
        mov     bx,offset buffer        ;point to output string
again:
        mov     ah,conout       ;set parameters for list output
        mov     dl,[bx]         ;get next character
        cmp     dl,0            ;test for end
        jz      cont
        push    bx
        int     21h             ;print it
        pop     bx
        inc     bx              ;next
        jmp     again
cont:
        mov     ah,boot         ;reboot
        int     21h
;
buffer:
        db      esc,'j'         ;save cursor
        db      esc,'xl'        ;enable 25th line
        db      esc,'Y',25+1fh,20h      ;put cursor in 25th line
        db      '  ',esc,'p'    ;turn on reverse mode
        db      'DIRECTORY '    ;label for key 1
        db      esc,'q ',esc,'p'
        db      'DISC COPY '    ;key 2
        db      esc,'q ',esc,'p'
        db      '  FORMAT  '    ;key 3
        db      esc,'q ',esc,'p'
        db      '  CHKDSK  '    ;key 4
        db      esc,'q ',esc,'p'
        db      '   EDIT   '    ;key 5
        db      esc,'q ',esc,'p'
        db      '  REPEAT  '    ;key 6
        db      esc,'q ',esc,'p'
        db      '  ABORT   '    ;key 7
        db      esc,'q','  '    ;end 25th line
        db      esc,'k'         ;put cursor back to saved position
```

```
        db      0

code    ends
        end     start
```

2.6.2 Microsoft MS-BASIC

```
10  '                *************************
15  '                *                       *
20  '                *  SET UP FOR 25TH LINE *
25  '                *                       *
30  '                *************************
35  '
40  WIDTH 255:PRINT CHR$(27);"E";TAB(28)"TEST FUNCTION KEY DISPLAY";
45  PRINT:PRINT:PRINT:INPUT "What is the Base Flag ";BASE
50  IF BASE<1 OR BASE>3 THEN RUN
60  GOSUB 100:GOSUB 200:GOTO 20
65  '
70  'variables       base    =   flags which display to print
75  '                fkct    =   number of keys to be set
80  '                fk$     =   array to hold key name
85  '                fksz    =   function key wording size
90  '
100 '*** subroutine to read key names as per base flag ***
110 E$=CHR$(27)
120 IF BASE = 1 THEN RESTORE 1010   rem restore data as per flag
130 IF BASE = 2 THEN RESTORE 1020
140 IF BASE = 3 THEN RESTORE 1030
150 READ FKCT                       rem read number of keys to set
160 FOR I=1 TO FKCT                 rem loop to read key names
170 READ FK$(I)                     rem set fk$(i) to name
180 NEXT
190 RETURN                          rem exit this subroutine
200 '*** subroutine to display key names on 25th line ***
210 FKSZ = INT((80-(FKCT-1))/FKCT)
220 X$="":C$=E$+"q"+" "+e$+"p"      rem c$= space in normal video
230 FOR I=1 TO FKCT                 rem loop to set display string
240 B$=LEFT$(FK$(I),FKSZ):J=INT((FKSZ-LEN(B$)+1)/2)
250 X$=X$+C$+LEFT$(SPACE$(J)+B$+SPACE$(FKSZ),FKSZ)
260 NEXT I
270 X$=E$+"p"+X$+E$+"q"             rem include on/off video
280 PRINT E$"x1";E$"j";e$"Y8 ";E$"1";X$;E$"Y  ";E$"k";:'print it
290 RETURN
1000 '*** data for key names ***
1010 DATA 7,"BASE1 KEY1","BASE1 KEY2","BASE1 KEY2","BASE1 KEY3","BASE1
KEY4","BASE1 KEY5","BASE1 KEY6","BASE1 KEY7"
1020 DATA 7,"BASE2 KEY1","BASE2 KEY2","BASE2 KEY3","BASE2 KEY4",
"BASE2 KEY5".BASE2 KEY6","BASE2 KEY7"
```

```
1030 DATA 7,"EVEN","NAMES","THAT","VARY","IN","LENGTH"," "
```

## 2.7 132-Column Display

Both WordStar and SuperCalc may be 'run' under the 132 Column display mode. You must, however, use the 'Install' programs of WordStar and SuperCalc to modify the screen dimensions for use in the 132 x 50 mode.

WordStar: To modify WordStar, simply type 'Install' and follow the prompts for customer terminal installation. When you get to the menu, follow the prompts to modify screen dimensions. You must change the screen to 50 lines and 132 columns (or less, if you desire).

SuperCalc: To modify SuperCalc, type 'Install' and select modify screen dimensions. You must change the lines to 38 and the column to 132 (or less, if you desire). A copy of SuperCalc 'Install' may be obtained from Barson Computers.

You may wish to copy or rename the modified program files (ie. WS132.CMD, SC132.CMD) so that you need not reconfigure the program each time you wish to change from 132 column to 80 column operation.

## SIRIUS 1 INPUT/OUTPUT PORT SPECIFICATION

### 3.1 Device Connection

Because of the 'soft' nature of the Sirius (that is, the configuration of the I/O ports is loaded from disc at boot time) you must always check that the operating system loaded is appropriate for the printer connected. If this is not the case then you may have to reconfigure the operating system permanently (using the System Configuration package) or temporarily using PORTSET and/or SETIO with MS-DOS or PORTCONF and/or STAT with CP/M-86. (see Section 3.6 for more details).

There are 5 ports (3 external, 2 internal) available on the Sirius 1 - they are as follows:

2 x Serial (RS232C) - Ports A and B
1 x Parallel (Centronics)
2 x Parallel (control - located on CPU board)

The ports are located on the rear of the Sirius 1 as shown in the following diagram:

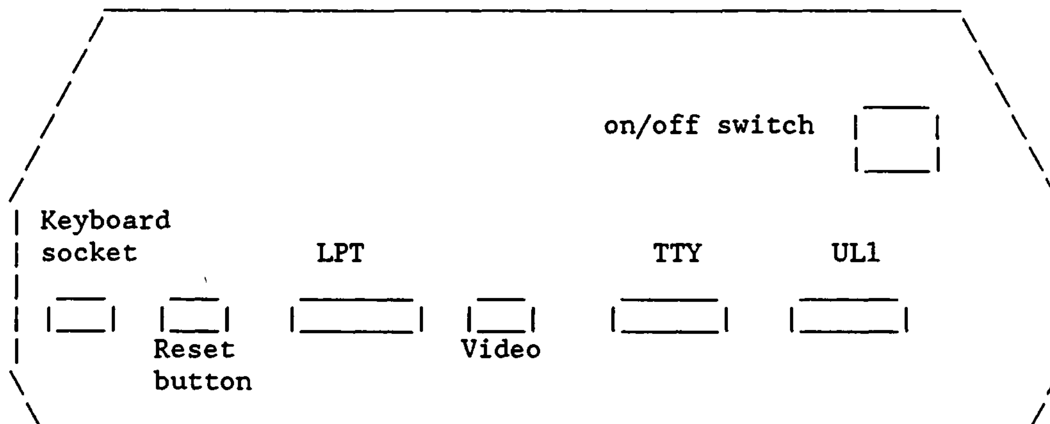



Figure 1
**Sirius 1 Parallel and Serial Ports**
(as viewed from the rear of the CPU)

## 3.2 Parallel Port Signals

Parallel printer interfacing is done on the parallel port: Parallel port interfacing is accessed through the 36 pin centronics compatible parallel port. Most parallel printer cable interfaces are simply 36-way flat ribbon cables with 36 pin male ribbon connectors at each end. The signals are directly routed from the Sirius 1 to the printer.

In parallel data transmission all data bits are transmitted asynchronously in an 8 bit parallel form. In parallel there is no parity checking or baud speeds.

| Pin Number | Signal |
|---|---|
| 1 | Data Strobe |
| 2 | Data 1 |
| 3 | Data 2 |
| 4 | Data 3 |
| 5 | Data 4 |
| 6 | Data 5 |
| 7 | Data 6 |
| 8 | Data 7 |
| 9 | Data 8 |
| 10 | ACK/ |
| 11 | Busy |
| 17 | Pshield |
| 12,18,30,31 | Not connected |
| Remaining | GND |

NOTE: For Epson printers pin 14 must not be connected (to eliminate double linefeeds.)

## 3.3 Parallel Printer Connection

To connect a parallel printer to the Sirius 1, a suitable cable is required - if the printer is supplied by Barson Computers, then it will be a matter of plugging the cable into both machines; cables should be attached as follows:

1) Disconnect power from both the computer and printer.
2) Disconnect the Sirius video connector (see 3.1)
3) Attach interface cable to Sirius and printer
4) Re-attach the video connector
5) Set the printer dip-switches as required
6) Make sure the operating system is configured for printing to the Centronics port by using SETIO (MS-DOS) or STAT (CP/M-86) as described in Section 3.6.

### 3.3.1 Parallel Cable Requirements

If a suitable parallel cable is not available, you will need to make one - use the guidelines that follow to create your own cable:

> You will need a male centronics-compatible Amphenol 57-30360 type connector for the Sirius 1 end of the cable; use the type of connector suggested by the printer manufacturer for the printer end, in general, another male centronics-compatible Amphenol 57-30360 type connector will be required. You will also require a length of 12-core cable (3m maximum length).

Refer to the port layout in your printer handbook - compare this with the Sirius 1 parallel port layout (see Section 3.2). If the pin numbers and signal requirements are the same, then construct the cable as follows:

```
 1 ------------------ 1
 2 ------------------ 2
 3 ------------------ 3
 4 ------------------ 4
 5 ------------------ 5
 6 ------------------ 6
 7 ------------------ 7
 8 ------------------ 8
 9 ------------------ 9
10 ------------------ 10
11 ------------------ 11
16 ------------------ 16
```

It does not matter which end of the cable is connected to the printer or the computer.

If your printer has the same signals as the Sirius 1, but on differing pins, then use the following guidelines:

1) Label one connector "Computer" and the other "Printer".
2) Connect pin 1 at the computer connector to the Data strobe pin at the printer connector.
3) Connect pins 2 through 9 at the computer connector to the Data1 (may be labelled Data0) through Data8 (may be labelled Data7) at the printer connector.
4) Connect pin 10 at the computer connector to the ACK pin at the printer connector.
5) Connect pin 11 at the computer connector to the BUSY pin at the printer connector.

6) Connect pin 16 at the computer connector to the GROUND (may be labelled GND) pin at the printer connector.

The printer cable is now complete - it must always be attached to the devices as marked on the connectors - if it is not, then the printer will not work.

### 3.4 Serial Port Signals

The two serial ports may be used to connect serial printers to the Sirius 1. In serial transmission data bits are transmitted over a data line one bit at a time. The industry standard for serial peripheral communication is the RS-232C serial binary data interchange.

The RS-232C standard is usable for data interchange rates up to 20,000 bits per second.

**Line Driver Output Voltages**

| State | Approx. Voltage |
|---|---|
| OFF | -12 VDC |
| GND | +/- 0 VDC |
| ON | + 12 VDC |

The RS-232C maximum recommended cable length is approximately 15metres. On the Sirius 1, serial printers may be hooked up to the following serial port pins.

SIRIUS SERIAL (RS232) PORT

| PIN | NAME | FUNCTION | DIRECTION |
|---|---|---|---|
| 1 | FG | Frame Ground | - |
| 2 | TD | Transmitted Data | From SIRIUS |
| 3 | RD | Received Data | To SIRIUS |
| 4 | RTS | Request to Send | From SIRIUS |
| 5 | CTS | Clear to Send | To SIRIUS |
| 6 | DSR | Data Set Ready | To SIRIUS |

| | | | |
|---|---|---|---|
| 7 | SG | Signal Ground | - |
| 8 | DCD | Data Carrier Detect | To SIRIUS |
| 15 | TC | Transmitter Clock | To SIRIUS |
| 17 | RC | Receiver Clock | To SIRIUS |
| 20 | DTR | Data Terminal Ready | From SIRIUS |
| 22 | RI | Ring Indicator | To SIRIUS |
| 24 | TTC | Transmitter Clock | From SIRIUS |

There are basically two classifications of data transceiving equipment, 1. Data Terminal Equipment (DTE) and 2. Data Communications Equipment (DCE). Most printers are set up as DTE's. The interface requirements for the two classifications are very different, so the classification of printer and computer must be determined before a cable interface can be designed. The Sirius 1 is configured as a (DTE).

(DCE) -------------------------------- (DTE)
(DTE) -------------------------------- (DTE) *

* Note: Typical Sirius 1 computer to printer interface.

### Signal Descriptions

PIN 1 CHASSIS GROUND
This signal is electrically connected to the machine frame.

PIN 2 TRANSMIT DATA
Data is transferred across this line. This line will be held in a marking (OFF) state during intervals between characters or words and when no data is being transmitted.

PIN 3 RECEIVE DATA
The printer receives data on this line. This line is held in a marking (OFF) state whenever data carrier detect (DCD PIN 8) is in the (OFF) state.

PIN 4 REQUEST TO SEND
This signal is used to prepare the printer to receive data. The (ON) state indicates to the printer that the DTE has data to be transmitted.

The (OFF) condition indicates the computer is in a non-transmit mode.
When the (RTS) line is transitioned from OFF to ON the printer knows to prepare to receive data. The printer responds to the RTS transition by transitioning the clear to send (CTS) line from off to on.

PIN 5 CLEAR TO SEND

Signals on this circuit indicate whether or not the data set is ready to transmit data. The ON condition is a response to the occurrence of a simultaneous ON condition on data set ready (DSR) and request to send (RTS).
In DTE's where RTS is not implemented RTS shall be assumed to be ON at all times, and CTS will respond accordingly.

PIN 6 DATA SET READY

This circuit is used to indicate the status of the local data set.

PIN 7 SIGNAL GROUND

PIN 8 DATA CARRIER DETECT

This signal issued by the DTE tells whether the data being received is of suitable quality. The (ON) state indicates suitable data is being received. The (OFF) state indicates that no data is being received or that the data being received is unsuitable.

PIN 20 DATA TERMINAL READY

This signal is used to switch the DTE to the communications channel. The ON condition prepares the DTE to be connected to the channel and maintains the connection established.

In serial printers the data is transmitted as single bits. The number of bits per second (for this application) is the baud rate. The computer and printer baud rates MUST match in order to maintain proper operation.

NOTE: Mismatched baud rates between printer and computer will cause the printer to print improper characters (garbage).

The data bit stream can be checked for accuracy by the use of parity bits. The parity bit is added to the transmitted data frame and decoded when received. The computer and printer baud rate must be equal for correct system operation.

The printer serial port may not conform to RS-232C pin configurations, thus the pinouts for the PRINTER serial port must be obtained before a serial interface can be developed.

### 3.5 Serial Printer Connection

To connect a serial printer to the Sirius 1, a suitable cable is required - if the printer is supplied by Barson Computers, then it will be a matter of plugging the cable into both machines; cables should be attached as follows:

1) Attach the cable between the Sirius 1 serial port B (see 3.1) and the printer connector.

2) Set the printer switches for 8-data bits, 1 stop bit, 4800 baud and no parity. Set DTR protocol (refer to printer manual).

You may set the baud rate at a rate different from that mentioned in (2) - but you will then be required to set the baud rate using the baud rate selection utility, PORTSET or PORTCONF (see 3.6), or alternatively you will need to build a new operating system. (see Programmer's Toolkit).

Make certain that the operating system is configured for printing to the serial port by using SETIO (MS-DOS) or STAT (CP/M-86) as described in Section 3.6.

#### 3.5.1 Serial Cable Requirements

If a suitable serial cable is not available, you will need to make one - use the guidelines that follow to create your own cable:

You will require 1 x D25 male, 1 x D25 female connectors, and a length of 6-12 core cable, with a maximum length of about 15m. Refer to the port layout in your printer manual, if pin 3 is received data (labelled RXD or RD), and pin 20 is data terminal ready (labelled DTR), then construct your cable as follows:

```
        Computer                 Printer

           1 --------------------- 1
           2 --------------------- 3
           3 --------------------- 2
           7 --------------------- 7
           5 --------------------- 20
```

This cable, often called a Modem Eliminator Cable, must be attached as shown - mark the Computer/Printer connectors as a reference.

With some printers you can add an extra connection:

```
          20 -------------------- 5
```

which allows the cable to be used with either end connected to the computer.

If pin 3 is receive data (RXD or RD) and pin 20 is not data terminal ready (DTR) then construct your cable as follows:

```
        Computer                 Printer

           1 --------------------- 1
           3 --------------------- 2
           2 --------------------- 3
           7 --------------------- 7
           5 --------------------- x
```

where x is the pin number of the BUSY signal (possibly pin 11 or 19). If this method is used, make sure that the polarity of the BUSY signal is correct; this is usually switch selectable (it should be LOW when printer is BUSY).

This cable must be attached as shown - mark the Computer/Printer connectors as a reference.

If the printer requires pins 6 and 8 to be held high then add the following connection:

```
          20 --------------------- 6
                                |----- 8
```

See Section 3.4 for details of Sirius serial port pinouts and Appendix C for further sample cables.

## 3.6 Operating System Port Utilities

Victor Technologies supplies a selection of programs under both CP/M-86 and MS-DOS to allow the temporary selection of both baud rate and list device port. If you attach a printer to your system you may be required to perform some of the following steps in order to use the printer. Before you use any of the utilities discussed you need to be aware of the port the printer is attached to; Port A, B or Parallel. You will also need to know, except in the case of a parallel printer, what baud rate, stop-bits and parity your printer is set up at. Note that many printers will start to lose data at baud rates above 4800, you must, therefore, select a baud rate that your printer can handle.

Ideally your operating system should be configured permanently using the system generation packages. You must tell the operating system which port you intend using and what baud rate the printer requires (if using a serial port).

The CP/M-86 System Configuration package is available separately and the MS-DOS System Generation package is found in the Programmer's Toolkit.

If you wish to temporarily change printer port you can use STAT (CP/M-86) or SETIO (MS-DOS) and the baud rate can be changed using PORTSET, PORTCONF or simple Basic programs.

### 3.6.1 SETIO - MS-DOS List Device Selection Utility

SETIO is a utility program to display or change the I/O byte. The I/O byte associates a logical device with a physical device.

SETIO has three modes. If invoked without parameters, the assignment table is displayed. If invoked with an invalid device assignment, the command format is displayed with the assignment table. If invoked with a valid device assignment, an updated table is displayed with the new assignment.

To select the correct port for the list device you have attached, the SETIO program has been provided. This program is used as follows:

```
SETIO LST = TTY   - printer is attached to port A
SETIO LST = UL1   - printer is attached to port B
SETIO LST = LPT   - printer is attached to parallel port
```

It is recommended that your printer be attached to either port B or the parallel port.

Once SETIO has executed, it displays a map of the ports, with the ones you selected highlighted on the screen - if this is not correct, repeat the process.

Examples:(highlighted fields are enclosed in brackets []).

```
A>setio <cr> (without parameters)

Logical Device                    Physical Devices
     CON                           TTY[CRT]BAT UL1
     AUXIN                        [TTY]PTR UR1 UR2
     AUXOUT                       [TTY]PTP UP1 UP2
     LST                           TTY CRT LPT[UL1]


A>setio ?<cr> (note: ? is an invalid parameter)

SET I/O VERSION n.n
usage: SETIO[<logical device> = <physical device>]

     CON                           TTY[CRT]BAT UL1
     AUXIN                        [TTY]PTR UR1 UR2
     AUXOUT                       [TTY]PTP UP1 UP2
     LST                           TTY CRT LPT[UL1]

A>setio lst = tty<cr> (valid parameters)

SET I/O VERSION n.n

     CON                           TTY[CRT]BAT UL1
     AUXIN                        [TTY]PTR UR1 UR2
     AUXOUT                       [TTY]PTP UP1 UP2
     LST                          [TTY]CRT LPT UL1
```

In this last example, we have set the printer port to port A (which usually comes set at 1200 baud).

**Logical and Physical Devices**

| Device Type/Name | Description |
|---|---|
| Logical Devices | |
| CON | Console device - the principal interactive console which communicates |

| | |
|---|---|
| | with the operator. Typically, CON: is a device such as a CRT or teletype. |
| LST | List device - the principal listing device; usually a hard-copy device, such as a printer or teletype. |
| AUXIN | Auxiliary input device. |
| AUXOUT | Auxiliary output device. |

Physical Devices

| | |
|---|---|
| TTY | Serial output-port A (teletype-style printer - RS232C) |
| CRT | Keyboard and cathode ray tube display. |
| LPT | Parallel port printer (Centronics). |
| UL1 | Serial printer - port B (RS232C). |
| BAT | Batch mode-reader as input; (AUXIN) a printer (LST) as output. |
| UC1 | External console (to be developed). |
| PTR | High speed read (to be developed). |
| UR1 | (to be developed). |
| UR2 | (to be developed). |
| PTP | High speed punch (to be developed). |
| UP1 | (to be developed). |
| UP2 | (to be developed). |

Examples:

| | |
|---|---|
| SETIO LST=LPT | Direct printer output to the centronics port. |
| SETIO LST=CON | direct printer output to the console (good for debugging software without wasting paper). |
| SETIO CON=TTY | redirect console I/O to port A. (this |

enables an external terminal to be connected to the Sirius, but note that the Sirius' own screen and keyboard will be inoperable).

### 3.6.2 STAT - CP/M-86 List Device Selection Utility

To select the correct port for the list device you have attached, the STAT program has been provided. This program is used as follows:

```
STAT LST:=TTY:   - printer is attached to port A
STAT LST:=UL1:   - printer is attached to port B
STAT LST:=LPT:   - printer is attached to parallel port
```

It is recommended that your printer be attached to either port B or the parallel port.

**SIRIUS 1 Device Name Assignment for CP/M-86**

**CP/M Physical Device Name**

| | |
|---|---|
| TTY: | Serial Output Port A |
| CRT: | Keyboard and Display CRT |
| UC1: | External Console (reserved) |
| PTR: | High Speed Read (reserved) |
| UR1: | (Reserved) |
| UR2: | (Reserved) |
| PTP: | High Speed Punch (reserved). |
| UP1: | (Reserved) |
| UP2: | (Reserved) |
| LPT: | Parallel Port (Centronics) |
| UL1: | Serial Printer - Port B |

**CP/M Logical Device Name**

| | | |
|---|---|---|
| CON: | Logical Console device | |
| | Typical Assignment: | CRT: |
| | Assignment options: | TTY:, UC1:, BAT: (see below) |
| LST: | Logical List device | |
| | Assignment options: | LPT:, UL1:, TTY:, CRT: |
| RDR: | Logical Reader Device | |
| | Assignment options: | TTY:, PTP:, UP1:, UP2: |
| PUN: | Logical Punch device | |
| | Assignment options: | TTY:, PTP:, UP1:, UP2: |

BAT: Batch mode reader (RDR:) as input, a printer (LST:) as output.

**Logical Device Characters**

CONSOLE The principal interactive console which communicates with the operator, accessed through CONST, CONIN, and CONOUT. Typically, the CONSOLE is a device such as a CRT or teletype.

LIST The principal listing device, if it exists on your system, is usually a hard-copy device such as a printer or teletype.

PUNCH The principal tape punching device, if it exists, which is normally a high-speed paper tape punch or teletype.

READER The principal tape reading device, such as a simple optical reader or teletype.

**IOBYTE Field Definitions**

CONSOLE field (bits 0,1) (CON:)

0 - console is assigned to the console printer (TTY:)
1 - console is assigned to the CRT device (CRT:)
2 - batchmode: use the READER as the CONSOLE INPUT, and the LIST device as the CONSOLE output (BAT:)
3 - user defined console device (UC1:)

READER field (bits 2,3) (RDR:)

0 - READER is the Teletype device (TTY:)
1 - READER is the high-speed reader device (PTR:)
2 - user defined reader #1 (UR1:)
3 - user defined reader #2 (UR2:)

PUNCH field (bits 4,5) (PUN:)

0 - PUNCH is the teletype device (TTY:)
1 - PUNCH is the high-speed punch device (PTP:)
2 - user defined punch #1 (UP1:)
3 - user defined punch #2 (UP2:)

LIST field (bits 6,7) (LST:)

0 - LIST is the teletype device (TTY:)
1 - LIST is the CRT device (CRT:)
2 - LIST is the line printer device (LPT:)
3 - user defined list device (UL1:)

**3.6.3 PORTSET - MS-DOS Baud Rate Selection Utility**

To select the correct baud rate for ports A or B (but this is not applicable to the parallel port), the PORTSET program is provided. This program is menu driven, and is used as follows:

To the prompt type PORTSET, the screen will display a choice of three ports:

1) Port A (RS232C)
2) Centronics/Parallel Port
3) Port B (RS232C)

Type either 1,2 or 3. If you type 1 or 3, the next menu screen is displayed - this screen has baud-rate choices labelled A through N - select one of the baud-rates.

**3.6.4 PORTCONF - CP/M-86 Baud Rate Selection Utility**

This program is used in exactly the same manner as PORTSET (see 3.6.3).

**3.7 Serial Input/Ouput Port Addresses**

The two serial input/output ports are memory mapped ports located in the memory segment E000hex; and they are mapped as follows:

E000:40 - port A data (input/output)
E000:41 - port B data (input/output)
E000:42 - port A control (read/write)
E000:43 - port B control (read/write)

The following information is available in each port's control register:

bit 0 - rx character available
bit 1 - not used
bit 2 - tx buffer empty
bit 3 - DCD
bit 4 - not used

bit 5 - CTS
bit 6 - not used
bit 7 - not used

See Section 3.4 for information on each port's pinouts.

Note that writing a 10hex to the relevant control register allows the resensing of the modem leads (i.e. DCD and CTS) with their current values being updated in the port's control register.

Since the Sirius 1 configures the NEC 7201 chip to operate in auto-enable mode, DCD (pin 8 on the port connector) must be ON, and CTS (pin 5 on the port connector) must be ON to enable the 7201's receiver and transmitter respectively. RTS and DTR are always ON as a convenient source for an RS-232C control ON (+12 volts).

### 3.8 Baud Rate and Data Input/Output - Sample Programs

The means of establishing the baud rates, receiving and transmitting data are discussed in the following programs. The serial port's control register are discussed in 3.7 - the means of accessing them is better described with the programming examples that follow.

The following programs provide information on how to set up the baud rates on the serial ports (A and B) - they also demonstrate how to send and receive data from these ports.

#### 3.8.1 Microsoft MS-BASIC -- Baud Rate and Data Input/Output

The following program may be used in place of PORTSET or PORTCONF if you omit the lines 500 through 740 inclusive.

```
10 DIM RATE(14)
20 REM Select the data port
30 PRINT CHR$(27)+"E"; : REM Clear the screen
40 PRINT : PRINT : PRINT : PRINT
50 PRINT "The serial ports are:" : PRINT
60 PRINT ,"          A - Serial Port TTY - left hand on back"
70 PRINT ,"          B - Serial Port UL1 - right hand on back"
80 PRINT : PRINT
90 PRINT ,"Select the port you want to use, A or B ";
100 PORT$ = INPUT$(1)
110 PRINT PORT$
120 IF PORT$ = "a" THEN STATIO=2 : DATIO=0 : GOTO 210
130 IF PORT$ = "A" THEN STATIO=2 : DATIO=0 : GOTO 210
```

```
140 IF PORT$ = "b" THEN STATIO=3 : DATIO=1 : GOTO 210
150 IF PORT$ = "B" THEN STATIO=3 : DATIO=1 : GOTO 210
160 GOTO 30
200 REM Set the baud rate
210 PRINT CHR$(27)+"E"; : REM Clear the screen
220 PRINT : PRINT : PRINT : PRINT
230 PRINT "The available baud rates are as follows:" : PRINT
240 PRINT ," 1 =     300 baud"
250 PRINT ," 2 =     600 baud"
260 PRINT ," 3 =    1200 baud"
270 PRINT ," 4 =    2400 baud"
280 PRINT ," 5 =    4800 baud"
290 PRINT ," 6 =    9600 baud"
300 PRINT ," 7 =   19200 baud"
310 PRINT : PRINT : PRINT
320 PRINT "Select one of the above baud rates: ";
330 RATE$ = INPUT$(1)
340 IF RATE$ > "7" THEN 210
350 IF RATE$ < "1" THEN 210
360 PRINT RATE$
400 REM Now set the baud rate in the port selected
410 DEF SEG = &HE002
420 IF DATIO = 0 THEN POKE 3,54 : IF DATIO = 1 THEN POKE 3,118
430 FOR I = 1 TO 14
440 READ RATE(I) : REM Set the baud rate matrix
450 NEXT I
460 NODE = (VAL(RATE$)-1)*2+1
470 POKE DATIO,RATE(NODE)
480 POKE DATIO,RATE(NODE+1)
500 REM Now data may be entered and sent down line
510 PRINT CHR$(27)+"E"; : REM Clear the screen
520 PRINT : PRINT ,"Baud rate established"
530 PRINT : PRINT : PRINT
540 DEF SEG = &HE004
550 PRINT ,"Enter data to be sent down line with return to end"
560 PRINT ,"or just press return to receive data -"
570 PRINT
580 TEXT$=INKEY$
590 IF TEXT$="" THEN 630
600 IF TEXT$=CHR$(13) THEN PRINT TEXT$ :TEXT$=CHR$(126) :GOTO 620
610 PRINT TEXT$;
620 GOSUB 650
630 GOSUB 690
640 GOTO 580
650 STATUS=PEEK (STATIO) : STATUS=STATUS AND 4
660 IF STATUS = 0 THEN 650 :REM Waiting to send char
670 POKE DATIO, ASC(TEXT$)
680 RETURN
```

```
690 STATUS = PEEK(STATIO) :STATUS = STATUS AND 1
700 IF STATUS = 0 THEN RETURN : REM No char available
710 DATUM = PEEK (DATIO) : DATUM = DATUM AND 127
720 IF DATUM = 126 THEN PRINT CHR$(13) : RETURN
730 PRINT CHR$(DATUM); :REM Show char from line
740 RETURN
1000 DATA 04,1,&H82,0,&H41,0,&H20,0,&H10,0,8,0,4,0
```

The above program may be used to send characters between two Sirius'. Use the following cable connection.

**CABLE SIRIUS TO SIRIUS**

```
 1 ------------------- 1
 2 ------------------- 3
 3 ------------------- 2
 7 ------------------- 7
 5 ----- |       |----- 5
 8 ----- |       |----- 8
20 ----- |       |----- 20
```

### 3.8.2 MACRO-86 Assembler -- Baud Rate and Data Input/Output

The following assembler modules may be included in a program and called with the stated parameters. The character input and output modules will need re-coding if your program requires status return rather than looping for good status.

```
rates   db      04h,1h,82h,0h     ;baud rate conversion table
        db      41h,0h,20h,0h
        db      10h,0h,8h,0h
        db      4h,0h

;**********************************************************
;
; Routine:      BAUD_SET
;
; Function:     To set Port A or B baud rate
;
; Entries:      AL = 0=PortA, 1=PortB
;               DX = 0=300 baud,  1=600 baud,  2=1200 baud
;                    3=2400 baud, 4=4800 baud, 5=9600 baud
;                    6=19200 baud
;
; Returns:      None
;
; Corruptions: ES, AX, BX, CX, DX
;
```

```
;*********************************************************

baud_set:
        mov     cx,0e002h               ;get the segment
        mov     ES,cx                   ;init the segment register
        mov     bx,3                    ;point to counter control
        or      al,al                   ;see if Port A or B to be set
        jnz     set_B                   ;AL > 0, so set Port B counter
;
        mov     byte ptr ES:[bx],36h    ;set it for port A
        jmp     short set_rate          ;  and input the Baud rate
;
set_B:
        mov     byte ptr ES:[bx],76h    ;set port B counter
;
set_rate:
        mov     bx,offset rates         ;get the baud rate table
        shl     dx,1                    ;DX = DX * 2 for words
        add     bx,dx                   ;point to baud rate entry
        mov     dx,[bx]                 ;get the baud rate
        xor     bh,bh                   ;BH=0
        mov     bl,al                   ;get the required port
        mov     byte ptr ES:[bx],dl     ;send first byte
        mov     byte ptr ES:[bx],dh     ; and last byte of rate
        ret                             ;baud rate established

;*********************************************************
;
; Routine:      SEND_CHAR
;
; Function:     To output a character to a serial port
;
; Entries:      AL = 0=PortA, 1=PortB
;               AH = Character to send
;
; Returns:      None
;
; Corruptions: ES, AX, BX
;
;*********************************************************

send_char:
        mov     bx,0e004h               ;get the port segment
        mov     ES,bx                   ;set the segment
        xor     bh,bh                   ;BH=0
        mov     bl,al                   ;get the required port
        add     bl,2                    ;required port status
;
```

```
in_status_loop:
        mov     al,ES:[bx]              ;get the status
        and     al,4h                   ;mask for TX empty
        jz      in_status_loop          ;not ready - loop
;
        sub     bl,2                    ;point to data
        mov     ES:[bx],ah              ;character gone
        ret

;**********************************************************
;
; Routine:      GET_CHAR
;
; Function:     To input a character from a serial port
;
; Entries:      AL = 0=PortA, 1=PortB
;
; Returns:      AL = character
;
; Corruptions: ES, AX, BX
;
;**********************************************************

get_char:
        mov     bx,0e004h               ;get the port segment
        mov     ES,bx                   ;set the segment
        xor     bh,bh                   ;BH=0
        mov     bl,al                   ;get the required port
        add     bl,2                    ;required port status
;
out_status_loop:
        mov     al,ES:[bx]              ;get the status
        and     al,1h                   ;mask for RX character avail
        jz      out_status_loop         ;not ready - loop
;
        sub     bl,2                    ;point to data
        mov     al,ES:[bx]              ;character received
        ret
```

## 3.9 Transferring Files to and from another computer

ASCII files may be transferred to and from another computer using the serial communications port (DB25 connector nearest the video connector).

Using the CP/M or MS-DOS system generation package, configure a system with the serial ports set to the required baud rates, stop and parity bits. (Avoid baud rates above 2400).

Make up a cable to connect the serial port to the other computer. Consult the specifications of the other computer carefully. The connection diagram for the MT-180 printer cable has been used successfully to communicate with other computers. Also strapping pins 4, 5 and 8 has been used with success.

Using PIP treat the communications port as the logical paper tape reader (RDR:) and punch (PUN:). (to run PIP under MS-DOS, use the CP/M-86 emulator).

To transfer a file into Sirius start with the following command:

```
PIP CON:=RDR:
```

to see if Sirius is receiving. If not receiving, check your connection cable, try swapping connections to pins 2 and 3. Make sure all signals going to the Sirius are correct.

If the above command produces weird characters (when transmitting ASCII) then it is possible that the top bit is set on some bytes (WordStar files do this, Sirius displays a 256 character set), try the following command:

```
PIP CON:=RDR:[Z]
```

which zeros the top bit.

Otherwise, if you get normal characters bearing little relationship to the original file, check the baud rates.

When communication is established, transfer data to a named file:

```
PIP FILENAME=RDR:
```

Remember to send an EOF (^Z) from the other computer so that PIP knows it has finished. Limit the size of the files transferred to under 32K (the size of PIP's buffer) otherwise characters will be lost.

If you wish to transfer binary files you will have to encode the binary data into ASCII. The problem with binary is that ^Z may be valid data. Certainly the [O] option can be used for transferring binary files internally but externally there is no way for PIP to know that the transfer is complete.

An alternative method of transferring files is to use the

VT52 emulation package and use the Sirius as a terminal onto another computer. This package allows the transfer of files to and from the host.

There is also some software available from Barson Computers which allows the Sirius to emulate a printer. Thus, the other computer need only list to the Sirius.

### 3.10 Sirius 1 IEEE-488 Port

The Sirius 1 IEEE-488 cable attaches to the parallel port - the pin number refers to the actual computer port connector; the IEEE-488 pin number refers to the standard IEEE-488 pin-out as they must attach to the parallel port.

The IEEE pin numbers referred to with the (**z) are wires that are to be bound together as twisted pairs.

An IEEE 488 operating system must first be created using the IEEE 488 Toolkit and the Programmer's Toolkit. The necessary IEEE 488 system files are copied to the system generation disc found in the Programmer's Toolkit and used to generate the operating system.

| Sirius Pin Number | IEEE Signal | IEEE Pin Number | |
|---|---|---|---|
| 1 | DAV | 6 | (**a) |
| 19 | GND | 18 | (**a) |
| 2 | DIO1 | 1 | |
| 3 | DIO2 | 2 | |
| 4 | DIO3 | 3 | |
| 5 | DIO4 | 4 | |
| 6 | DIO5 | 13 | |
| 7 | DIO6 | 14 | |
| 8 | DIO7 | 15 | |
| 9 | DIO8 | 16 | |
| 10 | NRFD | 7 | (**b) |
| 28 | GND | 19 | (**b) |
| 11 | SRQ | 10 | (**c) |
| 29 | GND | 22 | (**c) |
| 13 | NDAC | 8 | (**d) |
| 33 | GND | 20 | (**d) |
| 15 | EOI | 5 | |
| 17 | shield | 12 | |
| 34 | REN | 17 | |
| 35 | ATN | 11 | (**e) |
| 16 | GND | 23 | (**e) |
| 36 | IFC | 9 | (**f) |

27 ----------- GND ------------- 21 (**f)
20 ----------- GND ------------- 24

### 3.11 Sirius 1 Control Port (internal port)

| Pin Number | Signal |
|---|---|
| 1 | -12V |
| 2 | -12V |
| 3 | Not connected |
| 4 | Not connected |
| 5 | +12V |
| 6 | +12V |
| 7 | +5V |
| 8 | +5V |
| 9 | Not connected |
| 10 | Light Pen |
| 11 | GND |
| 12 | CA1 |
| 13 | GND |
| 14 | CA2 |
| 15 | GND |
| 16 | PA0 |
| 17 | GND |
| 18 | PA1 |
| 19 | GND |
| 20 | PA2 |
| 21 | GND |
| 22 | PA3 |
| 23 | GND |
| 24 | PA4 |
| 25 | GND |
| 26 | PA5 |
| 27 | GND |
| 28 | PA6 |
| 29 | GND |
| 30 | PA7 |
| 31 | GND |
| 32 | PB0 |
| 33 | GND |
| 34 | PB1 |
| 35 | GND |
| 36 | PB2 |
| 37 | GND |
| 38 | PB3 |
| 39 | GND |
| 40 | PB4 |
| 41 | GND |

```
42 --------------- PB5
43 --------------- GND
44 --------------- PB6
45 --------------- GND
46 --------------- PB7 / CODEC Clock Output
47 --------------- GND
48 --------------- CB1
49 --------------- GND
50 --------------- CB2
```

### 3.12 MS-DOS Logical Devices

As explained in the MS-DOS operating system manual, certain 3-letter file names are reserved for I/O devices:

- AUX refers to input from or output to an auxiliary device connected to serial port A (TTY).
- CON refers to keyboard input or output to the screen (CRT).
- LST refers to the printer which may be redirected to any of the I/O ports using SETIO.
- NUL does not refer to a particular file or device. NUL is used when the syntax of a command requires an input or output file name. NUL is sometimes referred to as a 'bit-bucket'. Any output to NUL will be lost, but will not cause the system to 'hang'. It is useful for debugging programs.

Examples of use:

1. At the Command level.

    ```
    COPY FILENAME AUX
    ```

    This command will transmit the file 'FILENAME' to the device connected to port A (the TTY device).

2. At the Command level.

    ```
    COPY CON LST
    ```

    This command will cause any keyboard input to be sent to the printer rather than the screen.

3. In Basic.

    ```
    10 OPEN "O",#1,"AUX"
    ```

```
20 PRINT #1,"XYZ"
30 CLOSE #1
```

This program will cause the string 'XYZ' to be sent to the device connected to serial port A. Note that the string is only sent after the file is closed or the internal buffer is full.

Using this technique, it is possible to toggle output between the printer (LST) and a device, such as another printer or a plotter, connected to port A (AUX).

4. In Basic.

```
10 INPUT "Do you have a printer attached (Y/N)",A$
20 IF A$="Y" OR A$="y" THEN F$="LST" ELSE F$="CON"
30 OPEN "O",#1,F$
40 PRINT #1,.......
```

This example demonstates the ability to send output to the screen if no printer is present. Useful for debugging.

5. In Basic.

An MS-BASIC demonstration program (only to be run under MS-DOS V1.25 BIOS 2.5 or later) allows the operator to choose the destiny of the output. The output can be to a file, or to a logical device. The logical devices are "CON" (=console), "LST" (=primary list device), or "AUX" (=auxiliary port).

In this example the "AUX" logical device is being used as a secondary printer.

```
10  PRINT CHR$(27)+"E"
20  PRINT CHR$(27)+"Y"+CHR$(41)+CHR$(35);
30  PRINT"WHERE IS THE DESTINATION OF THE LISTING TO BE?"
40  PRINT CHR$(27)+"Y"+CHR$(44)+CHR$(45);
50  PRINT"CONSOLE,   PRIMARY LIST DEVICE,  AUXILIARY PORT"
60  PRINT CHR$(27)+"Y"+CHR$(46)+CHR$(35);
70  PRINT"PLEASE TYPE IN F,C,P,or A"
80  PRINT CHR$(27)+"Y"+CHR$(52)+CHR$(60);
90  INPUT A$
100 IF A$<>"F" THEN IF A$<>"C" THEN IF A$<>"P" THEN IF A$<>"A" THEN 80
110 IF A$="C" THEN A$="CON"
120 IF A$="P" THEN A$="LST"
130 IF A$="A" THEN A$="AUX"
140 IF A$="F" THEN INPUT "FILE SPECIFIER";A$
150 CLOSE
```

```
160 PRINT CHR$(27)+"E"
170 OPEN "O",#1,A$              'OPEN A SEQUENTIAL FILE
180 FOR I=1 TO 20               'CREATE TEST DATA
190 B$="TEST DATA 1234567890 abcdefghijklmnopqrstuvwxyz"
200 PRINT #1,B$
210 NEXT I
220 CLOSE
230 SYSTEM
```

### 3.13 Sample Program for Initialising Printers

```
;
; PROGRAM NAME -- INIT
;
; Program to print a string to the printer. You will need the
; Programmers Toolkit and a suitable editor to proceed. Use the
; editor to generate or alter this program and store it in the
; file INIT.ASM.
;
; Change the string 'AAAAAA' at 'BUFFER' to the string you want.
; (This string can be any length you want)
;
; Use MACRO86 to assemble the program then use LINK to link it
; as follows:
;
;       MACRO86 INIT;
;       LINK INIT;
;
; LINK will produce an error 'Warning: No STACK segment', ignore it.
; The file INIT.EXE produced by LINK will not run. You must use
; DEBUG to generate INIT.COM as follows:
;
;       DEBUG INIT.EXE
;       N INIT.COM
;       W
;       Q
;
; The resulting program INIT.COM will run
code    segment
assume  cs:code, ds:code

boot    equu    0
listout equ     5               ; list output function
bdos    equ     21h             ; DOS function call
;
cr      equ     0dh             ; carriage return
lf      equ     0ah             ; line feed
;
```

```
        org     100h
start:
        mov     bx,offset buffer          ; point to output string
again:
        mov     ah,listout        ; set parameters for list output
        mov     dl,[bx]           ; get next character
        cmp     dl,0              ; test for end
        jz      cont
        push    bx
        int     bdos              ; print it
        pop     bx
        inc     bx                ; next
        jmp     again

cont:
        mov     ah,boot           ; reboot
        int     bdos
;
buffer  db      'AAAAAA'          ; place initialisation string here
        db      cr,lf
        db      0                 ; end of print string

code    ends
        end
```

# MS-DOS NOTES

## 4.1 MS-DOS PROGRAM LOAD

The operating system core provides no direct means to run user programs. Instead, to run a given program represented by a disc file, the file must be opened and read into memory using the normal system functions. These functions are requested by the user program that is currently running.

The first user program to run is the initialisation routine that follows a system boot, which normally loads and executes the file COMMAND.COM. This is a user program that accepts commands from the console and translates them into system function calls. COMMAND includes the capability to load and execute other program files; when these other programs terminate, COMMAND regains control. Thus COMMAND is responsible for the initial conditions that are present when a program is executed.

A standard set of initial conditions is provided by COMMAND on entry to another program. It is possible for programs other than COMMAND to load and execute program files, and they must also provide the same initial conditions so that a consistent interface may be assumed by the newly executing program.

### 4.1.1 MS-DOS Base Page Structure (see also Section 4.4)

The MS-DOS Base Page (sometimes called the Program Segment Prefix or PSP), is created when you enter an external command. COMMAND.COM will allocate a memory region to the external program, and will insert the Base Page prior to the origin of this program.

In the memory segment that the program is to load, COMMAND.COM places a Base Page, COMMAND.COM then loads the program at an offset of 100H, and hands over control to the external program. The external program, once its function is complete, hands control back to the operating system by a far JUMP or far RETURN to location zero within the Base Page; the instruction at this location is an INT 20, or return control to MS-DOS. This stage must be executed to allow MS-DOS to recover memory correctly (see Appendix I).

When an external program is loaded, the following conditions are true:

The file control blocks at Base Page locations 5CH and 6CH are created from the first two parameters entered on the

command line.

The command line at Base Page location 80H is created from the command line entered AFTER the program filename. The byte at location 80H contains the command line character count, the following bytes contain the raw command line as entered at the keyboard.

The word at offset 6; in the Base Page contains the number of bytes available in the segment.

The contents of register AX are established to reflect the validity of the drive(s) on the command line. Thus the following may be found:

AL = FFH when the first drive letter on the command line was not recognised by MS-DOS.

AH = FFH when the second drive letter on the command line was not recognised by MS-DOS.

The above applies equally to both .EXE and .COM type files. The EXE and .COM files do have differences when they load, and these are described more fully below.

When .EXE files load:

The contents of register DS and register ES are pointing at the Base Page segment address.

The registers CS, IP, SS and SP are initialised to those

values passed by the linker.

When .COM files load:

The contents of registers CS, DS, ES and SS are pointing to the Base Page segment address.

The register IP is set at 100H.

The register SP is set the high address in the program segment, or to the base of the transient portion of COMMAND.COM, whichever is the lower. The contents of the word at Base Page offset 6 are decremented by 100H to allow for a stack of that size.

A word of zeros is placed at the top of the stack.

All four segment registers have the same value, and the corresponding absolute memory address is the base of a "program segment". The program is loaded and begins execution at location 100 hex in the program segment. Other assignments in the program segment are:

00-01 Termination point. Contains an interrupt type 20 hex, which returns control to the originating program. Thus a JMP 0 or INT 20H are the normal ways to terminate a program.

02-03 Memory size in paragraphs. End of current allocation block contains the first segment number after the end of memory.

05-09 Far CALL to MS-DOS function dispatcher.

0A-0B Program terminate address as IP and CS.

0E-11 Address as CS and IP.

22-5B Default stack. The stack pointer is initially 5A hex, with a word of zeros on the top. Thus executing a "return" instruction will cause a transfer to location 0 and the program will terminate normally. This stack may be used as-is, or a new one may be set up. remember that 32 bytes of stack space are required to perform system calls.

5C-67 File Control Block #1, formatted as normal unopened FCB.

6C-77 File Control Block #2, formatted as normal unopened FCB.

80-FF Unformatted parameters. Count of characters on command line; followed by command line entered.

COMMAND prepares the parameter areas from the console input line that specified the program to be executed. For example, if COMMAND sees a line of the form

<progname> <file1> <file2>

this is a request to execute the file <progname>.COM. <file1> and <file2> each may or may not include a disc specifier or a file name extension, but in any case they appear in the formatted parameters at 5C hex and 6C hex. In addition, the entire input

line after the last letter of <progname> appears in the unformatted parameter area beginning at 81 hex, with the number of characters placed at 80 hex.

Suppose the input line is:

```
COPY T.BAK B:TEST.ASM
```

The formatted parameter at 5C hex will contain:

```
00 "T       BAK"
```

At 6C hex will be:

```
02 "TEST    ASM"
```

And at 80 hex will be:

```
17 " T.BAK B:TEST.ASM"
```

where the 17 is decimal.

```
CGROUP                  NAME      BASEPAGE
DGROUP                  GROUP     CODE
code                    GROUP     DATA
                        SEGMENT   PUBLIC 'CODE'
                        ASSUME    CS:CGROUP,DS:GROUP

                        EXTRN     MAIN : NEAR
                        PUBLIC    End_of_program

Start_of_program        PROC      NEAR
                        MOV       BX,DS ;Hold base page segment
                        MOV       AX,DGROUP
                        MOV       DS,AX  ;Fix DS to point to data
                                  group
                        MOV       Base_page_ptr+2,BX ;Save base page
                                  seg.
                        CALL      MAIN ;Execute program
End_of_program:                   ;Can jump here to end, if unable to
                                  ;RET
                        JMP       DWORD PTR [Base_page_ptr]

Start_of_program        ENDP

code                    ENDS
data                    SEGMENT   PUBLIC 'DATA'
                        PUBLIC    Base_page_ptr
```

```
Base_page_ptr       DD        0

data                ENDS

                    end       START_OF_PROGRAM
```

Above is a sample "base" page for a type .EXE file for MS-DOS, which executes a program starting at "MAIN". Note the fix up required for DS. When program starts DS/ES point to DOS related base. (Note, this is not required for .COM files ... for them DS, ES, CS are O.K. and IP is at 100h).

## 4.2 The Command Processor

### 4.2.1 Introduction

The command processor supplied with MS-DOS (file COMMAND.COM) consists of three distinctly separate parts:

1. A resident portion resides in memory immediately below the BIOS (see Section 1.6.1). This portion contains routines to process interrupt types 22H (end address), 23H (CTRL-C handler), 24H (critical error handling) and 27H (end but stay resident), as well as a routine to reload the transient portion if needed. (When a program ends, a checksum determines if the program had caused the transient portion to be overlaid. If so, it is reloaded). Note that all standard MS-DOS disc error handling is done within this portion of COMMAND. This includes displaying error messages and interpreting the reply of Abort, Retry, or Ignore.

2. An initialisation portion is given control during startup. This section contains the AUTOEXEC file processor setup and also the date prompt routine (used if no AUTOEXEC file is found). The initialisation portion determines the segment address at which programs can be loaded. It is overlaid by the first program COMMAND loads because it is no longer needed.

3. A transient portion is loaded below the resident portion. This is the command processor itself, containing all of the internal command processors, the batch file processor, and a routine to load and execute external commands (files with filename extensions of .COM or .EXE). This portion of COMMAND produces the system prompt (such as A>), reads the command from the keyboards (or batch file) and causes it to be executed. For external commands, it builds a Program Segment Prefix control block, loads the program named in the

command into the segment just created, sets the end and CTRL-C exit address (interrupt vectors 22H and 23H) to point to the resident portion of COMMAND, then gives control to the loaded program.

Note: Files with an extension of .EXE which are designated to load into high memory are loaded immediately below the transient portion of COMMAND to prevent the loading process from overlaying COMMAND itself.

Section 4.3 contains information describing the conditions in effect when a program is given control by COMMAND.

### 4.2.2 Replacing the Command Processor

Though the command processor is an important part of MS-DOS, its functions may not be needed in certain environments. Therefore, it has been designed as a user program to allow its replacement. If you decide to replace it with your own command processor;

1. Name your program file COMMAND.COM.

2. The entry conditions are the same as for all .COM programs.

3. Be sure to set the end and CTRL-C exit addresses in the interrupt vectors and in your own Program Segment Prefix to transfer control to your own code.

4. You must provide code to handle (and set the interrupt vectors for) interrupt types 22H (end address), 23H (CTRL-C handler), 24H (critical error handling) and if needed 27H (end but stay resident). Your COMMAND.COM is also responsible for reading commands from the keyboard and loading and executing programs, if needed.

### 4.2.3 Available MS-DOS Functions

MS-DOS provides a number of functions to user programs, all available through issuance of a set of interrupt codes. There are routines for keyboard input (with and without echo and CTRL-C detection), console and printer output, constructing file control blocks, memory management, date and time functions, and a variety of diskette and file handling functions. See MS-DOS Interrupts and Function Calls in Programmer's Toolkit for detailed information

### 4.2.4 Diskette/File Management Notes

Through the INT 21H (function call) mechanism, MS-DOS provides methods to create, read, write, rename and erase files. Files are not necessarily written sequentially on diskette - space is allocated one sector at a time as it is needed, and the first sector available is allocated as the next sector of a file being written. Therefore, if considerable file creation and erasure activity has taken place, newly created files will probably not be written in sequential sectors.

However, due to the mapping (chaining) of file sectors via the File Allocation Table, and the fields defined in the File Control Block, any file can be used in either a sequential or random manner. By using the current block and current record fields of the FCB and the sequential disc read or write functions, you can make the file appear sequential - MS-DOS will do the calculations necessary to locate the proper sectors on the diskette. On the other hand, by using the random record field and random disc functions, you can cause any record in the file to be accessed directly - again. MS-DOS will locate the correct sectors on the diskette for you. Among the most powerful functions are the random block read and write functions which allow reading or writing a large amount of data with one function call - this is how MS-DOS loads programs. As above, MS-DOS will handle locating the correct sectors on diskette to provide the image of sequential processing - you need not be concerned about the physical location of data on diskette.

### 4.2.5 The Disc Transfer Area (DTA)

The Disc Transfer Area (also commonly called "buffer") is the memory area MS-DOS will use to contain the data for all disc reads and writes. This area can be at any location within memory, and should be set by your program. (See function call 1AH).

Only one DTA can be in effect at a time, so it is the program's responsibility to inform MS-DOS what memory location to use before using any disc read or write functions. Once set, MS-DOS continues to use that area for all disc operations until another function call 1AH is issued to define a new DTA. When a program is given control by COMMAND a default DTA has already been established at 80H in the program's Program Segment Prefix large enough to hold 128 bytes.

### 4.2.6 Error Trapping

MS-DOS provides a method by which a program can receive control whenever a disc read/write error occurs, or when a bad memory image of the file allocation table is detected. When these events occur, MS-DOS executes an INT 24H to pass control to the error handler. The default error handler resides in COMMAND.COM but any program can establish its own by setting the INT 24H vector to point to the new error handler. MS-DOS provides error information via the registers and provides Abort, Retry or Ignore support via return codes. (See MS-DOS Interrupts and Function Calls in the Programmer's Toolkit).

Unlike the end and CTRL-C exit addresses, MS-DOS does not preserve the original contents of the critical error exit address when a program is given control. It is your program's responsibility to preserve the original contents (two words) of the INT 24H vector prior to setting this vector, and to restore the original contents before ending.

### 4.2.7 General Guidelines

The following guidelines and tips should assist in developing applications using the MS-DOS disc read and write functions.

1. All disc operations require a properly constructed FCB that the program must supply.

2. Remember to set the Disk Transfer Area address (function 1AH) before performing any reads or writes to a file.

3. All files must be opened (or created, in the case of a new file) before being read from or written to. Files which have been written to must also be closed to ensure accurate directory information.

4. A program may define its own logical record size by placing the desired size into the FCB. MS-DOS then uses that value to determine a record's location within the file. If using the "file size" function call, this field must be set by the calling program prior to the function call. If using the disc read and write routines, the field should be set after opening (or creating) the file but before any read or write functions are used. (Open function sets the field to a default value of 128 bytes).

5. New files must be created (function call 16H) before they can be written to. This call creates a new directory entry and opens the file.

6. If the amount of data being transferred is less than one sector (512 bytes), MS-DOS will "buffer" the data for the requesting program in an internal buffer within BIOS. Because there is only one disc buffer, performing less-than-sector-size operations in a random manner or against multiple files concurrently causes MS-DOS to frequently change the contents of the buffer. If such operations are in output mode, this forces MS-DOS to write a partially full sector to make the buffer available for any other diskette operation. Subsequently, the partially full sector would have to be re-read before further data could be written to the file. This is called "thrashing" and can be very time consuming. To remedy this situation, use of the Random block read and write routines is recommended, with a data transfer size as large as possible. (An entire file can be read this way, provided enough memory exists.) This method bypasses the "buffering" described above, by reading or writing directly to or from the DTA for as much of the data as possible. If the file size is not a multiple of 512 bytes, only the last portion of the file (the portion past the last 512-byte multiple) is buffered by MS-DOS.

### 4.2.8 Examples of Using MS-DOS Functions

This example illustrates the steps necessary for a program named TEST.COM to:

1. Create a new file named FILE1.

2. Load and execute a second program named PGM1.COM from the diskette in drive B.

The program is in a file named TEST.COM and was invoked from the keyboard by the command TEST FILE1 B:PGM1.COM.

When the program (TEST) received control the Program Segment Prefix has been set up as shown in section 4.3. The end and CTRL-C exit addresses in the Program Segment Prefix are the ones which the host (calling program) had established and should not be modified - they are restored to interrupt 22H and 23H vectors when this program ends. The FCBs at 5CH and 6CH are formatted to contain file names of FILE1 and PGM1.COM respectively - the first

FCB reflects the default drive and the second drive B. The default DTA is set to 80H into the segment (the unformatted parameter area of the Program Segment Prefix).

#### 4.2.9 To Create File FILE1

Because it is known that the data in the FCB at 6CH is needed to load and execute the program whose name it contains in a subsequent step that FCB must be preserved: opening the FCB at 5CH would cause it to be overlayed. The program should:

1. Copy the FCB at 6CH to an area within itself.

2. Using the FCB at 5CH call function 11H to be sure FILE1 does not already exist - if it did exist, it would be overwritten by this program.

3. Assuming it did not exist, create the file (function call 16H) - the file is now open.

4 Set the FCB current record and random record fields to zero, and the record size field to the desired size.

5. Build the memory image of the file's data.

6. Set the DTA to point to the memory image (function call 1AH).

7. Use the sequential write (15H), random write (22H), or random block write (28H) calls to write the file, ensuring the FCB fields and DTA are set properly for each call. In the case of call 28H (the preferred method) the entire file can be written with one call by setting CX to the number of records to be written (in terms of the FCB record size field).

8. Close the FCB at 5CH - the directory and file allocation table are updated and any partial data in MS-DOS's disc buffer (if it were performing blocking) are written to disc.

To Load and Execute Program PGM1.COM from drive B.

Assume that the current program (TEST) wished to control the action taken if CTRL-C is entered. (Until now, the CTRL-C address still pointed to COMMAND.COM which would end program TEST if CTRL-C were pressed).

TEST should:

1. Set the end and CTRL-C exit vectors (call 25H) to point to code within itself (the end address is where the program to be loaded will return when it ends).

2. Determine where PGM1.COM should reside in memory and set up a segment for it, including a Program Segment Prefix (call 26H). This copies the end and CTRL-C exit addresses just set into the new segment's Program Segment Prefix.

3. Set the DTA to offset 100H into the just-created segment. (Be sure the DS register contains the correct segment address). This is the offset at which PGM1.COM will be loaded.

4. Open the FCB that had been copied earlier (for PGM1.COM). The FCB file size field will be filled in by open to a default value of 128 bytes.

5. Set the FCB record size field to the desired size. (Setting it to 1 is very useful in this case).

6. Set the CX register to the number of records (based on the record size field) to read. If the record size was set to 1, then the number of records to read does not have to be computed - it can be obtained directly from the FCB file size field. In any case, if the product of the record size field and contents of the CX register are equal to or greater than the file size, then the entire file is read in the following step.

7. Read the file using the Random Block Read function (call 27H) into the new segment at offset 100H. (See step 3 above). There is no need to close the file since it was not written to.

8. Prepare the DS, ES, SS and SP registers for the loaded program and push a word of zeros on the top of its stack.

9. Set the DTA to offset 80H into the new segment.

10. Give control to the loaded program. (An intersegment jump is ideal, since it does not use stack space). When the called program ends via INT 20H, MS-DOS

restores interrupt vectors 22H and 23H from the values in the ending program's Program Segment Prefix (the values established in step 1) and pass control to the end exit address. TEST is now back in control, and can itself issue an INT 20H which will cause its caller (COMMAND.COM) to regain control.

Note: The example above was simplified by not discussing the checking of return codes from the function calls. Nearly all function calls do return exception or error indications, which should be checked by the calling program.

## 4.3 MS-DOS Diskette Directory

FORMAT builds the directory for each diskette on track 0 sectors 3-10, a total of 4096 bytes. The directory has room for 128 entries, each 32 bytes long. Each directory entry is formatted as follows. (Byte offsets are in decimal).

0-7 Filename. (E5H in byte 0 means this directory entry is not used.)

8-10 Filename extension.

11 File attribute. Contents can be 02H for a hidden file and 04H for a system file. (Both files are excluded from all directory searches unless an extended FCB with the appropriate attribute byte is used). For all other files this byte contains 00H. A file can be designated as hidden when it is created.

12-21Reserved

22-23Time

```
<        24       > <       22   >
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
 h  h  h  h  h  m m m m m m s s s s s
```

24-25Date the file was created or last updated. The mm/dd/yy are mapped in the bits as follows:

```
<        25       > <       24   >
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
 y  y  y  y  y  y y m m m m d d d d d
```

where:
yy is 0-119 (1980-2099)

mm is 1-12
dd is 1-31

26-27 Starting sector: the relative sector number of the first block in the file. (For file allocation purposes only, relative sector numbers start at 000 with track 0 sector 6. This is in contrast with DEBUG and the absolute disc read write routines, interrupts 25H and 26H which number relative sectors from the beginning of the diskette.)

The relative sector number is stored with the least significant byte first.

28-31 File size in bytes. The first word contains the low-order part of the size. both words are stored with the least significant byte first.

## 4.4 MS-DOS Program Segment

When you enter an external command, the COMMAND processor (see also Section 4.1.1) determines the lowest available address (immediately after the character fonts, see Section 1.6.1) to use as the start of available memory for the program invoked by the external command. This area is called the Program Segment.

At offset 0 within the Program Segment. COMMAND builds the Program Segment Prefix control block. (See section 4.1.) COMMAND loads the program at offset 100H and gives it control. (.EXE files can be loaded into high memory just below the transient portion of COMMAND.COM but the Program Segment Prefix will still be in low memory.)

The program returns to COMMAND by a jump to offset 0 in the Program Segment Prefix (The instruction INT 20 is the first item in the control block) by issuing an INT 20, or by issuing an INT 21 with register AH=0

NOTE: It is the responsibility of all programs to ensure that the CS register contains the segment address of the Program Segment Prefix when ending via any of these methods.

All three methods result in an INT 20 being issued, which transfers control to the resident portion of COMMAND.COM. It restores interrupt vectors 22H and 23H (end and CTRL-C exit addresses) from the values saved in the Program Segment Prefix of the ending program. Control is then given to the end address. (If this is a program returning to COMMAND, control transfers to its transient portion.) If a batch file was in process, it is

continued: otherwise, COMMAND issues the system prompt and waits for the next command to be entered from the keyboard.

When a program receives control, the following conditions are in effect.

For all programs:

- o Disk transfer address (DTA) is set to 80H (default DTA in the Program Segment Prefix).
- o File control blocks at 5CH and 6CH are formatted from the first two parameters entered when the command was invoked.
- o Unformatted parameter area at 51H contains all the characters entered after the command name (including leading and embedded delimiters with 80H set to the number of characters).
- o Offset 6 (one word) contains the number of bytes available in the segment. If the resident portion of COMMAND.COM is within the segment its value is reduced by its size.
- o Register AX reflects the validity of drive specifiers entered with the first two parameters as follows:
  - AL=FF if the first parameter contained an invalid drive specifier (otherwise AL=00).
  -AH=FF if the second parameter contained an invalid drive specifier (otherwise AH=00).

For .COM programs:

- o All four segment registers contain the segment address of the Program Segment Prefix control block.
- o The Instruction Pointer (IP) is set to 100H.
- o SP register is set to the end of the program's segment or the bottom of the transient portion of COMMAND.COM, whichever is lower. The segment size at offset 6 is reduced by 100H to allow for a stack of that size.
- o A word of zeros is placed on the top of the stack.

For .EXE programs:

- o DS and ES registers are set to point to the Program Segment Prefix. (See below).
- o CS,IP, SS and SP registers are set to the values passed by the linker.

# MISCELLANEOUS PROGRAMMING NOTES

## 5.1 ROUNDING NUMBERS IN BASIC-86

The following is a short program to illustrate one technique for rounding numbers to any number of decimal places, (within the limits of the machine).

```
10  DEFINT N
20  DEFDBL X
30  INPUT "ENTER NUMBER TO BE ROUNDED";X
40  IF X<1 OR X>7 GOTO 110
50  INPUT "ENTER NUMBER OF DECIMAL PLACES";N
60  A$=STR$(X+.5*10^-N*SGN(X))
70  NO=INSTR(A$,".")
80  X1=VAL(LEFT$(A$,N+NO))
90  PRINT "X=;X,"X1=";X1
100 GOTO 30
110 PRINT "END"
120 END
```

Rounding to zero decimal places can of course be achieved by using the INT function.

Rounding numbers to be output to the screen, printer or file can be achieved by the use of PRINT USING.

The routine given above can easily be simplified if it is always required that numbers be rounded to a fixed number of decimal places (eg. 2).

## 5.2 Undocumented Commands and Functions of the BASIC86 Interpreter

### 5.2.1 DATE$

The DATE$ function returns a ten character string variable that has the following format:

MM-DD-YYYY

where:

MM = month (1 - 12)
DD = day (1 - 31)
YYYY = year (1980 - 2099)

The delimiter is either - or /.

The date can be set through BASIC86, when doing so, the leading zeros are presumed for single months and days. The year can be entered as a 2 digit number.

Examples:

```
PRINT DATE$
DATE$="10-6-83"
DATE$="10/06/83"
DATE$="10-06-83"
DATE$=VAR$
```

### 5.2.2 TIME$

The TIME$ function returns an eight character string variable, 2 of which are delimiters. It takes the following format:

```
HH:MM:SS
```

where:

HH = hour of day (0-23)
MM = minutes (0-59)
SS = seconds (0-59)

delimiter is :

The Time can be altered within BASIC86, when doing so minutes and seconds are optional and so are leading zeros. The time is being constantly incremented by a hardware clock.

Examples:

```
PRINT TIME$
TIME$="15:10:40"
TIME$="7:5"
TIME$="8"
NOW$="14:15:06"
TIME$=NOW$
```

### 5.2.3 DATE

The DATE variable contains the numbers of days since the beginning of the year. (Does not work on compiler)

Example:

```
PRINT DATE
```

### 5.2.4 TIME

The TIME variable contains the number of seconds since midnight (00:00:00). (Does not work on compiler)

Examples:

```
PRINT TIME
```

The following is a self-timing program:

```
10 X=TIME
20 FOR I=1 TO 10000
30 NEXT I
40 PRINT TIME-X
```

### 5.2.5 BLOAD

Format:

```
BLOAD <filespec>[,<offset>]
```

Purpose: Loads a memory image into memory.

<filespec> is a string expression returning a valid file specification as described in "Input and Output," except for the extension. If the device name is omitted, the current diskette drive is assumed. The only valid extensions are:

| | |
|---|---|
| (none) | (no extension) |
| .B | for Basic programs in the internal format (created with the SAVE command). |
| .P | for protected Basic programs in the internal format (created with SAVE ,P command). |
| .A | for Basic programs in ASCII format (created with SAVE ,A command). |
| .M | for memory image files (created with BSAVE command). |

.D for data files (created by OPEN followed by output statements.

<offset> is a numeric expression in the range 0 to 65535. This is the address at which the loading is to start, specified as an offset into the segment declared by the last DEF SEG statement. If the <offset> is omitted, the <offset> specified in the last BSAVE is assumed.

WARNING: BLOAD does not perform address range checking. That is, it is possible to BLOAD anywhere in memory! You should be absolutely sure you are not overwriting the operating system, Basic, or your own program.

EXAMPLE

```
10 'LOAD AN ASSEMBLY PROGRAM INTO BASIC DS ASSUMING
20 'NO PROGRAM HAS BEEN LOADED.
30 DEF SEG    'SET THE DATA SEGMENT TO BASIC'S
40 BLOAD "MOVE",0   'LOAD THE CALLABLE PROGRAM.
```

### 5.2.6 BSAVE

Format:

BSAVE <filespec>,<offset>,<length>

Purpose:

Allows you to save portions of the computer's memory on the specified device.

<filespec> is a string expression returning a valid file specification as described in BLOAD.

<offset> is a numeric expression in the range 0 to 65535. This is the address at which the saving is to start, specified as an offset into the segment declared by the last DEF SEG statement.

<length> is a valid numeric expression returning an unsigned integer in the range 1 to 65535. This is the length of the memory image to be saved.

Example:

```
10 'save the first 100 bytes of memory located
```

```
20 'at the start of Basic's Data Segment.
30 DEF SEG
40 BSAVE "PROGRAM.M",0,100
```

### 5.2.7 OPEN

In addition to the syntax described in the MS-BASIC manual, MS-BASIC supports the more powerful GW-BASIC syntax described in the Graphics Toolkit.

Format:

```
OPEN [<dev>] <filename> [FOR <mode>] AS[#]
<file number>[LEN=<lrecl>]
```

Purpose:

The OPEN statement establishes addressability between a physical device and an I/O buffer in the data pool.

Remarks:

<dev> is optionally part of the file name string and conforms to the description in section 3.12.

<filename> is a valid string literal or variable optionally containing a <dev>. If <dev> is omitted, disc A: is assumed. Disc file names follow the normal MS-DOS naming conventions.

<mode> determines the initial positioning within the file and the action to be taken if the file does not exist. The valid modes and actions taken are:

| | |
|---|---|
| INPUT | Position to the beginning of an existing file. The "File Not Found" error is given if the file does not exist. |
| OUTPUT | Position to the beginning of the file. If the file does not exist, one is created. |
| APPEND | Position to the end of the file. If the file does not exist, one is created. |

If the FOR <mode> clause is omitted, the initial position is at the beginning of the file. If the file is not found, one is created. This is the Random I/O mode. That is, records can be read or written at will at any position within the file.

<file number> is an integer expression returning a number in the range 1 through 15. The number is used to associate an I/O buffer with a disc file or device. This association exists until a CLOSE <file number> or CLOSE statement is executed.

<lrecl> is an integer expression in the range 2 to 32768. This value sets the record length to be used for random files. If omitted, the record length defaults to 128-byte records.

When a disc file is OPENed FOR APPEND, the position is initially at the end of the file and the record number is set to the last record of the file (LOF(x)/128). PRINT, WRITE or PUT then extends the file. The program can position elsewhere in the file with a GET statement. If this is done, the mode is changed to random and the position moves to the record indicated.

Once the position is moved from the end of the file, additional records can be appended to the file by executing a GET #x,LOF(x)/<lrecl>.

### 5.3 An Example of Calling an Assembler routine from the MS-BASIC Compiler (For an example of calling an Assembler routine from the Interpreter, see Section 10.5)

The Assembler routine is coded and written to disc as a .ASM FILE using EDLIN or PMATE (fig 1). This is then assembled using MACRO-86 (fig 2).

The Basic program is coded and written to disc with the BASIC interpreter (ie. 'SAVED' with the ',A' option to produce an ASCII text file), (fig 3). This is then compiled using BASCOM (fig 4).

The object (.OBJ) modules produced by MACRO-86 and BASCOM are then linked using MS-LINK (fig 5). The Assembler object module must precede the Basic object module in the Link statement.

The resultant Run file (.EXE) may then be run simply by typing the name given to it during the link.

When writing the 'CALL' statement in BASIC the name used (lines 160, 220, and 280) must be the name that appears in the 'PUBLIC' statement in the assembler code and must also be the label used in the assembler code 'PROC' statement.

See also section 10.5 and chapter 11 for further examples.

```
    b:asmex.asm
NAME        EXAMPLE
CGROUP      GROUP       CODE
CODE        SEGMENT PUBLIC 'CODE'
ASSUME      CS:CGROUP,DS:CGROUP
PUBLIC      PFKDI
PFKDI       PROC        FAR
            PUSH        DS              ;save BASIC DS
            MOV         AX,CS           ;set-up DS for this module
            MOV         DS,AX
            MOV         DX,OFFSET LINE
            MOV         AH,9
            INT         21H
            POP         DS              ;restore BASIC DS & return to
            RET                         ;caller
LINE        DB          1BH,78H,31H
            DB          1BH,59H,38H,20H
            DB          20H,20H,1BH,70H
            DB          ' F KEY 1 '
            DB          1BH,71H,20H,20H,1BH,70H
            DB          ' F KEY 2 '
            DB          1BH,71H,20H,20H,1BH,70H
            DB          ' F KEY 3 '
            DB          1BH,71H,20H,20H,1BH,70H
            DB          ' F KEY 4 '
            DB          1BH,71H,20H,20H,1BH,70H
            DB          ' F KEY 5 '
            DB          1BH,71H,20H,20H,1BH,70H
            DB          ' F KEY 6 '
            DB          1BH,71H,20H,20H,1BH,70H
            DB          ' F KEY 7 '
            DB          1BH,71H,20H,20H,20H
            DB          1BH,48H
            DB          1BH,79H,31H
            DB          1BH,45H
            DB          '$'
PFKDI       ENDP
CODE        ENDS
            END
```

FIG. 1 SAMPLE ASSEMBLY MODULE

```
A>MACRO86 B:ASMEX;
The Microsoft MACRO Assembler
Version 1.00 (C)Copyright Microsoft 1981

Warning Severe
Errors  Errors
0       0
```

**FIG 2. INVOCATION OF MACRO-86 TO PRODUCE .OBJ FILE**

```
          b:msbex.bas
100  FOR X=1 TO 10
110  PRINT "THIS IS TEST LINE A 1234567890"
120  PRINT "THIS IS TEST LINE B 1234567890"
130  PRINT "THIS IS TEST LINE C 1234567890"
140  PRINT "THIS IS TEST LINE D 1234567890"
150  PRINT "THIS IS TEST LINE E 1234567890"
160  CALL PFKDI
170  FOR D=1 TO 250 : NEXT
180  PRINT "THIS IS TEST LINE F 0987654321"
190  PRINT "THIS IS TEST LINE G 0987654321"
200  PRINT "THIS IS TEST LINE H 0987654321"
210  PRINT "THIS IS TEST LINE I 0987654321"
220  CALL PFKDI
230  FOR D=1 TO 250 : NEXT
240  PRINT "THIS IS TEST LINE J ABCDEFGHIJ"
250  PRINT "THIS IS TEST LINE K ABCDEFGHIJ"
260  PRINT "THIS IS TEST LINE L ABCDEFGHIJ"
270  PRINT "THIS IS TEST LINE M ABCDEFGHIJ"
280  CALL PFKDI
290  FOR D=1 TO 250 : NEXT
300  NEXT
```

**FIG. 3. SAMPLE BASIC MODULE**

```
A>bascom b:msbex;

Microsoft BASIC Compiler
Version 5.32
(C)Copyright Microsoft Corp 1982

24241 Bytes Available
23463 Bytes Free

     0 Warning Error(s)
     0 Severe  Error(s)
```

FIG. 4. INVOCATION OF BASIC COMPILER TO PRODUCE .OBJ FILE

```
A>link b:asmex+b:msbex

  Microsoft Object Linker V1.08
(C) Copyright 1981 by Microsoft Inc.

Run file [A:ASMEX.EXE]: b:msbatst.exe
List file [NUL.MAP]: b:msbatst.map
Libraries [.LIB]:
```

FIG. 5. INVOCATION OF MS-LINK TO PRODUCE .EXE FILE

### 5.4 Program Size Limitations

The following limitations apply to program size as indicated:

| | |
|---|---|
| CBASIC | 62K |
| MS-BASIC INTERPRETER | 62K |
| MS-BASIC COMPILER | 64K code<br>64K data |
| MS PASCAL | 64K object code<br>64K default data segment<br>32767 lines of source code |
| MS FORTRAN | 64K object code<br>64K each named common block<br>64K all local variables<br>32767 lines of source code |

In FORTRAN and Pascal you can compile any number of compilands separately and link them together later; the real

limit on program size is thus determined by the capability of the linker (MS-LINK) which is approximately 386K.

In CBASIC and both MS-BASIC's, multiple programs may be linked together using the CHAIN command.

### 5.4.1 Memory Usage outside the default 64K segment in MSBASIC

The following routine is an example of the use of extra memory. The DEF SEG statement defines a segment address for POKEing to.

In this case, the address chosen is the start of the second memory board.

```
10  DEF SEG=&H2000
20  FOR I%=0 TO 20000
30  POKE I%,65
40  NEXT I%
50  FOR I%=0 TO 20000
60  J%=PEEK(I%)
70  A$=CHR$(J%)
80  PRINT "BYTE NUMBER ";I%;" - VALUE =";A$
90  NEXT I%
100 END
```

### 5.4.2 Memory Usage outside of the default 64K segment in MS-Pascal

A technique to access more than 64K of data space in MS-Pascal is to use the ADS facility of the language implementation.

Due to the storage mechanism of the language it is impossible to change segment under program control as in BASIC or to use named common blocks as in FORTRAN.

ADS gives a method of creating, manipulating and dereferencing actual machine addresses.

Examples of ADS are given in the Pascal Reference Manual, (Section 8, in V3.04 "Reference Types") and a more complete program listing using ADS and ADR is given on the following page.

NOTE. Great care must be taken with the use of this facility as you do deal with actual memory locations and, as no memory map is available for use at run-time the decision on where to store variables etc.... requires a great deal of thought.

```
TYPE B:MEM64.PAS
{
     PROGRAM  TO TEST THE ALLOCATION OF STORAGE AREAS OUTSIDE  OF  THE
     DEFAULT 64K SEGMENT OF MEMORY.

     DETAILS  OF  THE  FUNCTIONS USED ARE FOUND IN SECTION  6  OF  THE
     SIRIUS MS-PASCAL MANUAL.
                                                                     }

PROGRAM MEM64(INPUT,OUTPUT);

VAR
     INT_VAR : INTEGER;
     REAL_VAR : REAL;
     A_INT : ADR OF INTEGER;        {relative   machine   address   of
                                    integer}
     AS_REAL : ADS OF REAL;         {segmented machine address of real}
BEGIN
     INT_VAR :=1;
     REAL_VAR :=3.1415;
     A_INT : ADR INT_VAR;           {A_INT=relative  machine address of
                                    INT_VAR}
     AS_REAL :=ADS REAL_VAR         {AS_REAL=segmented machine  address
                                    of REAL_VAR}
     WRITELN (A_INT^,AS_REAL^);     {write  values pointed at by  A_INT
                                    and AS_REAL}
     AS_REAL.S := 16#0006;          {change  AS_REAL segment pointer to
                                    0006 HEX}
     AS_REAL^ :=REAL_VAR;           {load address pointed at by AS-REAL
                                    with REAL_VAR}
     REAL_VAR := 9.81;
     WRITELN (REAL_VAR,AS_REAL^);   {write   REAL_VAR  and  the   value
                                    pointed at by AS_REAL}
     AS_REAL.S := 16#0007;          {change AS_REAL segment pointer  to
                                    0007 HEX}
     AS_REAL^ := REAL_VAR;          {load address pointed at by AS_REAL
                                    with REAL_VAR}
     REAL_VAR := 1.0;
     WRITELN  (REAL_VAR,AS_REAL^);  {write   REAL_VAR  and  the   value
                                    pointed at by AS_REAL}
     AS_REAL.S := 16#0008;          {etc.........     }
     AS_REAL^ := REAL_VAR;
     REAL_VAR := 0.15;
     WRITELN (REAL_VAR,AS_REAL^)
END.
{    THIS  PROGRAMME IS BASED ON THE EXAMPLE GIVEN ON PAGE 6.38 OF THE
     PASCAL REFERENCE MANUAL.                                         }
```

```
            B:MEM64
                 1 3.1415000E+00
            9.8099990E+00 3.1415000E+00
            1.0000000E+00 9.8099990E+00
            1.5000000E+01 1.0000000E+00
         A>
```

### 5.4.3 Memory usage outside of the default 64K segment in MS-FORTRAN

A technique to access more than 64K of data space in MS-FORTRAN is shown below. Named COMMON blocks, each of which may be up to 64K bytes in size can be used.

In the example, three COMMON blocks are used, each with 10,000 elements of four bytes each, thus allowing access to 120,000 bytes of data storage.

```
$STORAGE:2
      PROGRAM BIGMEM
      COMMON /COM1/ARRAY1(10000)
      COMMON /COM2/ARRAY2(10000)
      COMMON /COM3/ARRAY3(10000)
      DO 10 I=1,10000
        ARRAY1(I)=I
        ARRAY2(I)=I+10000
        ARRAY3(I)=I+20000
10    CONTINUE
      OPEN(1,FILE='LST')
      DO 20 I=1,10000,1000
        WRITE(1,100)'ARRAY1(I)=',ARRAY1(I),'ARRAY2(I)=',ARRAY2(I),
     1'ARRAY3(I)=',ARRAY3(I)
20    CONTINUE
100   FORMAT(1X,A11,F7.0,A11,F7.0,A11,F7.0)
      END
```

### 5.5 Fix for ASYNC so it will load default file ASYN.IEM when running under MS-DOS.

**Using DDT:**

```
-rasync.mnu
 START         END
02C8:0000   02C8:51FF
-d5100
02C8:5100 3B 53 0B 49 1E 49 45 54 52 53 44 3F 1B 6E 49 6E ;S.I.IETRSD?.nIn
02C8:5110 49 6E 49 6E 49 6E 49 AC 49 A9 49 7F 03 05 08 0A InInInI.I.I.....
02C8:5120 0D 12 15 18 1B 17 55 24 55 CA 54 17 55 CA 54 CA ......U$U.T.U.T.
02C8:5130 54 D2 54 01 55 01 55 F8 54 00 49 45 2F 4D 4F 44 T.T.U.U.T.IE/MOD *
```

```
02C8:5140 45 4D 49 45 4D 00 00 00 00 13 14 15 16 00 02 1F EMIEM...........
02C8:5150 01 7E 4E FA 8E 03 1F 05 9A 00 00 00 00 32 30 54 .~N..........20T
02C8:5160 31 32 33 34 35 36 37 33 30 30 00 31 32 33 34 35 1234567300.12345
02C8:5170 36 37 38 31 32 33 01 24 0D 23 01 D3 4E BF 91 0E 678123.$.#..N...
02C8:5180 23 01 D3 4E 0A 92 04 49 01 D3 4E 93 8F 05 49 01 #..N...I..N...I.
02C8:5190 D3 4E DE 8F 06 49 01 D3 4E 29 90 07 49 01 D3 4E .N...I..N)..I..N
02C8:51A0 74 90 08 49 01 D3 4E BF 90 0C 49 01 D3 4E 9A 91 t..I..N...I..N..
02C8:51B0 0D 49 01 D3 4E E5 91 0E 49 01 D3 4E 30 92 41 44 .I..N...I..N0.AD
-s513a
02C8:513A 49 41
02C8:513B 45 53
02C8:513C 2F 59
02C8:513D 4D 4e
02C8:513E 4F 43
02C8:513F 44 20
02C8:5140 45 20
02C8:5141 4D 20
02C8:5142 49 ^Z

E>d5100
02C8:5100 3B 53 0B 49 1E 49 45 54 52 53 44 3F 1B 6E 49 6E ;S.I.IETRSD?.nIn
02C8:5110 49 6E 49 6E 49 6E 49 AC 49 A9 49 7F 03 05 08 0A InInInI.I.I.....
02C8:5120 0D 12 15 18 1B 17 55 24 55 CA 54 17 55 CA 54 CA ......U$U.T.U.T.
02C8:5130 54 D2 54 91 55 91 55 F8 54 00 41 53 59 4E 43 20 T.T.U.U.T.ASYNC  **
02C8:5140 20 20 49 45 4D 00 00 00 00 13 14 15 16 00 02 1F   IEM...........
02C8:5150 01 7E 4E FA 8E 03 1F 05 9A 00 00 00 00 32 30 54 .~N..........20T
02C8:5160 31 32 33 34 35 56 57 33 30 30 00 31 32 33 34 35 1234567300.12345
02C8:5170 36 37 38 31 32 33 01 24 0D 23 01 D3 4E BF 91 0E 678123.$.#..N...
02C8:5180 23 01 D3 4E 0A 92 04 49 01 D3 4F 93 8F 05 49 01 #..N...I..N...I.
02C8:5190 D3 4E DE 8F 06 49 01 D3 4E 29 90 07 49 01 D3 4E .N...I..N)..I..N
02C8:51A0 74 90 08 49 01 D3 4E BF 90 0C 49 01 D3 4E 9A 91 t..I..N...I..N..
02C8:51B0 0D 49 01 D3 4E E5 91 0E 49 01 D3 4E 30 92 41 44 .I..N...I..N0.AD
?
-wasync.mnu
-^C
```

* No "/" allowed under MS-DOS
** Fixed

## 5.6 Creating ASCII Text Files

Under MS-DOS it is possible to create an ASCII text file (such as a batch file) directly from the keyboard.

From the A> prompt type:

COPY CON A:FILENAME.EXT <CR>

This will allow you to type onto the screen, the text you wish to create. After each line hit the RETURN key.

You can edit the current line you are typing by back spacing over any errors. When finished do a control-Z [^Z] then press RETURN. This will then close the disk file.

For CP/M-86 you will need PIP:

```
PIP A:FILENAME.EXT=CON:"<CR>
```

PIP will then allow you to create a file as above but with the exception that after each line is entered and the RETURN key hit you must also do a control-J [^J] to force a line feed. When finished do a control-Z [^Z] to close the file.

## 5.7 CODEC PROGRAMMING

This describes what needs to be done in order to generate sound using the codec audio section of the Sirius 1.

Refer to technical reference manual for technical explanation of the CODEC.

Software has control over the following functions:

1. Volume clock rate
2. Volume level
3. Codec clock rate
4. Codec mode (Play/Record)
5. SDA Initialisation
6. SDA data transfers

### 5.7.1 Volume

Volume is controlled by the duty cycle of the ultra-audio chopper. The chop rate is generated by a 6522 VIA. This clock can be set once and left. The clock rate should be set at a rate above 20khz in order not to be heard as a tone in the audio output. The system normally sets this clock to the maximum rate the 6522 will generate.

Volume clock rate setup example:

```
INIT_VOLUME_CLOCK: proc;
port$ptr= via_20;
/* set shift register mode in acr to shift out on t2 */
via(acr)= (via(acr) and not(1ch)) or 10h;
```

```
/* set t2 to fastest clock rate */
via(t2)= 1;
/* init volume level to off value, 0= max,ff= off */
/* 00,01,03,07,0f,3f,7f,ff are valid volume values */
via(sr)= 0ffh;
end INIT_VOLUME-CLOCK;
```

Volume level set example:

```
SET_VOLUME: proc(level);
   dcl level     byte;
port$ptr= via_20;
/* move new level into sr to control duty cycle */
via(sr)= level;
end SET_VOLUME
```

### 5.7.2 CODEC Clock

The codec clock rate determines the quality of the recorded message. The higher the clock rate the higher the quality. This clock should be adjusted to fit the need of the application as far as quality and data space require.

Typical clock rate is 16khz for good speech, and 32khz for very good speech quality. The data rate at 16khz is 2k bytes/second and at 32khz it is 4k bytes/second.

The codec clock is generated by a 6522 via using timer 1. The value stored in the 6522 timer is one half the period of the codec clock.

N= ( 500,000/(F*8) )-1 or N= (62500/F)-1
N is the 6522 timer value
F is the desired clock frequency in Hz

Codec clock setup example:

```
/* freq is the desired clock rate in hz */
SET_CODEC_CLOCK: proc(freq);
   dcl freq     word;
port$ptr= via_80;
/* set new timer value for freq */
via(tl)= (62500/freq)-1;
end SET_CODEC_CLOCK;
```

### 5.7.3 Codec Mode Control

The codec can both encode speech input to digital data, and decode stored digital data back to speech output. The mode control for the codec is supplied by using the SM/DTR output of the SDA chip.

Example of codec mode control:

```
/* set mode of code to play or record */
/* 0 is play  1 is record */
SET_CODEC_MODE: proc(mode);
   dcl mode    byte;
if mode=0 then
     do;
     sda.r0=0;      /* select control reg 2 */
     sda.rl=5bh;    /* this sets SM/DTR low */
     end;

else
     do;
     sda.r0=0;      /* select control reg 2 */
     sda.rl=58h;    /* this sets SM/DTR high */
     end;
end SET_CODEC_MODE;
```

### 5.7.4 SDA Initialisation

The SDA performs the parallel to serial conversion and data buffering function for the CPU. This helps cut down the amount of time needed to service the CODEC data feeding and reading. The SDA must be initialised before any other codec operation is performed.

The functions that are setup are:

1. set word length to 8 bits no parity, play mode
2. set sync code to 0aah, on underflow send sync
3. set byte transfer ready to 2 bytes

SDA initialisation example:

```
SDA_INIT: proc;
/* set word length to 8,play mode,2 byte ready */
sda.r0=0;           /* select control reg 2 */
sda.rl=5bh;
/* set sync code to 0aah for quite pattern on underflow */
sda.r0=80h;         /* select sync code */
```

```
sda.r1=0aah;
/* set external sync mode for par to ser operation */
sda.r0=40h;         /* select control reg 3 */
sda.r1=0dh;         /* clear TUF and CTS */
sda.r0=40h;
sda.r1=01h;         /* enable TUF and CTS*/
end SDA_INIT;
```

### 5.7.5 SDA Data Transfer

The SDA is the interface the CPU uses to generate sound from the codec. This example is for educational purposes and is NOT the only method that can be used to drive the codec.

In order to record speech from the codec the CPU must:

1. initialise the SDA
2. set clock rate
3. set volume to off
4. set codec to record mode
5. read data from SDA receive register
6. set codec back to play mode

Example of recording data from codec:

```
/* record a buffer of codec data at buf$ptr */
/* record count bytes of data */
RECORD: proc(buf$ptr,count);
     dcl buf$ptr   pointer;
     dcl count     word;
     dcl buffer(1) based buf$ptr;
     dcl i         word;
call SET_VOLUME(0ffh);    /* set volume off */
call SET_MODE(1);         /* set to record mode */
i=0;
do while (count [ | 0);   /* read in count bytes of data */
     /* wait for receive byte ready in r0 */
     do while (sda.r0 and 1) [|1; end;
     buffer(i)=dsa.r1;    /* read and store data byte */
     i=i+1;
  end;
  call SET_MODE(0);
end RECORD;
```

Play back of codec data example:

```
/* play a buffer of codec data back */
/* buffer pointed at by buf$ptr */
```

```
/* play count bytes */
PLAY; proc(buf$ptr,count);
     dcl buf$ptr   pointer;
     dcl count     word;
     dcl i         word;
     call SET_MODE(0);    /* set play mode */
     do i=0 to count;
       /* wait for ready to send flag */
       do while (sda.r0 and 2)=0; end;
       sda.rl= buffer(i); /* store next byte */
     end;
end PLAY;
```

## 5.8 Data Security

This sample program appends characters to a file. It closes the file and then re-opens it after writing every character to ensure that at most one character is lost if the system crashes.

```
name      test

code      segment public 'code'
assume    cs:code, ds:code

; NOTE: This program assumes that an ASCII file named TEST is
; located on the default drive.  TEST is the file that the data
; is appended to.

lf        equ  0ah
cr        equ  0dh
altz      equ  lah

dir_con_io equ 06h
print     equ  09h
open      equ  0fh
close     equ  10h
write     equ  15h
setdma    equ  lah

test :
          push es        ;save ptr to base page

          mov  ax,code   ;set up DS
          mov  ds,ax

          lea  dx, buffer     ;set up dma
          mov  ah, setdma
          int  21h
```

```
open_it:
          lea  dx, fcb
          mov  ah, open  ;open file
          int  21h
          inc  al
          jz   file_not_found

          mov  fcb_recsize, 1      ;set record size to 1 byte

;  set  the current block and current record to point at the last
;  byte of the file since we want to append it.

          mov  ax, fcb_filesize
          xor  dx, dx
          mov  cx, 128
          div  cx
          mov  fcb_cur_block, ax
          mov  fcb_cur_rec, dl
```

-- listing continued on next page --

```
; get a character from the keyboard

get_char_from_kb:
          mov  ah, dir_con_io
          mov  dl, 0ffh
          int  21h
          jz   get_char_from_kb

;  if the character is ALT-Z, then stop

          cmp  al, altz
          jz   end_prog

; write the character to the file

          mov  buffer, al
          mov  ah, write
          lea  dx, fcb
          int  21h

;  if the character was a carriage return, then write a line feed
;  also
          cmp  byte ptr buffer, cr
          jnz  close_it

          mov  byte ptr buffer, lf
          mov  ah, write
          lea  dx, fcb
          int  21h
```

```
;   close the file to save what we just wrote in case the  system
;   crashes

close_it:
          mov  ah, close
          lea  dx, fcb
          int  21h

;  go to open the file and get the next character

          jmp  open_it

file_not_found:
          lea  dx, not_found_msg
          mov  ah, print
          int  21h

end_prog:

return    proc far
;  do  a  far return to the first byte  of  the  basepage,  which
;  contains an int 20 operation to terminate the program

          xor  ax, ax
          push ax
          ret

return    endp

fcb            db        0
fcb_name       db        'TEST      '
fcb_cur_block  dw        ?
fcb_recsize    dw        ?
fcb_filesize   dw        ?
               dw        ?
fcb_date       dw        ?
fcb_time       dw        ?
fcb_reserved   db        8 dup (?)
fcb_cur_rec    db        ?
fcb_rel_rec    db        4 dup (?)

buffer         db        0

not_found_msg   db       13,10,'The  sample  file  TEST  was  not
                         found.',13,10,'$'

code      ends
          end
```

## 5.9 MS-Pascal Date and Time Program (input, output)

This program demonstrates the use of the MS-Pascal Date and Time procedures, as well as showing how to change the date by using the MS-DOS 1.25 Set Date function with the DOSXQQ function call.

```
PROGRAM Date_Time (INPUT,OUTPUT);

PROCEDURE DATE(VAR s: STRING   ); EXTERNAL;

PROCEDURE TIME(VAR s: STRING   );EXTERNAL;

FUNCTION DOSXQQ( command,parameter: WORD): BYTE;EXTERNAL;

VAR date_str,time_str: LSTRING(8);

FUNCTION Set_Date(CONST date_str: STRING):BOOLEAN;
(* Changes the system date if date_str is valid,  otherwise
 returns FALSE. *)

CONST setdate = QN2B;

VAR month_word,day_word,year_word,param: WORD;
    date_lstr: LSTRING(2);
    date_status,month_byte,day_byte: BYTE;

BEGIN
    Set_date: = FALSE;
    date_lstr.LEN: = 2;
    FOR VAR l:= 1 TO 2 DO date_lstr[l]: = date_str[l];
    IF NOT DECCDE(date_lstr,month_word) THEN RETURN;
    FOR VAR l: = 4 TO 5 DO date_lstr[l-3]: = date_str[l];
    IF NOT DECODE(date_lstr,day_word) THEN RETURN;
    month_byte:= LOBYTE(month_word);
    day_byte:= LOBYTE(day_word);
    param:= BYWORD(month_byte,day_byte);
    FOR VAR l:= 7 TO 8 DO date_lstr[l-6]:= date_str[l];
    IF NOT DECODE(date_lstr,year_word) THEN RETURN;
    year_word:= year_word + 1900;
    CRCXQQ:= year_word;
    date_status:= DOSXQQ(setdate,param);
    IF date_status <> QNFF THEN
     Set_Date:= TRUE;
    END;

BEGIN
    time_str.LEN:= 8;
```

```
    TIME(time_str);
    WRITELN('The current system time is ',time_str);
    date_str.LEN:= 8;
    DATE(date_str);
    WRITELN('The current system date is ',date_str);
    WRITE('Enter  a  new date (mm-dd-yy) to change the  date,  or
          else <cr>:');
    READLN(date_str);
    WRITELN;
    IF date_str.LEN > 0 THEN
      BEGIN
        IF Set_Date(date_str) THEN
         BEGIN
          DATE(date_str);
          WRITELN('The new system date is ',date_str);
         END
        ELSE
         WRITELN('Date ',date_str,'is not a valid date');
       END;
      END.
```

### 5.10 Accessing System Time in dBASE II

ENTRY:BX - Pointer to length byte at start of string
- 11 byte (space) string passed from dBASE

EXIT: string is passed back to dBASE with time

CHANGE: All registers destroyed, but dBASE will return machine's state when it regains control.

This program takes an 11 byte long string from dBASE and puts the MS-DOS time into it. The string must be of character type. If the string is not 11 bytes long, the routine is exited with no change. By typing DTIME at the system prompt before entering dBASE, the first part of the program loads the second half at address 65024 decimal in the current program segment. (Actually, any location above A400H is okay. Further, the dBASE command load [filename] could load an assembly language routine within dBase). This address is then used inside dBASE as the argument for the SET CALL TO command. Because the 1/100th seconds clock in MS-DOS returns either 00 or 50 it was felt to be of limited use and was not included in this program. A 12 hour clock with an AM or PM tag at the end was employed for ease of use. If a 24 hour clock is desired the conversion code can easily be eliminated. A SORT routine might overwrite the time code if it is large enough. If this is a problem, the code can be modified to use INT 27H to create a protected area in MS-DOS. A common sequence of commands to get the time would be:

```
A>DTIME

      A>DBASE

      Copyright (C) 1982 RSP Inc.

      *** dBASE II/86     Ver 2.4        1 July 1983

      .STORE"12345678901" TO TIME
      12345678901
      .SET CALL TO 65024
      .CALL TIME
      ? TIME
      5:23:00 PM

STOUT          GROUP          CODE
ASSUME         CS:STOUT,DS:STOUT              ;"STring OUT"

This first part loads GETTIME
CODE           SEGMENTPUBLIC 'CODE'
               ORG  100H          ;load over PSP
               MOV  CX,0A6H       ;prepare to move A6H bytes
               LEA  SI,GETTIME    ;put start of time code in SI
               MOV  DI,0FE00H     ;set destination above dBASE
               CLD
               REP  MOVSB         ;move it
               INT  20H           ;terminate

This is the actual time code

GETTIME        PROC NEAR

               NOP  ; entry target
               MOV  DI,BX       ;put address of length byte in DI
               XOR  BX,BX         ;clear it
               MOV  BL,0BH        ;put expected length in BL
               CMP  [DI],BL       ;is it 11 bytes long?
               JNZ  SHORT DONE    ;no, exit with no change

               INC  DI            ;move DI to start of string
               MOV  AH,2CH        ;get time in CX and DX request
               INT  21H           ;get it

               MOV  AL,CH         ;put hours in AL
               CMP  AL,0CH        ;after 12:00 noon?
               JGE  AMPM_FLAG     ;keep under 12, flag as PM
               CMP  AL,00         ;is it zero o'clock?
               JZ   SHORT MIDNIGHT  ;make it midnight
```

```
CONHOURS:       CALL CONVERT        ;convert to ASCII
                MOV  BH,0FFH        ;flag to OUT that this is hours
                CALL OUT            ;put hours in string

                MOV  AL,C           ;put minutes in AL
                CALL CONVERT
                CALL OUT

                MOV  AL,DH          ;put seconds in AL
                CALL CONVERT
                MOV  BH,0AAH        ;flag for OUT
                CALL OUT

                CMP  BL,0FFH        ;is it night?
                JZ   SHORT POUT
                MOV  [DI],BYTE PTR 'A'  ;give me an A
MOUT:           INC  DI
                MOV  [DI],BYTE PTR 'M'  ;give me an M
                JMP  SHORT DONE
POUT:           MOV  [DI],BYTE PTR 'P'  ;give me a P
                JMP  SHORT MOUT

DONE:           RET  ;all finished


CONVERT:        XOR  BH,BH          ;clear counter
                PUSH BX             ;save AM/PM flag

CONVERT2:       AAM                 ;unpack AL
                ADD  AL,30H         ;bump to ASCII
                ADD  BH,01          ;loop count
                CMP  AH,0           ;quotient zero?
                JZ   SHORT CVRTDONE  ;then we're through
                MOV  BL,AL          ;save LSD
                MOV  AL,AH          ;for next unpack
                JMP  SHORT CONVERT2  ;do it again

ONEDIGIT:       MOV  AH,30H         ;MSD is zero
                POP  BX
                RET

CVRTDONE:       CMP  BH,01          ;one digit number?
                JZ   SHORT ONEDIGIT  ;put '0' in AH
                MOV  AH,AL          ;put MSD in AL for OUT
                MOV  AL,BL          ;put LSD in AL for OUT
                POP  BX
                RET
```

```
AMPM_FLAG:      MOV  BL,0FFH        ;flag as PM
                CMP  AL,0CH         ;is it after 12 noon?
                JG   SHORT CHOP     ;then keep hours under 12
                JMP  SHORT CONHOURS

CHOP:           SUB  AL,0CH
                JMP  SHORT CONHOURS

MIDNIGHT:       MOV  AL,0CH
                JMP  SHORT CONHOURS

OUT:            CMP  BH,0FFH        ;is it hours?
                JZ   SHORT ZEROKILL  ;might kill first zero
OUT2:           MOV  [DI],AH        ;move out first digit
                INC  DI             ;point to next string position
                MOV  [DI],AL        ;move out second digit
                INC  DI             ;point to next string position
                CMP  BH,0AAH        ;is it end of time numbers?
                JZ   SHORT BLANKOUT   ;send a space
                MOV  [DI],BYTE PTR ':'  ;send out colon
                INC  DI             ;advance to next position
OUT3:           RET

ZEROKILL:       XOR  BH,BH          ;clear flag that got us here
                CMP  AH,'0'
                JZ   SHORT ZEROFF
                JMP  SHORT OUT2


ZEROFF:         MOV  AH,' '
                JMP  SHORT OUT2


BLANKOUT:       XOR  BH,BH          ;clear flag that got us here
                MOV  [DI],BYTE PTR ' '  ;send space
                INC  DI
                JMP  SHORT OUT3

GETTIME         ENDP

CODE            ENDS

                END
```

## 5.11 Programming the 8253 Timer

The 8253 Timer, counter 2 is used for timer interrupt. Clock rate from Sirius hardware is 100 kHz.

For example, to obtain 100 microsecond interrupts we need to divide by 10. The 8253 can be used in a BCD or binary mode.

In assembler:

```
     org 100h

start:
     mov  bx,0e000h       ; ES points to I/O
     mov  es,bx
     mov  bx,23h          ; address mode register
     mov  al,0b5h         ; counter 2, mode 2, BCD
     mov  es:[bx],al      ; write mode word
     mov  bx,22h          ; address counter 2
     mov  al,10h          ; least significant byte (LSB)
     mov  es:[bx],al
     mov  al,0            ; most significant byte (MSB)
     mov  es:[bx],al
```

The result is a 10 microsecond pulse every 100 microseconds.

Maximum time = 620 milliseconds (ms) approx. (MSB=0, LSB=0)
Minimum time = 20 microseconds (us) approx. (MSB=0, LSB=2)
(Binary mode)

$10^4$ = 100 ms (0,0)

$10^3$ = 10 ms (10h,0)

$10^2$ = 1 ms (1,0)

$10^1$ = 100 us (0,10h)

## 5.12 Manipulating a Batch File

```
100 '-----------------------------------------------------------------
110 '---   START.BAS     by Keith Pickup                              ---
120 '---                 Barson Computers (Sydney - Australia)        ---
130 '---                                                              ---
140 '---   This program manipulates a batch file so that programs    ---
150 '---   may be executed by selecting them from a master menu.      ---
160 '---   Parameters may be passed to the command string by tagging  ---
```

```
170 '---   them to the program name string as in lines 710 & 720        ---
180 '---   The program could also be compiled which would help          ---
190 '---   speed it up.                                                 ---
200 '---                                                                ---
210 '---   This program requires an AUTOEXEC.BAT file to be created     ---
220 '---   which contains the command 'MENU'.                           ---
230 '---   MENU is in fact MENU.BAT which contains the following        ---
240 '---                                                                ---
250 '---           MSBASIC START (or just START if compiled)            ---
260 '---           OPTION (which is a dummy command)                    ---
270 '---           PAUSE **** Program Terminated ****                   ---
280 '---           MENU (which calls the original .BAT file             ---
290 '---                                                                ---
300 '---   The PAUSE allows programs like CHKDSK to display their       ---
310 '---   information and wait for a response from the keyboard        ---
320 '---   before re-loading the master menu program                    ---
330 '---                                                                ---
340 '---   Lastly there is a file called FINISH.BAT which contains      ---
350 '---   no commands at all therefore allowing exit to the system     ---
360 '---                                                                ---
370 '---   NOTE: The command 'dir/w|sort|more' relates to DOS 2.0       ---
380 '---                                                                ---
390 '--------------------------------------------------------------------
400 '
410 ' *** define program variables ***
420 '
430 WIDTH 255:HEADING$="  **** MASTER MENU ****  "
440 ESC$=CHR$(27):REV$=ESC$+"p":ROFF$=ESC$+"q":CLR$=ESC$+"z":HOME$=ESC$+"H"
450 DIM SELECT$(10),PROG$(7),TEMP$(4),FK$(7):TOTAL.OPTIONS=7
460 FOR K%=1 TO TOTAL.OPTIONS:READ SELECT$(K%):NEXT K%
470 DATA "Sorted Directory Listing","Check disk space","Format new disk"
480 DATA "Copy diskettes","Edit ASCII file","Microsoft Basic"
490 DATA "Return to Operating System"
500 PAUSE$="pause **** Program Terminated ****"
510 PROG$(1)="dir/w|sort|more":PROG$(2)="chkdsk":PROG$(3)="format/e"
520 PROG$(4)="dcopy/e":PROG$(5)="edlin":PROG$(6)="msbasic":PROG$(7)="finish"
530 '
540 ' *** read current contents of the MENU.BAT file ***
550 '
560 OPEN "I",#1,"MENU.BAT"
570 FOR K%=1 TO 4
580 INPUT #1,TEMP$(K%)
590 NEXT K%:CLOSE 1
600 '
610 ' *** display menu for selection of option ***
620 '
630 PRINT CLR$;:KEYBASE=1:GOSUB 820
640 PRINT HOME$;TAB(40-(LEN(HEADING$)/2)+.5) REV$;HEADING$;ROFF$:PRINT:PRINT
```

```
650 FOR K%=1 TO TOTAL.OPTIONS
660 PRINT TAB(25);REV$;K%;ROFF$;"   ";SELECT$(K%):PRINT
670 NEXT K%
680 PRINT:PRINT TAB(25) "Please select option required ";
690 IP$="":IP$=INKEY$:IF IP$="" THEN 690
700 IP=VAL(IP$):IF IP < 1 OR IP > TOTAL.OPTIONS THEN 690
710 IF IP=5 THEN PRINT:PRINT TAB(25);:INPUT "Edit file name ";FILE.NAME$
720 IF IP=5 THEN PROG$(5)=PROG$(5)+" "+FILE.NAME$
730 TEMP$(2)=PROG$(IP):TEMP$(3)=PAUSE$
735 FOR I%=1 TO 7:PRINT ESC$+"41"+CHR$(I%)+CHR$(&HF0+I%):NEXT I%
740 '
750 ' *** write out new batch file replacing 'option' with program name ***
760 '
770 OPEN "O",#1,"MENU.BAT"
780 FOR K%=1 TO 4
790 PRINT #1,TEMP$(K%)
800 NEXT K%:CLOSE 1
810 PRINT CLR$:SYSTEM
820 '
830 ' *** set up for the 25th line ***
840 '
850 IF KEYBASE=1 THEN RESTORE 1010
860 IF KEYBASE=2 THEN RESTORE 1020
870 IF KEYBASE=3 THEN RESTORE 1030
880 FOR I%=1 TO 7:PRINT ESC$+"41"+CHR$(I%)+CHR$(&H30+I%):NEXT I%
890 READ FKCT
900 FOR I%=1 TO FKCT:READ FK$(I%):NEXT I%
910 GOSUB 920:RETURN
920 FKSZ=INT((80-(FKCT-1))/FKCT)
930 X9$="":C9$=ESC$+"q"+" "+ESC$+"p"
940 FOR I%=1 TO FKCT
950 B9$=LEFT$(FK$(I%),FKSZ):J9%=INT((FKSZ-LEN(B9$)+1)/2)
960 X9$=X9$+C9$+LEFT$(SPACE$(J9%)+B9$+SPACE$(FKSZ),FKSZ)
970 NEXT I%
980 X9$=ESC$+"p"+X9$+ESC$+"q"
990 PRINT ESC$"x1";ESC$"j";ESC$"Y8 ";ESC$"1";X9$;ESC$"Y  ";ESC$"k";
1000 RETURN
1010 DATA 7,"SORT DIR","CHK DISK","FORMAT","DCOPY","EDIT","MSBASIC","EXIT"
1020 DATA 7," "," "," "," "," "," "," "
1030 DATA 7," "," "," "," "," "," "," "
```

### 5.13 CALC (or UDCCALC) - Calculator Function

CALC, the calculator function (version 1.1 or later) on both MS-DOS and CP/M-86 work as follows:

Call up the calculator function by typing:

CALC <CR>

and striking SHIFT with CALC key.

### Division

- no quirks

eg. 9÷6 (CALC KEY)

### Multiplication

- one quirk - Your X key on the keyboard may not be configured so use the * key instead. (ie. SHIFT and 8 key).

eg. 9 X 6 (CALC KEY)

or 9 * 6 (CALC KEY)

### Addition and Subtraction

- many quirks - Do not use the (CALC KEY) for the equal sign as it does not work. Only use it to clear the accumulator.

eg. 9 + 6 = would be 9 + 6 +

9 - 6 = would be 9 - 6 -

## 5.14 Directory Entries

The maximum number of directory entries in each version of the operating system is as follows:

| | |
|---|---|
| CP/M 1.0 | 128 |
| CP/M 1.1 | 128 |
| MS-DOS 1.25/2.5 or earlier (floppy discs) | 128 |
| MS-DOS 1.25/2.5 (hard discs) | no practical upper limit |

Notes

1. The maximum number of directory entries possible is not necessarily the same as maximum number of files.

   a. CP/M-86 records are 128-byte blocks of data. When a file is created an extent (area) of disc is allocated to contain a maximum of 128 records i.e. 16K bytes of data. If files are larger than 16K bytes they will need extra extents.

      The directory of a CP/M-86 disc contains an entry for each extent of each file. Therefore a file with more than one extent will have a multiple directory entry for each file displayed.

   b. MS-DOS does not block records into 128 bytes each, nor does it use 16K file extents. A file can be scattered over the disc, not necessarily residing in contiguous sectors. Therefore the directory contains an entry for each block of file on disc.

      It is advisable to run the Check disc utility (CHKDSK.COM) to reclaim disc space after files have been deleted.

5.15 MS-DOS File Sizes and Disc Structure

Floppy disc

The CP/M-86 operating system blocks individual records into multiples of 128 bytes. eg. An 80 character record will always occupy 128 bytes of disc space.

MS-DOS on the otherhand, does not do this. It does however, group records together into blocks or allocation units of 2K bytes (2048 bytes). These are the smallest units that can be written to or read from disc at any one time. Therefore a file will always occupy multiples of 2K bytes.

example

A file with fixed length records of 80 characters, containing 150 records.

Data file = 80 * 150 bytes
= 12000 bytes

The disc space will then be allocated as follows:

Number of 2K byte blocks = size of data file divided by 2048
= 12000 divided by 2048
= 5.859
= 6 (rounding upwards to the nearest whole block)

Total disc space = 6 * 2K blocks
= 6 * 2048 bytes
= 12228 bytes

The DIR command will give the file size as 12000 bytes however, the actual amount of disc space allocated for the file is 12228 bytes.

The CHKDSK command will give the sum of the actual disc space allocated for the files.

| FILES | DIRECTORY SIZE | DISC SPACE |
|---|---|---|
| COMMAND.COM | 5737 | 6144 |
| SETIO.COM | 1012 | 2048 |
| CHKDSK.COM | 1976 | 2048 |
| DCOPY.COM | 15776 | 16384 |
| FORMAT.COM | 17132 | 18432 |
| RDCPM.COM | 11214 | 12288 |
| MSBASIC.COM | 31360 | 32768 |
| UDCCALC.COM | 4917 | 6144 |
| EDLIN.COM | 2432 | 4096 |
| DISKID | 1536 | 2048 |
| | 93092 | 102400 |

CHKDSK gives the value of 102400 bytes.

Hard_Disc

The only difference with the Internal Winchester Sirius and the floppy disc drive Sirius is the size of the allocation units. On the floppy disc drive the allocation unit size is fixed at 2K. On the hard disc the user can specify different allocation units for each volume. therefore files on disc can be multiples of 2K, 4K, 8K etc.

# WORD PROCESSING NOTES

## 6.1 INSTALL - Brief notes on the use of Install supplied with Wordstar Ver 3.21 (MS DOS VERSION)

** BACK UP YOUR MASTER DISC BEFORE INSTALLING WORDSTAR

The install program allows you to change Wordstar to suit different installations.

When you invoke Install (by typing INSTALL <CR>) it logs on and provides you with most of the information you require. It then asks a number of questions.

Please read the questions carefully before answering. Below are some of the questions (in brief) with the usual answer.

Q. Which MicroPro product do you wish to install?

A. WS <CR>

Q. Name of file to install, or <RETURN> for WSU.COM

A. <RETURN> or WS.COM

(The file WSU.COM is an uninstalled version of Wordstar WS.COM has been installed for the Sirius and Diablo printer. Normally you would simply want to make minor changes to an already installed version. If you want to start from scratch then install WSU.COM

Q. Name of file for installed WORDSTAR, or <RETURN> for WS.COM
A. <RETURN> or filename.COM

(You can write your installed WORDSTAR back to WS.COM if you like but make sure you have a backup of the old version just in case.)

A Menu of Terminals
B Custom Installation of Terminals
C Menu of Printers
E Menu of WORDSTAR features
F Custom Modification of WORDSTAR
X Exit

In more detail -

A Menu of Terminals
B Custom Installation of Terminals

Normally you would make no changes here since WS is already installed for the Sirius terminal, otherwise select option E from Menu 2.

If you want to install Wordstar for use with 132 column mode or you don't like the method of highlighting, then try option B.

If you want to change any of the screen attributes you must refer to the screen driver escape codes found in the Dealer Users Guide. (Which can be obtained from Barsons).

C Menu of Printers

Select your printer from the list given. If your printer cannot be found (eg MT180) then select "I - Teletype-like printer".

Communication protocol.

Answer A. (handled outside Wordstar by the interface if your cable is correct).

List device is normally "Primary List Device".

E Menu of WORDSTAR features

This section allows you to choose such things as initial help level, decimal tab character, initial justification etc.

F Custom Modification of Wordstar

Allows you to change individual bytes. Use in conjunction with listing of user-definable section.

This section only allows you to change bytes at addresses in the range 100H-949H. (Use DEBUG to change bytes outside this range.)

## 6.2 Summary of the WordStar Patch Locations

(For a more detailed description, see WordStar Patch notes, obtainable from your dealer).

Use the "F" command in Install to make these changes:

Example 1:

The byte at location 2D2H (hexadecimal notation) controls the length of time the WordStar Sign-on banner remains on the screen. Its initial value is 16 (10H). To change this to a smaller value (say zero) enter the Install program and select the "F" option (custom modification of WordStar).

Enter a starting address of 2D2H and Install will display the contents of 2D2H and the next 15 characters (bytes). Check that the first byte has the value 10H and if so, answer the next question with "Y". Enter the new value 0 then enter a full stop "." to exit this mode. Follow the remaining instructions to return to the main menu.

Example 2:

WordStar on the Sirius uses reverse video to highlight menus and other messages. Some people prefer to use high intensity for this. The strings required by Sirius to set and clear reverse video are "Esc p" and "Esc q" respectively (hex: 1B,70 and 1B,71). The strings required to set and clear high intensity are "Esc,(" and "Esc,)" respectively (hex: 1B,28 and 1B,29).

The "turn on highlighting" string starts at location 284H, therefore we must enter three bytes starting at this location. The first byte is a byte count (in this case 2) followed by the required bytes. Thus we enter the three bytes (in hex) 02, 1B, 28 starting at location 284H and the three bytes 02, 1B, 29 starting at location 28BH.

| **STARTING ADDRESS (in hex)** | **NAME** | **NUMBER OF BYTES AVAILABLE FOLLOWING BYTE COUNT (IF ANY)** | **PURPOSE** |
|---|---|---|---|
| 248 | HITE | NIL | Screen height (in lines) |
| 249 | WID | NIL | Screen width (in characters) |
| 284 | IVON | 6 | Turn on highlighting |
| 28B | IVOFF | 6 | Turn off highlighting |
| 2D2 | DEL4 | NIL | Sign-on delay 0-16 |

| | | | |
|---|---|---|---|
| 2D3 | DEL5 | NIL | Screen refresh delay 0-10 |
| 360 | ITHELP | NIL | Initial help level 0-3 |
| 362 | ITITOG | NIL | Set to zero to boot with Insert Off |
| 363 | ITDSDR | NIL | Set to zero for initial no file display |
| 385 | | NIL | Word wrap flag |
| 386 | | NIL | Justify flag |
| 387 | | NIL | Variable tabs flag |
| 388 | | NIL | Soft hyphen flag |
| 389 | | NIL | Hyphen help flag |
| 38A | | NIL | Print control & soft hyphen display flag |
| 38B | | NIL | Display ruler flag |
| 38C | | NIL | Dynamic page break flag |
| 38D | | NIL | Page break display flag |
| 38E | | NIL | Initial line spacing flag |
| 746 | POSMTH | NIL | =1 for daisy wheel, =0 for backspacing printer, =FF for CR then another whole line |
| 747 | BLDSTR | NIL | Number of strikes for "boldface" |
| 748 | DBLSTR | NIL | Number of strikes for "double strike" |
| 74C | PSCRLF | 10 | String to advance printer to next line |
| 757 | PSCR | 6 | String to return carriage to start of same line |
| 75E | PSHALF | 6 | String to do carriage return & half line feed |
| 765 | PBACKS | 5 | String to backspace |
| 76B | PALT | 4 | String to set alternate character width |
| 770 | PSTD | 4 | String to reset to standard character width |

| STARTING ADDRESS (in hex) | NAME | NUMBER OF BYTES AVAILABLE FOLLOWING BYTE COUNT (IF ANY) | PURPOSE |
|---|---|---|---|
| 775 | ROLUP | 4 | String to roll carriage up partial line |
| 77A | ROLDOW | 4 | String to roll carriage down partial line |
| 77F | USR1 | 4 | String for user function 1 |
| 784 | USR2 | 4 | String for user function 2 |
| 789 | USR3 | 4 | String for user function 3 |
| 78E | USR4 | 4 | String for user function 4 |
| 793 | RIBBON | 4 | String to change ribbon to alternate colour |
| 798 | RIBOFF | 4 | String to reset ribbon colour |
| 79D | PSINIT | 16 | String to initialise printer |
| 7AE | PSFINI | 16 | String to reset |

6.3 Summary of the keyboard table AUSWP4.KB

| KEY LABEL | UNSHIFTED | SHIFTED | ALTERNATE |
|---|---|---|---|
| ESC | ESCAPE<br>ESC | ESCAPE<br>ESC | ABANDON FILE<br>^KQ |
| INT ON/OFF | BOLD ON/OFF<br>^PB | LEFT END OF LINE<br>^QS | RIGHT END OF LINE<br>^QD |
| RVS ON/OFF | DISPLAY HELP MENU<br>^J | SET MIN. HELP<br>^JH0 | SET MAX. HELP<br>^JH3 |
| UNDL ON/OFF | UNDERLINE ON/OFF<br>^PS | DIRECTORY ON/OFF<br>^KF | HIDE MARKERS<br>^KH |

| | | | |
|---|---|---|---|
| F1 | SET TEMP. LEFT MARGIN ^OG | SET LEFT MARGIN AT CURSOR ^OL,ESC | SET LEFT MGN AT COL. ENTRY ^OL |
| F2 | MARK BEGINNING OF BLOCK ^KB | MARK END OF BLOCK ^KK | COPY MARKED TEXT ^KC |
| F3 | READ FILE INTO TEXT ^KR | WRITE MARKED TEXT ^KW | MOVE MARKED TEXT ^KV |
| F4 | REFORM PARAGRAPH ^B | CENTRE TEXT ^OC | PAGE BREAK .PA |
| F5 | FIND (ONLY) ^QF | FIND & REPLACE ^QA | GLOBAL CHANGE G,W,<CR> |
| F6 | BACK UP FILE AND RETURN ^KS^QP | END EDIT WITH SAVE ^KD | EXIT WS WITH SAVE ^KX |
| F7 | SET RIGHT MARGIN AT CURSOR ^OR,ESC | SET RIGHT MARGIN AT COL. ENTRY ^OR | TOGGLE JUSTIFY ^OJ |
| BACKSPACE | BACKSPACE & DELETE DEL | DELETE WORD LEFT ^A^T | BACKSPACE ^H |
| CLR | GO TO TOP OF SCREEN ^QE | GO TO BOTTOM OF SCREEN ^QX | |
| DEL CHAR | DELETE CHAR RIGHT ^G | DELETE WORD RIGHT ^T | |
| INS MODE/LINE | INSERT ON/OFF ^V | INSERT CARRIAGE RETURN ^N | |
| DEL EOL/LINE | DELETE EOL ^QY | DELETE LINE ^Y | DELETE BLOCK ^KY |
| SCROL | SCROLL DOWN ^W | SCROLL UP ^Z | SCROLL UP CONTINUOUSLY ^Q^Q^C |

| | | | |
|---|---|---|---|
| LTRL | REPEAT FIND/ REPLACE ^L | RE-ENTER LAST FILE ^R | |
| CURSOR UP | CURSOR UP ONE LINE ^E | CURSOR UP ONE SCREENFULL ^R | |
| CURSOR DOWN | CURSOR DOWN ONE LINE ^X | CURSOR DOWN ONE SCREENFULL ^C | |
| CURSOR LFT | LEFT ONE CHARACTER ^S | LEFT ONE WORD ^A | |
| CURSOR RGHT | RIGHT ONE CHARACTER ^D | RIGHT ONE WORD ^F | |
| TAB | TAB ^I | SET TAB @ CURSOR ^OI,ESC | DEL. TAB AT CURSOR ^ON,ESC |
| Z | (NORMAL) | (NORMAL) | SUBSCRIPT ON/OFF ^PV |
| X | (NORMAL) | (NORMAL) | SUPERSCRIPT ON/OFF ^PT |
| ± | CANCEL FUNCTION ^U | TOP OF FILE ^QR | BOTTOM OF FILE ^QC |
| 00 | (COMMA) | (COMMA) | 00 |

## 6.4 How to turn a CP/M version of WordStar 3.21 into an MS-DOS version using DDT86

```
DDT86
DDT86 1.1
-RWS.CMD
  START      END
03C0:0000 03C0:52FF
-S0324
03C0:0324 E9 90
03C0:0325 39 90
03C0:0326 00 C3
03C0:0327 E9 90
03C0:0328 4A 90
03C0:0329 00 C3
03C0:032A 01 00
03C0:0328 00 .
-S0356
03C0:0356 00 FF
03C0:0357 00 .
-WWS.COM,180,52FF
```

Use RDCPM to copy WS.COM and all overlays to an MS-DOS disc. Depending on the version of DDT86, and the version of the operating system, DDT86 may load to a different segment address. Start changing bytes at location 324 anyway. You may also find that some of your original bytes differ from those shown above, change them regardless.

## 6.5 Using the C.Itoh F10 printer with WordStar

To use the F10 with WordStar the switch pack SW41 (the right-hand set of switches under the front panel) should be set as follows:

O O C C C C C O C O

where O=Open and C=Closed.

Using the Install program supplied with WordStar select the C.Itoh/Starwriter printer option.

If you are using the Easifeed cut-sheet feeder then use the following codes at the top of each file:

```
.PL90
.MB32
```

to set the page length at 90 lines and the bottom margin at 32 lines.

## 6.6 Benchmark

Function keys 6 and 7 are implemented within Benchmark 3.0M Rev C, as described below:

ALT-function key 6 - Prints Content of Screen to Printer

This allows either a menu or any page within a document displayed on the screen to be sent out to a printer. Once ALT-function key 6 (ALT-F6) is pressed, the following message will be displayed on the bottom of the screen:

"Press:CONT to Print theScreen; F1 to Go to Top of Form; CAN to Skip"

To obtain a printout of the screen, press the PAUSE/CONT key.

NOTE: On the Sirius the CAN key is the DEL EOL key.

ALT-function key 7 - System Interrupt/Stop Print

While in "Print" mode, depression of ALT-function key 7 (ALT-F7) will stop the printing of a document. The following message will appear at the bottom of the screen:

"Printer Interrupt, Restart Printer?; Press Y -Yes or N-No"

If "Y" is pressed, the document will continue to print. If "N" is pressed, all printing stops and the program will return to the main screen (create, revise, print, etc.).

## 6.7 XON/XOFF Printer Driver for WordStar

This patch uses the user-installed patch areas. To use this patch install Wordstar for user-installed printer driver and XON/XOFF protocol. Enter the patch starting at location 0811h.

Use the Wordstar Install program to make the patches. With Wordstar version 3.3 you must use DDT-86 (CP/M) or DEBUG (MS-DOS) to make the patches because the install program does not give you full access to the code.

Refer to the Wordstar Installation Guide and patch listings for further information

Before exiting the Wordstar Install program (or DDT-86 or DEBUG), check that the following locations have these values:

| Location | Name | Value |
|---|---|---|
| 7C9h | CSWITCH | 2 |
| 7CAh | HAVBSY | FFh |
| 879h | PROTCL | 2 |

The patch uses direct I/O to the Sirius hardware serial port B. To use port A, change iostat to 2 and iodat to 0.

```
  E004          iobase   equ   0e004h       ; i/o port address
  0003          iostat   equ   3            ; status port (2=A, 3=B)
  0001          iodat    equ   1            ; data port (0=A, 1=B)
  0001          instat   equ   1            ; input status mask
  0004          outstat  equ   4            ; output status mask

                         org   811h
0811 E9 06 00            jmp   pubsy        ; test for printer busy
0814 E9 16 00            jmp   pusend       ; print a character
0817 E9 20 00            jmp   puinp        ; input a character
                pubsy:
081A BB 04 E0            mov   bx,iobase    ; point to i/o port
081D 8E C3               mov   es,bx
081F BB 03 00            mov   bx,iostat    ; point to status port
0822 26 8A 07            mov   al,es:[bx]   ; get status in AL
0825 24 04               and   al,outstat   ; bit 2=0 if busy
0827 75 02               jnz   pubsyl
0829 F9                  stc                ; return CY=1 if busy
082A C3                  ret
                pubsyl:
082B F8                  clc                ; return CY=0 if not busy
082C C3                  ret

                pusend:
082D BB 04 E0            mov   bx,iobase    ; point to i/o port
0830 8E C3               mov   es,bx
0832 BB 01 00            mov   bx,iodat     ; point to data port
0835 26 88 07            mov   es:[bx],al   ; print data from AL
0838 F8                  clc                ; return CY=0 if done
0839 C3                  ret
                puinp:
083A BB 04 E0            mov   bx,iobase    ; point to i/o port
083D 8E C3               mov   es,bx
083F BB 03 00            mov   bx,iostat    ; point to status port
0842 26 8A 07            mov   al,es:[bx]   ; get status byte in AL
```

```
0845 24 01              and     al,instat    ; bit 0=0 if no data
0847 74 08              jz      puinpl
0849 BB 01 00           mov     bx,iodat     ; data ready, so get it
084C 26 8A 07           mov     al,es:[bx]
084F F8                 clc                  ; CY=0 if we have data
0850 C3                 ret
                puinpl:
0851 F9                 stc                  ; CY=1 if no data ready
0852 C3                 ret
```

# CP/M-80 SYSTEM - Z-80 CARD

## 7.1 Z-80 CPU CARD

The Z-80 CPU board was designed to accommodate software written to run under the CP/M-80 operating system. The Z-80 card allows users to run existing Z-80 software while using all the advantages of the Sirius 1 computer system. The Z-80 board contains a Z-80B microprocessor running at 6 MHZ, 64K of dynamic RAM memory, and a Corvus hard disc interface.

The Z-80 board uses the Sirius 1 for all I/O including graphics capabilities, disc storage, and access to I/O ports. The Corvus hard disc interface allows the user to connect a Corvus 5, 10, or 20 megabyte hard disc to the Sirius 1. Up to four 20 megabyte drives may be on line at one time and up to 64 Sirius 1 computers may be networked together using the Corvus constellation multiplexer board. High speed backup onto standard video tape is available from Corvus by ordering the corvus mirror option.

CP/M-80 is a disc operating system that manages program and data files. CP/M-80 programs will run on a Z-80 system provided sufficient memory is available (64K bytes max).

The Z-80 card occupies one of the available I/O port addresses. The Z-80 board comes factory set for I/O port address 0. This address may be changed if necessary. The two 10 position dip switch banks set the I/O port address for the Z-80 card.

(NOTE: When the Z-80 card hardware port address is changed, the user must also change the CP/M-80 software I/O port address for the Z-80 board by using the NEWSYS program).

The table below describes the I/O port address switch banks.

**SWITCH 1** (near centre of card)

| PIN # | I/O ADDRESS BIT # |
|---|---|
| 1 | 0 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |
| 5 | 4 |
| 6 | 5 |
| 7 | 6 |

| | |
|---|---|
| 8 | 7 |
| 9 | 8 |
| 10 | 9 |

**SWITCH 2** (furthest from centre of card)

| PIN # | I/O ADDRESS BIT # |
|---|---|
| 1 | 10 |
| 2 | 11 |
| 3 | 12 |
| 4 | 13 |
| 5 | 14 |
| 6 | 15 |
| 7 | RESERVED |
| 8 | RESERVED |
| 9 | RESERVED |
| 10 | RESERVED |

By typing a command, the Sirius 1 user can switch between CP/M-80 and CP/M-86 operating systems. (Note: CP/M operating system disc must be loaded to access CP/M-80 operating system.)

To load CP/M-80, from CP/M-86 type:
80<CR>

To return to CP/M-86 from CP/M-80 type:
86<CR>

Refer to the CP/M-80 User's Guide for further software information.

Note that the Z-80 card cannot be used with the MS-DOS operating system and therefore cannot be used in a hard-disc based Sirius 1 to access the hard disc.

# HARD DISC

## 8.1 HARD DISC

Victor currently offers an internal or external hard disc drive with their microcomputer. Although it is similar in physical size to the floppy disc drive nearly nine times the amount of data can be stored (approximately 10.6 megabytes formatted) and the speed of data access is much greater than the speed of the floppy disc drive. Programs will run significantly faster and larger data files can be maintained.

Also known as the Winchester disc drive, a name coined by IBM in 1973 to describe a dual 30 megabyte disc configuration (30/30), Tandon Corporation became the initial supplier. Two different models may be encountered when servicing the hard disc, the TM502 or the TM603SE, both of which are compact units that use a moving head, noncontact recording method with standard Winchester technology on a 130mm rigid medium. The storage medium is contained within the drive in a fixed, non-operator removable, configuration.

The hard disc subsystem consists of five major hardware components; the Winchester disc drive, Spindle and Motor Control Board, TM600 Main uP Board, Xebec Controller (pronounced zee-beck), and the DMA Interface Board.

## 8.2 Disc Drive Functional Characteristics

The following information pertains to the TM502 and TM603SE:

### 8.2.1 Disc Rotation

The medium is rotated at 3,600 rpm ±1 percent by a direct drive brushless D.C. motor, giving an average latency of 8.3 milliseconds. Multiple track access time is reduced (TM502 drives only) by the use of an on-board 8748 microprocessor which calculates the optimum positioning algorithm.

### 8.2.2 Head Positioning

Head positioning is by a split band, open loop, rotary positioning system. The track-to-track step time is three milliseconds plus fifteen milliseconds for head settling time after the last step of a seek. Heads automatically reposition to Track 000 at power up.

### 8.2.3 Start/Stop

The drive reaches its operating speed 15 seconds after power is applied to the drive circuitry. Internal hard disc units reach operating speed 15 seconds after the power switch on the microcomputer mainframe is switched on. External hard disc units reach operating speed 15 seconds after the power switch on the external hard disc unit is turned on. In addition, the disc stops rotating within 15 seconds after power is removed from the motor drive circuitry. A solenoid-operated, mechanical brake is provided for rapid spindle deceleration, and to preclude the possibility of head or disc damage during shipping.

### 8.2.4 Air Filtration

A self-contained, recirculating air filtration system supplies clean air through a 0.3 micron filter. A secondary absolute filter is provided to allow pressure equalisation with the ambient atmosphere without contamination. The entire head-disc-actuator compartment is maintained at a slightly positive pressure to further ensure an ultraclean environment.

### 8.2.5 Media

The TM502 media consists of two (2) lubricated 130mm platters providing six recording surfaces.

The TM603SE media consists of three (3) lubricated 130mm platters providing six recording surfaces.

### 8.2.6 Storage Capacity

Storage capacities are listed in Table 1. Capacity is the maximum number of bytes that can be recorded irrespective of any gaps and formatting.

Table 1: Storage Capacities

| | TM502 | TM603SE |
|---|---|---|
| Capacity Unformatted: | | |
| per drive . . . . . . . . . . . | 12.75mb | 14.40mb |
| per surface . . . . . . . . . | 3.19mb | 2.40mb |
| per track . . . . . . . . . . | 10.40kb | 10.40kb |
| Number of: | | |
| platters . . . . . . . . . . . . | 2 | 3 |
| Active data surfaces . . . . . | 4 | 6 |

| | | |
|---|---|---|
| Maximum flux reversal density . . . . | 7690 FRPI | 9625 FRPI |
| Track density . . . . . . . . . . . . | 345 TPI | 255 TPI |
| Cylinders . . . . . . . . . . . . . . . | 306 | 230 |
| Tracks . . . . . . . . . . . . . . . . . | 1224 | 1380 |
| Read/Write heads . . . . . . . . . . . | 4 | 6 |
| Data Transfer Rate . . . . . . . . . . | 104kb/sec | 104kb/sec |

## 8.3 Winchester Drive Handling Precautions

Winchesters are delicate instruments that require proper care and handling. These units are expected to perform when needed. Misuse and/or mishandling will adversely affect the expected performance.

The Winchester drive presents a new set of problems to the field in the sense that much care must be taken in handling the product. Because of the small size and light weight Winchesters are much more susceptible to damage. This product can be carried in one hand and is easily taken for granted, thus making it an easy candidate for unintentional exposure to high shock forces during handling.

### 8.3.1 DO'S AND DON'TS

1. The hard disc unit should be placed on a side or end when not in a system or shipping container.

2. Never drop or jar a hard disc unit, place the unit carefully on foam padding on work or storage surfaces.

3. NEVER TURN DAMPER OR SPINDLE MOTOR BY HAND !!

## 8.4 Hard Disc System Diagnostics

Designed for use in the end user's environment the diagnostic diskette HDFIELD will help the field engineer diagnose hardware and media related problems in hard disc systems. Whether the problem is with the hard disc subsystem or any of the other major subsystems of the microcomputer HDFIELD's primary objectives are to:

1. Determine if the hardware components are defective or degraded.

2. Determine the existence of defective hard disc media which is not currently logged in the drive header label.

NOTE: Users of the current Sirius Level 1 diagnostic disc (LEV1P) will experience a BDOS error if attempting to use that particular disc in a hard disc system. When this occurs (if you wish to use these diagnostics) simply type ALT C, and then log on to the B drive by typing B:

Following is a brief description of the programs which are included in the HDFIELD package that exercise the hard disc subsystem:

**SHOWSTAT** Program SHOWSTAT reads the drive label and prints a summary of the label to the screen. This specifically includes a list of the current bad tracks. The distinction is made between bad tracks listed when the hard disc was initialised and those added after its initialisation. The number of bad tracks which have been added during normal usage should be monitored as an indication of disc drive performance degradation. Also displayed is the recorded serial number. Optionally, all displayed data can also be printed as hard copy output to the LST port.

**DMATEST** Program DMATEST tests the hard disc DMA interface to CPU board expansion bus. This testing is also performed in HDDISK under the DMA test (F2) option. The distinction is that DMATEST is limited to transferring data between the system main memory and hard disc controller memory. No drive access is performed.

**HDDISK** Program HDDISK is the hard disc test utility program. This utility allows the operator to test each component in the hard disc subsystem. The program is menu driven. The auto test (F1) is recommended as a start. If longer term tests are desired, either the random read (F5) or butterfly test (F6) should be executed. This program writes only to the pre-established inservice diagnostic track (if it can be identified).

## 8.5 Hard Disc Problems

1. When using the HDSETUP program, never assign the boot volume to the floppy. If you insert a floppy in the right-hand drive and close the door, and then press the reset button,

the system will boot from the floppy automatically. Assigning the boot volume to the floppy is, therefore, unnecessary and will cause the system to hang.

2. When creating device assignments, you must ensure that the assignments are contiguous. There must be no 'holes' in the list. We have discovered that a number of people were creating systems with Volumes A and B on the hard disc and volume F on the floppy. This will cause the system to hang the next time you boot and is a very difficult situation to recover from.

3. RDCPM and FORMAT are two utility programs which are designed to perform functions with the floppy. They were written in the days when Sirius 1 had only a floppy based system where the left drive was 'A' and the right drive was 'B'. Both of these programs adhere to this method of naming drives. Thus if you have a system with volumes A and B on the hard disc and volume C on the floppy, you would expect to address the floppy as drive C. Unfortunately, this is not the case. You must address the floppy as B. You might find it easier in this case to consider that the B stands for the right (physical) drive and not the logical volume.

4. In HDSETUP (version 1.0 or 1.1), only configure volume sizes of 2 megabytes, 4 megabytes or 10 megabytes. Any other values may result in problems.

# LOCAL AREA NETWORK

## 9.1 LOCAL AREA NETWORK

### 9.1.1 Introduction to Local Area Networking

Today's businesses require a computer system capable of multiple input and processing. These capabilities have been available in minicomputer systems for many years. In recent years the microcomputer has been replacing the minicomputer in office environments. This has forced micro systems to have the same multi input and processing features as minicomputer systems. Microcomputer systems are now being used in multi-terminal configurations to form local area networks (LAN).

LAN's are distributed processing systems that incorporate multiple user stations, mass data storage, and background printing facilities. there are basically three types of networks; centralised, decentralised and distributed. A centralised network has a "master" system controller controlling network operation. A decentralised network is basically a group of interconnected centralised networks. A distributed network has no main controller and each node on the system shares in the network control process.

The microcomputer operating system along with the network hardware allows the user multiple station input (multi-user) and concurrent processing (multi-tasking). The Sirius LAN will be initially offered in a multi-user, single tasking configuration. The network software for this configuration will be MS-DOS version 2.0 (servers) and MS-DOS version 1.25 for network stations. The network will (at a later date) be offered in a multi-user, multi-tasking configuration. The multi-tasking operating system will be based on the BELL LABS UNIX III operating system. Multi-tasking will allow concurrent processing of data, thus increasing system throughput.

The International Standards Organisation (ISO) and the American National Standards Institute have developed a seven layer hierarchical network model. The Sirius LAN board will implement the four lower levels.

### 9.1.2 ISO Seven Layer Network Model

Layer 1, Physical layer

The unit of exchange is the bit; considerations are voltage or current levels, signal timing, and connector pin assignments.

Layer 2, Data link layer

The unit of exchange is the frame, independent of any data content; considerations are error detection, frame acknowledgement, retransmission on errors, and duplicate frame detection.

Layer 3, Network layer

The unit of exchange is the packet; considerations are message/packet conversion, verification of receipt, etc.

Layer 4, Transport layer

The unit of exchange is the message; considerations are message ordering, host to host communication, etc.

Layer 5-7, Session Presentation and Application layers.

The unit of exchange is the message; considerations are applications oriented, such as billing, encryption, code compression etc.

The Sirius network software and applications software will implement the upper three layers.

Network communication protocol will be the packet. A packet may contain data, node addresses, error information and control information.

Each station on the network will need some method of accessing the network communications channel. The technique used will be carrier sense multiple access (CSMA). CSMA is a technique where a node requiring access to the channel will sense the channel for the presence of a carrier (activity), if no carrier is sensed for a predetermined amount of time, the node accesses the channel. If activity is sensed, the node will calculate a "waiting" period before trying to access the channel again. The "wait for retry" period is calculated only when no carrier is sensed to prevent nodes from queueing up to the channel.

The Sirius LAN system uses Positive message acknowledgement to ensure proper message reception. Positive message acknowledgement is a system in which all packet(s) transmitted MUST be acknowledged by the receiving station. If the packet is not acknowledged the host retransmits the packet until it is acknowledged or the packet retry limit is reached. CSMA along

with positive message acknowledgement should effectively eliminate collisions.

Mass data storage is an important feature of a LAN. The Sirius LAN mass storage units will be hard disk units with expanded memory (256K min.). These units will be called network servers. The network servers will also handle background printing of user files (spooling).

### 9.2 Sirius 1 Local Area Network Overview

The Sirius LAN is based on the Corvus OMNI-NET network system. The network supports 64 users (10 servers, 54 stations). The communications channel is a shielded, twisted pair cable with a maximum end to end length of 1.2km. The network data transfer rate approaches 1 mega-bit per second.

Each node on the network will contain a transponder board (network card). The network card is directly connected to the network communications channel and the host (node) computer's data bus. The network card will handle all network functions thus freeing the host processor from the duty of controlling the network. The network board communicates with the host system data bus. The network board contains a DMA (direct memory access) controller that allows the network card direct access to memory without host CPU intervention.

The network card will perform functions such as packet transmission and reception, packet formatting, error detection, and DMA transfers.

Each node on the network has a unique address. The address is switch selectable on the network card. The eight position DIP switch at location 1A is used to select the board address segment and set the interrupt priority level.

The network board default setting is at E810h (hexadecimal segment address). Switches 1-6 on switch 1A select the board segment address.

| Switch # | Address bit # |
|---|---|
| 1 | A12 |
| 2 | A11 |
| 3 | A10 |
| 4 | A9 |
| 5 | A13 |
| 6 | A14 |

Example: SW. 1-6 "ON" selects address segment E810h

Switches 7 and 8 are for selecting interrupt priority levels.

| Switch # | Interrupt Level |
|---|---|
| 7 (ON) | 4 |
| 8 (ON) | 5 |
| 7,8 (OFF) | Disable interrupts |

The eight position DIP switch at location 3M is used to select the user node address (0-9 servers, 10-63 stations). The switches are used to set a binary value from 0 to 63. A switch in the "ON" position represents a binary 0.

| Switch # | Value |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 8 |
| 4 | 4 |
| 5 | 16 |
| 6 | 32 |
| 7,8 | not used |

Example: switches 1,3,5 "ON" all others "OFF" selects node address 38.

The network card can be functionally broken down into four parts:

1. DMA controller
2. MC6854 advanced data link controller
3. 6801 microprocessor
4. RS-422 transceivers SN74174/SN74175

The DMA controller chip is a custom gate array chip designed to control the interface between the Sirius computer bus and the Network card. Each DMA cycle is explicitly invoked by the 6801 microprocessor allowing the network software complete control of the DMA transfer between the Sirius computer and the network card.

The MC6854 advanced data link controller (ADLC) controls many of the functions specified in the ISO data link and network layers. The ADLC performs such functions as data serialisation, error code detection (CRC) and generation, packet framing, bit protocol implementation (NRZI non return to zero inverted) and zero insertion.

The 6801 microprocessor oversees the operation of the DMA controller and the ADLC. The 6801 controls the transfer of data and control information between the Sirius computer and the Network card.

The SN75174/75175 chips form the RS-422 transceiver. The 75174 is a differential line driver, and the 75175 is a differential line receiver. Because the driver and receiver use differential circuits, they offer high noise immunity. The twisted wire pair was chosen as the communications medium because of its ease of installation and RFI (radio frequency interference) immunity level.

## 9.3 Network Software Overview

The Server Network product supports a local network of Sirius workstations with one or more network servers providing mass storage and printer access for network users. The network servers are dedicated to providing services for network users and cannot be used as work stations. The network can include discless network stations that boot from the network and have all their mass-storage on the network server(s). Each network server supports a ten megabyte hard disc and double-sided floppy disc drive, but does not need a screen or keyboard.

The network servers run network software under the MS-DOS 2.0 operating system while the network stations run a network interface in conjunction with MS-DOS 1.25. The appearance of the file system on a network server is transparent to programs running on the work station, with the network server's hierarchical file system being used to provide each user with private directories. Except for the private directories, all directories on a network server are treated as common storage and can be shared by one or more network users.

The Server Network product satisfies the following requirements:

1. The product supports discless network stations.

2. The product supports multiple network servers.

3. The product supports existing MS-DOS based applications consistently.

4. Each network station user can keep private files on the servers.

5. Network stations can have common access to public files on network servers.

6. The product does not require a "super user" or system administrator. Configuration of the system and addition of network stations requires only basic MS-DOS 1.25 operation and usage familiarity.

7. Network server and station failures do not bring down the network, as long as at least one network server and station are operative.

8. The product supports background printing of files on the network servers.

An INSTALL program adds users to the network by taking an eleven-character user name, optional password, and assigning drive designators to link the user to disc volumes on the network servers. The INSTALL program takes the available drive designators (those not used for local work station storage) and sequentially assigns them to network server volumes with private and common directories being assigned in a ratio of one to two. The assignments are displayed, and can be changed if desired. Choosing the default assignments ensures that all users have consistent links to all network server volumes. Network installation is simplified by the use of standard AUTOSET files to configure network server hard disc volumes.

Common directory assignments can be made to a subset of network users allowing groups that work on common data to share files that, by installation conventions, are unavailable to other users of the network. Login to the network can be performed automatically if the network station is not shared among several people, or a user can be required to login by giving his or her name and password (if required). Utilities can be run to check the status of the network, list the network users, print files on the network server, or reserve files for exclusive access.

The network server supports three protection schemes for basic file sharing in common directories.

1. Files on a network server can be set to read/only using the PROTECT command. Any writes to a read/only file fail with a "write protect" error.

2. For read/write files, an automatic mechanism prevents concurrent access to a file that is being written or updated. Writes to the file succeed only if no one else has

opened the file, and effectively lock the file so that no one else can open or modify it until the writing process closes the file. Attempts to write to a file opened by two or more users generates a write protect error, while attempts to open a file that is write-locked generate a "file locked or reserved" error. This mechanism can be turned off for applications capable of managing their own concurrent access to a file or files.

3. A user can reserve a set of files that are located on a network server using the RESERVE command. The reserve command ensures that no other network station will be able to open or modify the files until they are released by the user with the RELEASE command. This facility allows a set of files to be updated without the danger of concurrent access by another user or an unexpected error from the automatic mechanism described above. Attempts to open a file reserved by another user result in a "file locked or reserved" error. A RESERVE-KEY option allows a process to reserve an arbitrary semaphore for applications that provide their own concurrent access management.

To facilitate sharing of up to three printers connected to a network server the network station interface can redirect list output at the network station to a specified file on a network server for later printing.

Network server volume organisation is determined by choosing an AUTOSET file or using HDSETUP with the server machine configured as a local work station (keyboard and screen connected). AUTOSET configuration can be performed by creating a special configuration disc, keyed to the server number, that automatically formats the server when booted. This alleviates the need to have a screen and keyboard connected to the server.

Each server's hard disc can be configured as a number of logical disc volumes, but must have at least two. These are assigned to A: and C: (B: is used for the floppy). The hard disc is optimised for the smallest possible allocation unit size without regard for memory usage (this is why 256K RAM is required in network servers). Small allocation unit sizes have the effect of increasing the effective size of the disc and allow more files to be stored for the network users.

A network server appears as a set of remote volumes to programs running on the network stations, with the server's hierarchical directories providing private directories for each user. A users private directory is a sub-directory on a network

server volume named with the user's eleven character login name. Each volume's root directory contains the common files for that volume. No change directory command is provided, so that network users do not have to understand hierarchical directories, and, they cannot access any sub-directories that have not been assigned to them by INSTALL. This ensures that other user's sub-directories are not accessible, and remain private to their owners.

The network station accesses remote volumes on the network by using standard MS-DOS drive designators (A: to O:). If a network station has local disc drives, the remote volumes should use different drive designators than the local drives, although the INSTALL program does not prohibit this.

The file system is composed of all network servers on the network. Each network server has its hard disc volumes partitioned into multiple directories. The private directories of each user are only accessible by that user. MS-DOS 2.0 will enforce "read/only" or "read/write" protection on all files on both regular and private directories. These file attributes are setable by anyone able to access that file (only the user for private directories, and anyone with a drive assignment for common directories).

A network server may have attached printers, or other output devices which may be in demand by multiple network users. The devices may be connected to the server through the parallel interface or through Serial A or Serial B RS232 ports. In order to share these devices output is spooled to the network server's disc. The names of the spool files and the user names are placed into a queue maintained by the network server for each printer attached to the server.

# HIGH RESOLUTION GRAPHICS

## 10.1 HIGH RESOLUTION GRAPHICS

In the high resolution mode, all 16 bits of each font cell word are displayed. The screen buffer is filled with pointers to successive cells in dynamic RAM and the programmer must manipulate the contents of dynamic RAM to create the required display. To set up a hi-res screen, there are 3 steps which must be performed.

1. A 40k byte region in dynamic RAM should be chosen and cleared for use as a high-res screen area.

2. The screen buffer is then filled with pointers to this area of dynamic RAM.

3. The CRT controller is reprogrammed to give the correct timing for the hi-res mode.

## 10.2 Clearing a Hi-Res area

The starting address of the hi-res screen must be on a 32 byte boundary (ie. an address that is divisible by 32). This is because the lower 4 address bits are used by the CRT controller to address the 16 words of the font cell. The hi-res screen must be contained completely in the upper or lower 64K segment of the first 128K block of memory (0 - 1FFFFh).

As CP/M loads into the top of memory and loads programs directly below itself, a convenient place for a hi-res screen is directly above the operating system character table at location 2C00h (see Memory Allocation Map of CP/M system, section 1.6.2).

This area can also be used under MS-DOS except that this area above the character table must be claimed by the user. This is because the programs are loaded directly above the character table under MS-DOS. It is therefore necessary for the user to include a 40,000 byte buffer at the start of the software. It is also necessary for the user to ensure that the start of the buffer lies on a 32 byte boundary. This can be done by loading the ES segment register contents and ensuring that it is divisible by 32 with no remainder.

The following routine clears 40,000 bytes starting at location 2C00h.

```
      MOV  BX,02C0H         ;ES segment points to our hi-res screen
      MOV  ES,BX
      MOV  BX,0             ;use BX to index the 40k byte RAM area
      MOV  CX,4E20H         ;counter for 20,000 words
      MOV  AX,0             ;we  want to store 0 throughout the  RAM
                            ;area
CLEAR:MOV  ES:[BX],AX
      INC  BX               ;next word
      INC  BX
      LOOP CLEAR
```

## 10.3 Setting the Screen Buffer Pointers

We have defined our high resolution screen to start at address 2C00h and so our first pointer must be 2C00h divided by 32. (ie. the word in screen RAM represents the upper 11 bits of address to dynamic RAM; the CRT controller supplies the lower 4 bits). This means the screen pointers should start at 0160h. Incrementing this number by one translates to an increment in the address of 16 (ie. one complete cell). The following routine fills the screen buffer with pointers to our high resolution screen. Note we only fill 25 rows x 50 columns = 1250 bytes of the 4096 bytes available in the buffer.

```
          MOV  BX,0F000H        ;address screen buffer RAM
          MOV  ES,BX
          MOV  BX,0             ;BX indexes the 4k RAM area
          MOV  CX,0432H         ;1250 byte counter
          MOV  AX,0160H         ;starting address of DRAM pointers
STORE:
          MOV  ES:[BX],AX       ;store font pointer
          INC  BX               ;next word
          INC  BX
          INC  AX               ;next DRAM pointer
          LOOP STORE
```

## 10.4 Reprogramming the 6845 CRT Controller

In order to derive the correct timing from the display circuit, 16 of the internal registers should be reloaded. The correct data for both the text and high resolution modes are shown on page 99 of the Hardware Reference Manual. The following routine transfers the 16 bytes of data in the table to the 16 internal registers of the CRT controller.

```
          mov  bx,0E800h        ;address CRT controller
          mov  ex,bx
          mov  bx,0
```

```
        mov  si,0001h
        mov  cx,offset data ;pointer to register string
        mov  dl,0
loop:   mov  al,dl          ;set address register AR
        inc  dl             ;address next register
        mov  es:[bx],al
        xchg bx,cx          ;point to data
        mov  al,[bx]        ;get data byte
        xchg bx,cx
        inc  cx             ;address next byte
        mov  es:[bx+si],al  ;set register
        cmp  dl,11h         ;last register ?
        jnz  loop           ;no: address next register

data    db   3Ah,32h,34h,0C9h,19h,06h,19h
        db   19h,03h,0Eh,20h,0Fh,20h,0,0,0
```

## 10.5 Examples

### 10.5.1 Microsoft MACRO-86 Assembly Language

This assembly-language routine demonstrates two things, one, interfacing an assembly-language routine to the MS-Basic interpreter and two, using the high-resolution graphics from MS-Basic.

1. To interface an assembly-language routine to MS-Basic the routine must load itself into memory then exit and remain resident (int 27h). This is done before Basic is loaded. The routine must locate its position in memory and report this either to the programmer or to the subsequent Basic program. This example does the former. It determines the value of the CS register which is to be used in a Basic DEF SEG statement and the entry point for Basic to be used as an OFFSET.

   The program prints these values on the screen in a form which can be used by Basic directly. The form is:

   ```
   10 DEF SEG=&Hxxxx
   20 HI.RES=&Hyyyy
   30 CALL HI.RES
   ```

   where xxxx is the hexadecimal value of CS and yyyy is the hexadecimal value of the offset within the segment. A Basic program including the above code execute the code starting at location INIT: in the listing. The alternative method is for the routine to leave these values in some known memory

location which can be interrogated by a Basic program. Suitable locations are to be found in the Interrupt Vector Table (see Appendix I.2), namely interrupts 80h to BFh, which are reserved for use by application programs. See the IEEE-488 Toolkit, Audio Toolkit and Network Users Guide for examples.

2. Details of how the high-resolution screen works can be found in the Hardware Reference Manual and the Supplementary Technical Reference Manual. This program sets aside a 40k (decimal) buffer (initialised to zero) for the high-resolution screen, which must start on a 32-byte boundary. The software locates the first available 32-byte boundary and reports this to the programmer. When called from Basic the routine initialises the screen pointers and the CRT controller then returns to Basic. This routine could be extended to include specialised graphics functions for particular applications where the Grafix Kernel (found in the Graphics Toolkit) either does not provide the required functions or is too general. Provided this extended routine and the application package are not too large, they could easily run in a 128K machine.

When Basic calls an assembly-language routine it pushes the return address and the value of its CS onto the stack, then enters the called routine. This routine must save any other segment registers it intends using. This example uses DS and ES.

```
code            segment
                assume  cs:code,ds:code
; Written by    Greg Johnstone and Keith Rea
;               Barson Computers
;               335 Johnston St.
;               Abbotsford, 3067
;
;               16-9-83
;
; Program HI-RES
;
;
; Assemble this program as follows
;
;       MACRO86 HI-RES;
;       LINK HI-RES;
;
; Link will produce a warning - 'No stack segment', ignore it. Link
; produces a file HI-RES.EXE which will not run. You must produce a .COM
; file as follows:
```

```
;
;       DEBUG HI-RES.EXE
;       N HI-RES.COM
;       W
;       Q
;
; The resulting program HI-RES.COM will run
;


buf_st  equ     190h            ; offset to hi-res buffer
scrn_rm equ     0f000h          ; segment address of screen RAM
count   equ     1250            ; no. of cell locations in hi-res mode
crt     equ     0e800h          ; segment address of CRT controller
bdos    equ     21h             ; MS-DOS function call
boot    equ     00h             ; exit to MS-DOS function.
conout  equ     02h             ; console output function.
cr      equ     0dh             ; carriage return
lf      equ     0ah             ; line feed
esc     equ     1bh             ; escape

        org     100h

loader  proc    near

start:
        call    main            ; find out where routine loaded in RAM
;
; Exit and remain resident
;

        mov     dx,offset top+1 ; Set DX to the top of part to remain ..
        int     27h             ; .. resident then quit.

loader  endp

basic   proc    far

; Set screen RAM pointers to font RAM work area

init:
        push    es              ; save segment registers used by Basic ..
        push    ds
        mov     ax,cs           ; ..and point DS to current CS
        mov     ds,ax
        mov     bx,scrn_rm      ; point to screen ram
        mov     es,bx
        mov     bx,0
```

```
        mov     cx,count            ; counter for 1250 cells
        mov     ax,hi_res           ; starting address of pointers ..
        shr     ax,1                ; .. = RAM address/2
init1:
        mov     es:[bx],ax          ; store font pointer
        inc     bx                  ; address next word and put ..
        inc     bx
        inc     ax                  ; .. next font cell
        loop    init1

; set CRT controller for high resolution

        mov     bx,crt              ; point to CRT controller
        mov     es,bx
        mov     bx,0
        mov     si,1
        mov     cx,offset data      ; point to register string
        mov     dl,0

init2:
        mov     al,dl               ; set address register AR
        inc     dl                  ; address next register
        mov     es:[bx],al
        xchg    bx,cx               ; point to data
        mov     al,[bx]             ; get data byte
        xchg    bx,cx
        inc     cx                  ; address next byte
        mov     es:[bx+si],al       ; set register
        cmp     dl,11h              ; last register?
        jnz     init2               ; no, address next register

        pop     ds                  ; restore Basic's segment registers
        pop     es

        ret

data:   db      3ah,32h,34h,0c9h,19h,06h,19h
        db      19h,03h,0eh,20h,0fh,20h,0,0,0
enddata:

        org     buf_st              ; start graphics buffer on 16 ..
                                    ; .. byte boundary. Initialise to 0
                                    ; reserve extra 16 bytes just in case ..
buff:   db      40016 dup(0)        ; .. it falls on an odd boundary.
hi_res: dw      0                   ; segment address of hi-res RAM
top:                                ; routine up to here needs to stay in RAM.

basic   endp
```

```
load2   proc    near

; Now we have to figure out where we are in memory and tell everyone
; about it.

main:

        call    msprnt          ; Print the following message

        db      esc,'E','HI-RES ver. 1.0',cr,lf
        db      cr,lf,'Hi-res graphics interface for MS-Basic interpreter'
        db      cr,lf,cr,lf
        db      'Include the following statements in your program',cr,lf,cr,lf
        db      '       10 DEF SEG=&H',0

;
; get the segment address
;

        mov     ax,cs           ; Get the contents of CS register ..
        call    hexprnt         ; .. convert to hex and print it

        call    msprnt          ; print next message

        db      cr,lf
        db      '       20 HI.RES=&H',0

;
; get the entry point for Basic
;

        mov     ax,offset init  ; Get offset of the start of the ..
        call    hexprnt         ; .. routine and print it.

        call    msprnt

        db      cr,lf
        db      '       30 CALL HI.RES'
        db      cr,lf
        db      'The hi-res screen starts at segment address &H',0

;
; now lets find where we put the hi-res screen
;

        mov     ax,cs           ; point to CS
        add     ax,buf_st/16    ; add offset to start of hi-res ram
        test    al,1            ; check if on 32-byte boundary. Segment ..
```

```
        jz      mainl           ; .. address should be even if it is.
        inc     ax              ; if odd, make it even ..
mainl:
        mov     hi_res,ax       ; .. and save it.
        call    hexprnt
        call    msprnt

        db      cr,lf,'Use DEF SEG to point to this location in memory and'
        db      cr,lf,'start POKEing data into hi-res RAM',cr,lf,0

        ret
;
; subroutine MSPRINT - print bytes following the CALL till zero
;

msprnt:
        pop     bp              ; get message starting address
        mov     al,cs:[bp]      ; get byte pointed to by bp
        inc     bp              ; next byte
        push    bp
        and     al,al           ; is the byte equal to 0 ?
        jnz     msprntl         ; no: print it
        ret                     ; yes: end of string

msprntl:
        mov     dl,al           ; console output routine
        mov     ah,conout
        int     bdos
        jmp     msprnt
;
; subroutine HEXPRNT - convert a byte to hex and print it
;

hexprnt:
        push    ax
        xchg    ah,al
        call    hexprntl
        pop     ax
        call    hexprntl
        ret

hexprntl:
        mov     dl,al           ; save the byte
        and     al,0fh          ; get lower nibble
        mov     cl,04h          ; then the upper nibble
        sar     dl,cl
        and     dl,0fh
        cmp     dl,09h          ; if less than 10 then get the ..
```

```
        jg      hexprnt2        ; .. ASCII value of the numeric digit
        add     dl,30h
        mov     hi_byte,dl
        jmp     hexprnt3

hexprnt2:
        add     dl,37h          ; else get the ASCII value of the ..
        mov     hi_byte,dl      ; .. alpha digit

hexprnt3:
        cmp     al,09h          ; repeat for upper nibble
        jg      hexprnt4
        add     al,30h
        mov     lo_byte,al
        jmp     hexprnt5

hexprnt4:
        add     al,37h
        mov     lo_byte,al

hexprnt5:
        mov     dl,hi_byte      ; print the hex. bytes, one at a time
        mov     ah,conout
        int     bdos
        mov     dl,lo_byte
        mov     ah,conout
        int     bdos
        ret

lo_byte db      '0'             ; hex value of lower nibble (in ASCII)
hi_byte db      '0'             ; hex value of upper nibble (in ASCII)

load2   endp

code    ends
        end     start
```

### 10.5.2 Microsoft MS-BASIC Interpreter

This program calls the routine given in section 10.5.1.

```
100 ROUTINE=&H2C8       'Segment address of HI-RES
110 SCREEN=&H2E2        'Segment address of hi-res screen
120 PRINT CHR$(27);"z"  'Clear screen
130 PRINT CHR$(27);"x5"
140 DEF SEG=ROUTINE     'Point to start of HI-RES segment
150 HI.RES=&H108        'Offset to HI-RES
```

```
160 CALL HI.RES         'Initialise hi-res screen
170 GOSUB 250           'Throw some data in it
180 A$=""               'Hang about till someone presses a key
190 WHILE A$=""
200 A$=INKEY$
210 WEND
220 PRINT CHR$(27);"z"  'Put screen back into character mode
230 END                 'And that's it folks
240 '
250 DEF SEG=SCREEN      'Point to start of hi-res screen
260 FOR I=0 TO 31 STEP 2'Start filling hi-res RAM, a word at a time
270 READ HI,LO  'Remember that the screen displays bit 0 first
280 POKE I,LO: POKE I+1,HI
290 NEXT
300 RETURN
310 'This data prints the letter G in the top left corner
315 ' of hi-res screen
320 DATA 255,255,128,1,128,1,128,1,0,1,0,1,0,1,0,1
330 DATA 0,1,255,1,128,1,128,1,128,1,128,1,128,1,255,255
```

## 10.6 Printer Configuration Tables in the Grafix Kernel

```
; Printer  configuration tables for Grafix  ver.
; 1.2.  To get addresses for Grafix ver. 1.3 add
; 10h to each address.
;
; The  printer configuration tables contain  the
; following information.
;
; prttyp - printer  algorithm  type.  Currently,
;          all  printers use the same algorithm,
;          which is defined as type 1.   This is
;          the sequence of events:
;             1) send   initialisation    string
;                (initstr)
;             2) send  beginning of line  string
;                (bolstr)
;             3) send graphics string 1 (grstr1)
;             4) send    count    of    graphics
;                characters
;             5) send graphics string 2 (grstr2)
;             6) send graphics characters
;             7) send end of line string (eolstr)
;             8) if more data, go to 2
;             9) send final string (finalstr)
;
; grcnt_typ - graphics  character  count  type.
```

```
                        ;           There are currently 3 count  types
                        ;           implemented.  1 = 2 hex bytes sent
                        ;           as count.
                        ;           2  =  3 ASCII  numeric  characters
                        ;           sent as count
                        ;           3 = no count is sent
                        ;
                        ; eneedles -  the  number of needles used in the
                        ;           print    head  to  print   graphics
                        ;           characters.  This  is positive  if
                        ;           the   top   dot    is    the   least
                        ;           significant bit of the data  sent.
                        ;           negative   if   it  is  the   most
                        ;           significant bit.
                        ;
                        ; rtmin - the minimum number of characters  that
                        ;         will be printed on a line. If the user
                        ;         data  contains fewer  characters,  the
                        ;         remainder sent will be blank (spaces).

 = 0000                 nul             equ     0
 = 0002                 stx             equ     2
 = 0003                 etx             equ     3
 = 000A                 lf              equ     10
 = 000D                 cr              equ     13
 = 000E                 so              equ     14
 = 001B                 escape          equ     27
 = 0020                 space           equ     32
 = 00FF                  end_tblf       equ     0ffh     ;end   of  table
                                                         ;flag
 = 000A                  tbl_entry_lgth equ     10       ;maximum   table
                                                         ;string length


                         ;------------------------------------------
                         ;--    Epson MX80/MX100 Configuration Data   --
                         ;------------------------------------------

 = 3B41                  epson_tbl      equ     $

 3B41   01               ep_prttyp      db      1  ;epson algorithm typ
 3B42   01               ep_grcnt_typ   db      1  ;epson format of
                                                   ;graphics bytes
 3B43   0008             ep_eneedles    dw      8  ;Epson num of scan
                                                   ;lines/printed line
 3B45   0000             ep_rtmin       dw      0  ;epson min line
                                                   ;length to print
 3B47   04 0A 1B 41 08   ep_initstr     db      4,lf,escape,'A',8;epson
                                                   ;init string
```

```
3B4C  02 1B 32        ep_finalstr   db    2,escape,'2' ;epson
                                             ;final string
3B4F   00  00         ep_bolstr     db    0,nul    ;epson beginning
                                             ;of line string
3B51  02 0D 0A        ep_eolstr     db    2,cr,lf      ;epson end
                                             ;of line string
3B54  02 1B 4C        ep_grstrl     db    2,escape,'L' ;epson
                                             ;graphics mode
3B57  00 00           ep_grstr2     db    0,nul        ;epson
                                             ;graphics mode
3B59  FF              ep_endf       db    end_tblf     ;end of
                                             ;table flag


                      ;-----------------------------------------------
                      ;--  Tally Configuration Data                 --
                      ;-----------------------------------------------

= 3B5A                tally_tbl     equ   $

3B5A  01              t_prttyp      db    1            ;tally
                                              ;algorithm type
3B5B  01              t_grcnt_typ   db    1            ;tally
                                              ;format of graphics
                                              ;bytes
3B5C  FFF8            t_eneedles    dw    -8           ;tally num
                                              ;of scan lines/printed
                                              ;lines
3B5E  00C8            t_rtmin       dw    200          ;tally min
                                              ;line length to
                                              ;print
3B60  00 00           t_initstr     db    0,nul        ;tally init
                                              ;string
3B62  00 00           t_finalstr    db    0,nul        ;tally
                                              ;final string
3B64  00 00           t_bolstr      db    0,nul        ;tally beg
                                              ;of line string
3B66  02 0D 0A        t_eolstr      db    2,cr,lf      ;tally end
                                              ;of line string
3B69  02 1B 4C        t_grstrl      db    2,escape,'L' ;tally
                                              ;graphics mode
3B6C  01 20           t_grstr2      db    1,space      ;tally
                                              ;graphics mode
3B6E  FF              t_endf        db    end_tblf     ;table end
                                              ;flag
```

```
                        ;-----------------------------------------
                        ;--      C. Itoh Configuration Data      --
                        ;-----------------------------------------

 = 3B6F                 citoh_tbl     equu    $

3B6F  01                c_prttyp      db      1           ;c.itoh
                                                  ;algorithm type
3B70  02                c_grcnt_typ   db      2           ;c.itoh
                                                  ;format  of graphics
                                                  ;bytes
3B71  FFF8              c_eneedles    dw      -8          ;c.itoh num
                                                  ;of scan lines/printed
                                                  ;line
3B73   0000             c_rtmin               dw      0          ;c.itoh
                                                  ;line length  to
                                                  ;print
3B75  08 1B 45 1B 54 31 c_initstr     db 8,escape,'E',escape,'T16',cr,lf
      36 0D 0A                                    ;c.itoh init string
3B7E  04 1B 4E 1B 41    c_finalstr    db      4,escape,'N',escape,'A'
                                                  ;c.itoh final string
3B83  00 00             c_bolstr      db      0,nul       ;c.itoh beg
                                                  ;of line string
3B85  02 0D 0A          c_eolstr      db      2,cr,lf     ;c.itoh end
                                                  ;of line string
3B88  03 1B 53 30       c_grstrl      db      3,escape,'S0';c.itoh
                                                  ;graphics mode
3B8C  00 00             c_grstr2      db      0,nul       ;c.itoh
                                                  ;graphics mode
3B8E  FF                c_endf        db      end_tblf    ;table  end
                                                  ;flag

                        ;-----------------------------------------
                        ;--    Okidata Configuration Data        --
                        ;-----------------------------------------
                        ;
                        ; The  Okidata printer has a 7 dot graphics head
                        ; instead  of 8 dots.  Since our characters  are
                        ; 10x16,  the Okidata is programmed to print the
                        ; dots out in 4 rows of 4 dots each (instead  of
                        ; 2 rows of 7 dots, with 2 dots left over). This
                        ; allows  the program to use the same  algorithm
                        ; type  for the Okidata as it uses for the other
                        ; printers.  However,  it  is  slower  since  it
                        ; requires 4 passes for every line of characters
                        ; instead of two passes.

 = 3B8F                 okidata_tbl   equ     $
```

```
3B8F  01          ok_prttyp     db   1           ;okidata
                                                 ;algorithm type
3B90  03          ok_grcnt_typ  db   3           ;okidata
                                                 ;format of graphics
                                                 ;bytes
3B91  FFF9        ok_eneedles   dw   -7          ;okidata
                                                 ;num of scan
                                                 ;lines/printed line
3B93  0001        ok_rtmin      dw   1           ;okidata
                                                 ;line length to
                                                 ;print
3B95  01 1D       ok_initstr    db   1,29        ;okidata
                                                 ;init string - 12
                                                 ;chars/inch
3B97  02 03 02    ok_finalstr   db   2,etx,stx   ;okidata
                                                 ;final string
3B9A  00 00       ok_bolstr     db   0,nul       ;okidata
                                                 ;beg of line string
3B9C  02 03 0E    ok_eolstr     db   2,etx,so    ;okidata
                                                 ;end of line string
3B9F  01 03       ok_grstrl     db   1,etx       ;okidata
                                                 ;graphics mode
3BA1  00 00       ok_grstr2     db   0,nul       ;okidata
                                                 ;graphics mode
3BA3  FF          ok_endf       db   end_tblf    ;table end
                                                 ;flag
```

## 10.7 Patching the Grafix Kernel for the MT-180 Printer

```
A>DEBUG GRAFIX.COM

DEBUG-86  VERSION 1.07
>d3b00
0473:3B00 2E A0 3F 0E 2E 08 06 1A-0B C3 00 00 00 00 00 00 . ?......C......
0473:3B10 0A 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0473:3B20 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0473:3B30 00 00 00 00 32 00 00 00-00 00 00 00 00 00 00 00 ....2...........
0473:3B40 00 01 01 08 00 00 00 04-0A 1B 41 08 02 1B 32 00 ..........A...2.
0473:3B50 00 02 0D 0A 02 1B 4C 00-00 FF 01 01 F8 FF C8 00 ......L.....x.H.
0473:3B60 00 00 00 00 00 00 02 0D-0A 02 1B 4C 01 20 FF 01 ...........L....
0473:3B70 02 F8 FF 00 00 08 1B 45-1B 54 31 36 0D 0A 04 1B .x.....E.T16....
>e3b5c
0473:3B5C F8.08   FF.00   C8.00   00.
0473:3B60 00.01   00.0D   00.01   00.0C   00.     00.     02.     0D.

0473:3B68 0A.     02.03   1B.     4C.25   01.34   20.00   FF.00   01.FF
>d3b00
```

```
0473:3B00 2E A0 3F 0E 2E 08 06 1A-0B C3 00 00 00 00 00 00 . ?......C......
0473:3B10 0A 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0473:3B20 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0473:3B30 00 00 00 00 32 00 00 00-00 00 00 00 00 00 00 00 ....2...........
0473:3B40 00 01 01 08 00 00 00 04-0A 1B 41 08 02 1B 32 00 ..........A...2.
0473:3B50 00 02 0D 0A 02 1B 4C 00-00 FF 01 01 08 00 00 00 ......L.........
0473:3B60 01 0D 01 0C 00 00 02 0D-0A 03 1B 25 34 00 00 FF ...........%4...
0473:3B70 02 F8 FF 00 00 08 1B 45-1B 54 31 36 0D 0A 04 1B .x.....E.T16....
>ngr180.com
>w
Writing 5409 bytes
>q
A>
```

Procedure for modifying GRAFIX Ver 1.2 to work with the Tally MT180 printer. This modifies the MT140 driver to use MT180 escape codes. To invoke the new version for BUSIGRAF, for example, type:

GR180 $S1C3PS <cr>

NOTE that this modification clobbers the ability of GRAFIX to operate the C.ITOH printer, so you should keep your original copy of GRAFIX in case you wish to use a C.ITOH at a later date.

This modification also prints a Formfeed after the graphics dump is finished. If you do not want this feature then change bytes 3B62 and 3B63 to 00.

To patch Grafix Ver 1.3, add 10h to each of the above addresses. That is, start changing bytes at location 3B6C.

## 10.8 Character Printing

One of the Sirius 1's most useful graphics features seems to be greatly overlooked: its ability to do character graphics printing. If you design a keyboard file using Keygen that has special non-ASCII characters, and you try printing a document in the standard fashion, you will find that these special characters will not be printed.

Suppose, for instance, that you have loaded the future character set (FUTURE.CHR) into your system from the Graphics Toolkit. All the characters displayed on the CRT are in the future type, but when you try printing the file, you will find that the printed characters are once again the standard ASCII type. To remedy this, you must use the CHRPRINT.EXE file, also found on the Graphics Toolkit.

CHRPRINT.EXE causes documents to be printed in the same style in which they appear on the screen. This utility will work with any file, as long as the file has been saved in a standard ASCII format; thus, any file created with EDLIN, PMATE, BENCHMARK or WORDSTAR (using non-document mode), or any file just copied from the screen to a file, can be printed with a dot matrix printer in any number of character styles and scripts.

To call up CHRPRINT, type:

CHRPRINT filename<cr>

The computer will ask you to identify your printer from a menu displayed on the screen; once you have done this, the document will start printing.

**10.9 Patching CHRPRINT for MT-180**

```
A>ren chrprint.exe=temp

A>debug temp

DEBUG-86 version 1.07
>d06c0
0473:06C0  02 0D 0A 02 1B 4C 00 00-46 01 01 F8 FF C8 00 00   .....L..F..x.H..
0473:06D0  00 00 00 00 00 02 0D 0A-02 1B 4C 01 20 46 01 02   ..........L. F..
0473:06E0  F8 FF 00 00 08 1B 45 1B-54 31 36 0D 0A 04 1B 4E   x.....E.T16....N
0473:06F0  1B 41 00 00 02 0D 0A 03-1B 53 30 00 00 46 01 03   .A.......SO..F..
0473:0700  FC FF 00 00 03 1D 0D 0A-07 1B 25 39 00 1E 0D 0A   |.........%9....
0473:0710  00 00 08 03 02 1B 25 39-08 0D 0A 01 03 00 00 46   ......%9.......F
0473:0720  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0473:0730  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
>e06b
0473:06CB  F8.08   FF.00   C8.00   00.     00.01
0473:06D0  00.0d   00.01   00.0c   00.     00.     02.     0D.     0A.
0473:06D8  02.03   1B.     4C.25   01.34   20.00   46.00   01.46   02.01
0473:06E0  f8.02
>w
Writing 0980 bytes
>q

A>ren temp=chrprint.exe
```

N.B. This fix probably clobbers other printer drivers so keep a spare copy of the original.

# ASSEMBLY TO HIGH-LEVEL INTERFACE

## 11.1 Interfacing Basic with Assembly Language

Sometimes it may be desirable to write certain subroutines in assembly language instead of BASIC because of speed, size, or other constraints. This section explains how to successfully combine BASIC programs with assembly language modules so that parameters are passed correctly between the BASIC program and the assembly language subroutine.

To understand this section fully, the reader should have a basic knowledge of the following:

1) The BASIC interpreter

2) The assembler

3) The register structure of the 8086/8088

### 11.1.1 Calling Assembly Language Subroutines

In order to call an assembly language subroutine from an interpretive BASIC program, it is necessary to know the address of the assembly language routine. The routine must be resident in memory when BASIC is loaded, and its entry address must be known. The module containing the subroutine should also contain a short program that loads the module into memory, using the MS-DOS terminate and remain resident function (Int 27 Hex). When the program is run, it loads the subroutine into memory permanently. The program should also display the entry address of the subroutine, or store the entry address to some specific memory location that the BASIC program can PEEK in order to determine where the routine is. A possible location to store this information is an Interrupt Vector Table entry, but be very careful not to use an entry that is used by the operating system! Interrupt Vector Table entries available for use include 128 through 191 (80 - BF Hex). Since each entry is four bytes long, entry 128 is at memory address 0:200 Hex, entry 129 is at address 0:204 hex, and so on, with entry 191 at address 0:2FC Hex.

The BASIC program can then determine the address of the subroutine by PEEKing the four consecutive bytes that were saved by the assembly loader program. After the BASIC program has PEEKed these locations, it can set up the entry address of the assembly language subroutine. The following program segment shows how this is done.

```
10 '
20 '   do  a  DEF  SEG  to the segment  where  the  entry
25 '   address is stored
30 '   LOCATION  =  offset address of the  entry  address
35 '   location
40 '
50 LOWOFF = PEEK(LOCATION)
60 HIOFF  = PEEK(LOCATION+1)
70 LOWSEG = PEEK(LOCATION+2)
80 HISEG  = PEEK(LOCATION+3)
90 ASM.SEG = (256*HISEG)+LOWSEG
100 SUBROUTINE = (256*HIOFF)+LOWOFF
110 DEF SEG = ASM.SEG
```

After these statements have been executed, calls to the assembly language subroutine can be performed as follows:

```
150 CALL SUBROUTINE(PARAMETER1, PARAMETER2, ...)
```

The assembly language subroutine must follow some simple rules in order to work correctly.

1) It must be declared FAR.

2) Segment registers DS and ES must be restored to their entry values before returning to BASIC.

3) The general purpose registers (AX, BX, CX, DX, SI, DI, and BP) can have any value when returning to BASIC.

4) The assembly language routine MUST NOT change the length of any BASIC strings.

5) The assembly language routine must perform a RET <n> (where n = 2 times the number of parameters) to restore the stack pointer to its proper value.

6) Values can be returned to BASIC by passing a parameter that the result will be returned in.

11.1.2 Basic Data Types

It is necessary to understand how the various data types are represented in memory. When a subroutine is called, BASIC will pass the address of one of the following data representations.

1) Integer - two byte two's complement number

2) Single Precision Number - four byte binary floating point quantity. The most significant byte contains the value of the exponent minus 127. The remaining three bytes contain the mantissa. The most significant byte of the mantissa contains the sign bit, followed by the seven highest bits of the mantissa. A positive number is represented with a 0 as the sign bit, a negative number with a 1 as the sign bit. The binary point is to the left of the most signifi- cant bit of the mantissa. A 1 is always assumed to exist immediately to the left of the mantissa, although it is not represented. Thus the number is represented as

(<sign> 1.<mantissa> * 2) ^ (exponent-127)

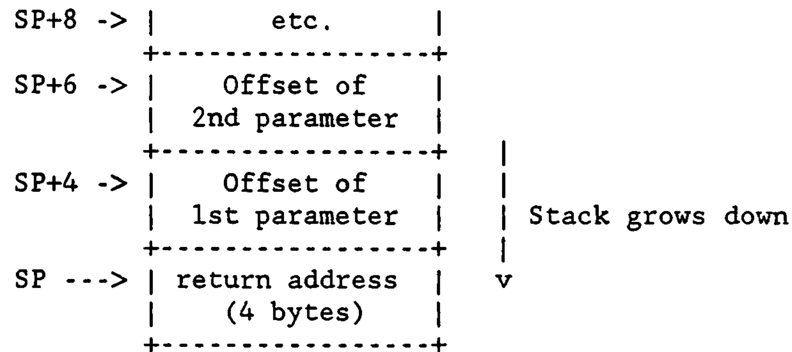
3) Double Precision Number - eight byte binary floating point quantity. It is represented exactly the same as a single precision number, except that the mantissa is made up of 41 bits (7 bytes less the sign bit).

4) String - BASIC will pass a pointer to a 'string descriptor' which is a three byte data structure. The first byte of the string descriptor contains the length of the string. The second and third bytes contain the address where the actual ASCII string is located. The assembly language subroutine is allowed to modify the string, but must not change the string descriptor.

5) Array - arrays are made up of sequential elements of the array type. For example, an integer array containing twenty elements is represented as twenty sequential integers in memory.

### 11.1.3 Passing Parameters

BASIC passes all subroutine parameters by reference. The offset of each parameter's address is pushed onto the stack in the same order that the parameters are listed in the procedure call. Upon entry to the subroutine, the stack will be arranged as follows:


```
SP+8 -> |       etc.      |
        +-----------------+
SP+6 -> |    Offset of    |
        |  2nd parameter  |
        +-----------------+    |
SP+4 -> |    Offset of    |    |
        |  1st parameter  |    | Stack grows down
        +-----------------+    |
SP ---> | return address  |    v
        |    (4 bytes)    |
        +-----------------+
```

The parameters can then be referenced by using the BP register to get their address off of the stack. The following example shows how to do this.

### 11.1.4 Example

This example shows how to call an assembly language routine from BASIC. The assembly language routine performs modulo arithmetic on two integers, returning the remainder that results when the first integer is divided by the second. The assembly language module consists of two procedures. The first procedure loads the module into memory, and puts the entry address of the second procedure into interrupt vector table entry 128. The second procedure is called from BASIC, and performs the modulo function. The BASIC program peeks the Interrupt Vector Table to get the entry address of the modulo function, and the performs the call with some sample data.

ASSEMBLY LANGUAGE MODULE

```
name    modulo

code    segment public 'code'
assume  cs:code, ds:code

        org     100h        ; necessary for .COM program

; This procedure loads the module into memory and sets up
; interrupt vector table entry 128.

loader  proc    near

        IVT_seg    equ 0        ; Interrupt Vector Table
        Int128_off equ 512      ;   entry 128 is at 0:512

        push    cs
```

```
        pop     ds              ; DS = CS in .COM program

; set up Interrupt vector table entry 128 to
; point to the Modulo arithmetic function.

        mov     ax, IVT_seg
        mov     es, ax
        mov     bx, Int128_off
        mov     ax, offset modulo
        mov     es:[bx], ax
        mov     es:[bx+2], cs

; Terminate and remain resident. Dx = last byte of program + 1.

        mov     dx, offset mod_ends
        inc     dx
        int     27h

loader  endp

modulo  proc    far             ; must be declared far

; This module is called from BASIC with 3 parameters.
; It divides the first parameter by the second and
; returns the remainder in the third.

        mov     bp, sp          ; BP used to get parameters
        mov     bx, [bp+8]      ; BX = pointer to dividend
        mov     ax, [bx]        ; AX = value of dividend
        mov     bx, [bp+6]      ; BX = pointer to divisor
        mov     cx, [bx]        ; CX = value of divisor
        mov     dx, 0           ; DX:AX = dividend
        idiv    cx              ; AX = quotient, DX = remainder
        mov     bx, [bp+4]      ; BX = address of result
        mov     [bx], dx        ; return result to BASIC
        ret     6

mod_ends:
modulo  endp
code    ends
        end
```

BASIC PROGRAM

```
5 ' Get address of assembly language MODULO routine from
6 ' Interrupt Vector Table entry 128, which is located at
7 ' memory address 0:512.
8 '
```

```
10 DEF SEG = 0
20 LOWOFF = PEEK(512)
30 HIOFF  = PEEK(513)
40 LOWSEG = PEEK(514)
50 HISEG  = PEEK(515)
60 SEG = (256*HISEG)+LOWSEG
70 MODULO = (256*HIOFF)+LOWOFF
80 DEF SEG = SEG
85 '
90 ' call the MODULO routine
95 '
100 A% = 140
110 B% = 11
120 REMAINDER% = 0
130 CALL MODULO(A%,B%,REMAINDER%)
140 PRINT A%;"modulo";B%;"is";REMAINDER%
150 END
```

This example illustrates one other important point. All parameters must be variables, and they must be initialized before calling the assembly language subroutine. After assembling and linking the assembly language module, it is necessary to convert the resulting .EXE file into a .COM file in order for the terminate and remain resident function to work correctly. An easy way to do this is with the Microsoft debugger, using the following sequence of instructions:

```
debug asm_module.exe
nasm_module.com
w
q
```

Then the assembly language module can be loaded by running the .COM program. After it has loaded, you can run your BASIC program which calls the assembly language module.

## 11.2 Interfacing Compiled Basic with Assembly Language

Occasionally, you may wish to write certain subroutines in assembly language instead of Compiled BASIC because of speed, size, or other constraints. This section explains how to combine compiled BASIC programs with assembly language modules so that parameters are passed correctly between the Compiled BASIC program and the assembly language subroutine.

To understand this section fully, the reader should have a basic knowledge of the following:

1) The MS-BASIC Compiler

2) The Microsoft assembler

3) The Microsoft linker

4) The register structure of the 8086/8088

### 11.2.1 Assembly Language Subroutines

Assembly language subroutines can be invoked from compiled BASIC using either the CALL statement or the CALLS statement. The CALL statement pushes the offset addresses of any parameters on the stack before it transfers execution to the subroutine, while the CALLS statement pushes both the segment and offset addresses of any parameters on the stack. The example later in this discussion will fully illustrate this difference.

The assembly language subroutine must follow some simple rules in order to work correctly.

1) It must be declared FAR.

2) It must be declared PUBLIC.

3) Segment registers DS and ES must be restored to their entry values before returning to Compiled BASIC.

4) The general purpose registers (AX, BX, CX, DX, SI, DI, and BP) can have any value when returning to Compiled BASIC.

5) The assembly language routine MUST NOT change the length of any Compiled BASIC strings.

6) The assembly language routine must perform a RET <n> (where n = 2 times the number of parameters) to restore the stack pointer to its proper value.

7) Values can be returned to Compiled BASIC by passing a parameter that the result will be returned in.

### 11.2.2 Compiled Basic Data Types

In order to manipulate data passed to an assembly language subroutine, it is necessary to understand how the various data types are represented in memory. When a subroutine is called, Compiled BASIC will pass the address of one of the following data

representations.

1) Integer - two byte two's complement number

2) Single Precision Number - four byte binary floating point quantity. The most significant byte contains the value of the exponent minus 127. The remaining three bytes contain the mantissa. The most significant byte of the mantissa contains the sign bit, followed by the seven highest bits of the mantissa. A positive number is represented with a 0 as the sign bit, a negative number with a 1 as the sign bit. The binary point is to the left of the most significant bit of the mantissa. A 1 is always assumed to exist immediately to the left of the mantissa, although it is not represented. Thus the number is represented as

   (<sign> 1.<mantissa> * 2) ^ (exponent-127)

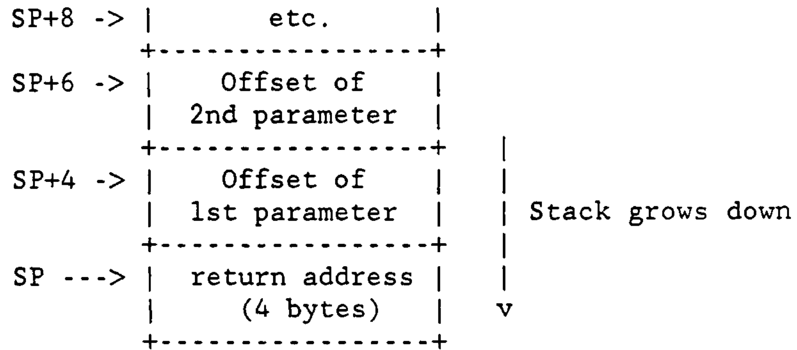
3) Double Precision Number - eight byte binary floating point quantity. It is represented exactly the same as a single precision number, except that the mantissa is made up of 41 bits (7 bytes less the sign bit).

4) String - Compiled BASIC will pass a pointer to a 'string descriptor' which is a four byte data structure. The first two bytes of the string descriptor contain the length of the string. The last two bytes contain the address where the actual ASCII string is located. The assembly language subroutine is allowed to modify the string, but must not change the string descriptor.

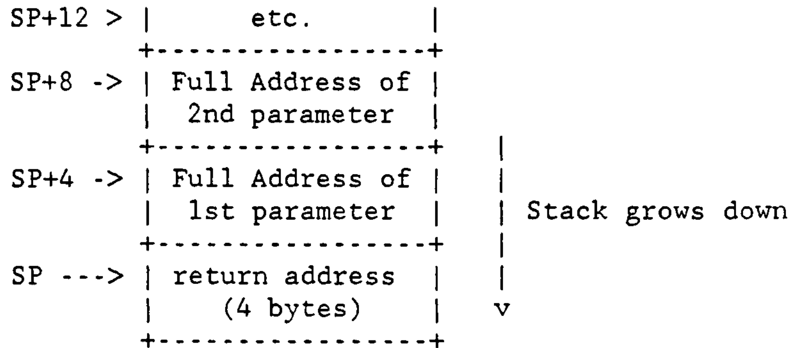
5) Array - arrays are made up of sequential elements of the array type. For example, an integer array containing twenty elements is represented as twenty sequential integers in memory.

### 11.2.3 Passing Parameters

Compiled BASIC passes all subroutine parameters by reference. In a CALL statement, the offset of each parameter's address is pushed onto the stack in the same order that the parameters are listed in the procedure call. It is important to note that all parameters to the assembly language subroutine must be variables. Upon entry to the subroutine, the stack will be arranged as follows:




If a CALLS statement is used instead, then the stack will look like this when a subroutine is entered:




The parameters can then be referenced by using the BP register to get their address off of the stack. The following example shows how to do this.

### 11.2.4 Example

This example shows how to link an assembly language subroutine with a Compiled BASIC program. The assembly language routine performs modulo arithmetic on two integers, returning the remainder that results when the first integer is divided by the second. The example program is shown twice, once using a CALL statement and once using a CALLS statement.

1) Compiled BASIC program with CALL statement

```
10 '
20 ' call the MODULO routine
30 '
40 A% = 140
50 B% = 11
60 REMAINDER% = 0
```

```
70 CALL MODULO(A%,B%,REMAINDER%)
80 PRINT A%;"modulo";B%;"is";REMAINDER%
90 END
```

Assembly language module for use with CALL statement

```
name     modulo

code     segment public 'code'
assume   cs:code, ds:code

public   modulo

modulo   proc    far

; This module is called from Compiled BASIC with 3 parameters,
; using the CALL statement. It divides the first parameter by
; the second and returns the remainder in the third.

         mov     bp, sp              ; BP used to get parameters
         mov     bx, [bp+8]          ; BX = pointer to dividend
         mov     ax, [bx]            ; AX = value of dividend
         mov     bx, [bp+6]          ; BX = pointer to divisor
         mov     cx, [bx]            ; CX = value of divisor
         mov     dx, 0               ; DX:AX = dividend
         idiv    cx                  ; AX = quotient, DX = remainder
         mov     bx, [bp+4]          ; BX = address of result
         mov     [bx], dx            ; return result to BASIC
         ret     6

modulo   endp
code     ends
         end
```

2) Compiled BASIC program with CALLS statement

```
10 '
20 ' call the MODULO routine
30 '
40 A% = 140
50 B% = 11
60 REMAINDER% = 0
70 CALLS MODULO(A%,B%,REMAINDER%)
80 PRINT A%;"modulo";B%;"is";REMAINDER%
90 END
```

Assembly language module for use with CALLS statement

```
name    modulo

code    segment public 'code'
assume  cs:code, ds:code

public  modulo

modulo  proc    far

; This module is called from Compiled BASIC with 3 parameters, using
; the CALLS statement. It divides the first parameter by the second
; and returns the remainder in the third.

        mov     bp, sp                  ; BP used to get parameters
        les     bx, dword ptr [bp+12]   ; ES:BX = pointer to dividend
        mov     ax, es:[bx]             ; AX = value of dividend
        les     bx, dword ptr [bp+8]    ; ES:BX = pointer to divisor
        mov     cx, es:[bx]             ; CX = value of divisor
        mov     dx, 0                   ; DX:AX = dividend
        idiv    cx                      ; AX = quotient, DX = remainder
        les     bx, dword ptr [bp+4]    ; ES:BX = address of result
        mov     es:[bx], dx             ; return result to BASIC
        ret     6

modulo  endp
code    ends
        end
```

After compiling and assembling the various modules, use the Microsoft linker to create the executable program. The compiled BASIC object modules should be listed before the names of the assembly language object modules. After your modules have been linked, your program is ready to run.

## 11.3 Calling Assembly Language Subroutines from GWBasic

Assembly language subroutines can be invoked from GWBASIC using the CALL statement. The CALL statement pushes the offset addresses of any parameters on the stack before it transfers execution to the subroutine.

The assembly language subroutine must follow some simple rules in order to work correctly.

1) It must be declared FAR.

2) Segment registers DS and ES must be restored to their entry values before returning to Compiled BASIC.

3) The general purpose registers (AX, BX, CX, DX, SI, DI, and BP) can have any value when returning to GWBASIC.

4) The assembly language routine MUST NOT change the length of any GWBASIC strings.

5) The assembly language routine must perform a RET <n> (where n = 2 times the number of parameters) to restore the stack pointer to its proper value.

6) Values can be returned to GWBASIC by passing a parameter that the result will be returned in.

**11.3.1 GWBasic Data Types**

In order to manipulate data passed to an assembly language subroutine, it is necessary to understand how the various data types are represented in memory. When a subroutine is called, GWBASIC will pass the address of one of the following data representations.

1) Integer - two byte two's complement number

2) Single Precision Number - four byte binary floating point quantity. The most significant byte contains the value of the exponent minus 127. The remaining three bytes contain the mantissa. The most significant byte of the mantissa contains the sign bit, followed by the seven highest bits of the mantissa. A positive number is represented with a 0 as the sign bit, a negative number with a 1 as the sign bit. The binary point is to the left of the most significant bit of the mantissa. A 1 is always assumed to exist immediately to the left of the mantissa, although it is not represented. Thus the number is represented as

   (<sign> 1.<mantissa> * 2) ^ (exponent-127)

3) Double Precision Number - eight byte binary floating point quantity. It is represented exactly the same as a single precision number, except that the mantissa is made up of 41 bits (7 bytes less the sign bit).

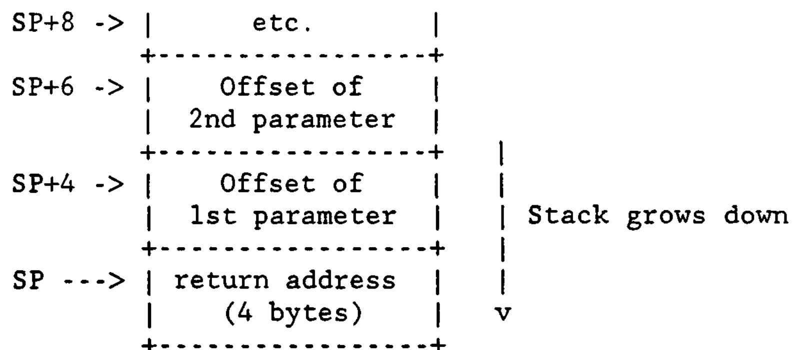
4) String - GWBASIC will pass the offset address of a

'string descriptor' which is a three byte data structure. The first byte of the string descriptor contains the length of the string. The last two bytes contain the address where the actual ASCII string is located. The assembly language subroutine is allowed to modify the string, but it must not change the string descriptor.

5) Array - arrays are made up of sequential elements of the array type. For example, an integer array containing twenty elements is represented as twenty sequential integers in memory.

### 11.3.2 Passing Parameters

GWBASIC passes all subroutine parameters by reference. In a CALL statement, the offset of each parameter's address is pushed onto the stack in the same order that the parameters are listed in the procedure call. It is important to note that all parameters to the assembly language subroutine must be variables. Upon entry to the subroutine, the stack will be arranged as follows:




The parameters can then be referenced by using the BP register to get their address off of the stack. The following example shows how to do this.

### 11.3.3 Example

This example shows how to load an assembly language subroutine from a GWBASIC program. The assembly language routine performs modulo arithmetic on two integers, returning the remainder that results when the first integer is divided by the second. In this example, the assembly language module is loaded at address 1664:0 Hex, but this address will be different for different applications. The method of determining this

address is explained after the example.

GWBASIC program

```
10 '
20 ' load the MODULO routine
30 '
40 DEF SEG = &H1664
50 BLOAD "MODULO",0
60 MODULO = 0
70 '
80 ' call the MODULO routine with some sample data
90 '
100 A% = 140
110 B% = 11
120 REMAINDER% = 0
130 CALL MODULO(A%,B%,REMAINDER%)
140 PRINT A%;"modulo";B%;"is";REMAINDER%
150 END
```

Assembly language module for use with CALL statement

```
name    modulo

code    segment public 'code'
assume  cs:code, ds:code

modulo  proc    far

; This module is called from GWBASIC with 3 parameters,
; using the CALL statement. It divides the first parameter by
; the second and returns the remainder in the third.

        mov     bp, sp          ; BP used to get parameters
        mov     bx, [bp+8]      ; BX = pointer to dividend
        mov     ax, [bx]        ; AX = value of dividend
        mov     bx, [bp+6]      ; BX = pointer to divisor
        mov     cx, [bx]        ; CX = value of divisor
        mov     dx, 0           ; DX:AX = dividend
        idiv    cx              ; AX = quotient, DX = remainder
        mov     bx, [bp+4]      ; BX = address of result
        mov     [bx], dx        ; return result to BASIC
        ret     6

modulo  endp
code    ends
        end
```

## LOADING THE ASSEMBLY LANGUAGE MODULE

In order to call the assembly language module, it is necessary to know the address that it is located at. The BLOAD statement allows you to load the module at any physical address desired. However, to use the BLOAD statement to load a module, you must first create the disk file containing the module with the Microsoft linker and debugger and the BSAVE statement, as follows:

1) After assembling your module to create the object file, use the linker to create the .EXE file. Use the /HIGH switch when linking so that the module will load in high address memory.

2) Use the debugger to load the .EXE file produced in step 1.

3) Display the register values (with the R command) to determine where the subroutine was loaded. Write down the values contained in the CS:IP register pair and the CX register. The CS:IP register pair contains the starting address of the subroutine and the CX register contains its length.

4) Load and execute GWBASIC from DEBUG with this sequence of commands:

```
NGWBASIC
L
N
G
```

   Note that your assembly language module is still loaded in high address memory.

5) Set the segment value in GWBASIC with a DEF SEG statement:

```
DEF SEG = <value in CS register>
```

6) Save the module with a BSAVE statement:

```
BSAVE "module_name", <value in IP reg.>, <value in CX reg.>
```

The assembly language subroutine is now ready to be called from your GWBASIC program. The following statements are required in your GWBASIC program before the subroutine can be called:

```
DEF SEG = <value in CS register>
BLOAD "module_name", <value in IP register>
```

SUBROUTINE = <value in IP register>

The subroutine can then be called with statements of the form:

CALL SUBROUTINE(PARAMETER1, PARAMETER2, ...)

## 11.4 Interfacing COBOL with Assembly Language

Occasionally, you may wish to write certain subroutines in assembly language instead of Cobol because of speed, size, or other constraints. This section explains how to combine Cobol programs with assembly language modules so that parameters are passed correctly between the Cobol program and the assembly language subroutine.

To understand this section fully, the reader should have a basic knowledge of the following:

1) The Cobol Compiler

2) The Microsoft MACRO Assembler

3) The Microsoft linker

4) The register structure of the 8086/8088

### 11.4.1 Calling Assembly Language Subroutines

Assembly language subroutines can be invoked from Cobol using the CALL statement with the assembly language module name as a literal. Parameters can be passed to the assembly language routine with the addition of the USING clause. The CALL statement pushes the offset addresses of any parameters on the stack before it transfers execution to the subroutine. The examples later in this discussion will fully illustrate the calling procedure.

The assembly language subroutine must follow some simple rules in order to work correctly.

1) It must be declared FAR.

2) It must be declared PUBLIC.

3) Segment registers DS and ES along with register BP must be restored to their entry values before returning to Cobol.

4) The general purpose registers (AX, BX, CX, DX, SI, and DI) can have any value when returning to Cobol.

5) The assembly language routine must perform a RET <n> (where n = 2 times the number of parameters) to restore the stack pointer to its proper value.

6) Values can be returned to Cobol by passing a parameter to the assembly language subroutine that the result be returned in.

### 11.4.2 COBOL Data Types

In order to manipulate data passed to an assembly language subroutine, it is necessary to understand how the various data types are represented in memory. When a subroutine is called, Cobol will pass the address of one of the following data representations.

1) Computational-0

Also known as a binary item, uses the base 2 system to represent an integer in the range -32768 to 32767. It occupies one 16-bit word, with the leftmost bit reserved for the operational sign.

It should also be noted that Cobol represents all data types, except Index, internally in reverse order. For example, if you have the following Cobol declaration :

```
77      EXAMPLE1        PIC 99 COMP-0 VALUE 50.
```

It would be represented internally, in hex, as:

| low byte | high byte |
|---|---|
| 00 | 32 |

instead of :

| low byte | high byte |
|---|---|
| 32 | 00 |

2) Computational-3

Also known as an internal decimal item, is stored internally is binary-coded decimal format. A Computational-3 data item, defined by n 9's in its PICTURE, occupies 1/2 of (n + 2)

bytes of memory. All bytes except the rightmost contain a pair of digits, and each digit is represented by the binary equivalent of a valid digit value from 0 to 9. The item's low order digit and the operational sign are found in the rightmost byte. The compiler considers a Computational-3 item to have an arithmetic sign, even if the original PICTURE lacked an S character. The operational sign, contained in the rightmost byte, is hexadecimal F for positive numbers and hexadecimal D for negative numbers.

3) External Decimal

An external data item is an item in which one byte is employed to represent one numeric digit. An unsigned external data item is reprepresented internally as its ASCII equivalent. A signed external data item is represented internally as its ASCII equivalent, EXCEPT the low order byte on negative items, which have the following rules :

| If the low order digit is : | The value internally is : |
|---|---|
| 0 | 7D hex |
| 1 | 4A hex |
| 2 | 4B hex |
| 3 | 4C hex |
| 4 | 4D hex |
| 5 | 4E hex |
| 6 | 4F hex |
| 7 | 50 hex |
| 8 | 51 hex |
| 9 | 52 hex |

For example, if you have the following Cobol declaration :

```
77       EXAMPLE2         PIC S999 VALUE -121.
```

The value internally would be :

31 32 4A

4) Alphanumeric

An alphanumeric data item is represented

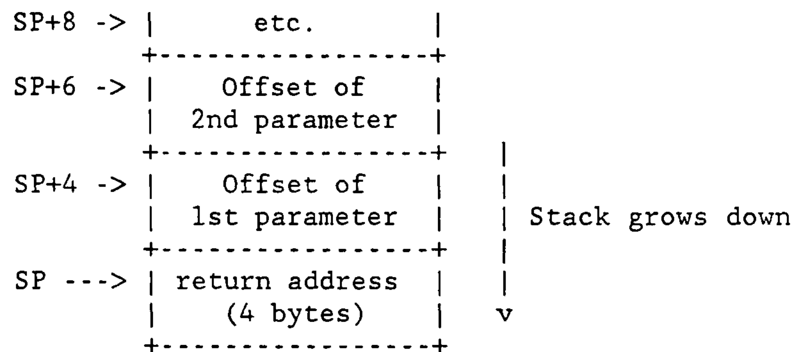
internally as its ASCII equivalent.

5) Alphabetic
An alphabetic data item is represented internally as its ASCII equivalent.

6) Index
An index data item is represented internally as a binary word.

### 11.4.3 Passing Parameters

Cobol passes all subroutine parameters by reference. In a CALL statement, the offset of each parameter's address is pushed onto the stack in the same order that the parameters are listed in the USING clause. Upon entry to the assembly language subroutine, the stack will be arranged as follows:




The parameters can then be referenced by using the BP register to get their address off of the stack. The following example shows how to do this.

### 11.4.4 Example

This example shows how to link an assembly language subroutine with a Cobol program. The assembly language routine performs modulo arithmetic on two Computational-0 variables, returning the remainder that results when the first variable is divided by the second.

Cobol program with CALL/USING statement

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE.
ENVIRONMENT DIVISION.
DATA DIVISION.
```

```
WORKING-STORAGE SECTION.
77  PARM1    PIC  99  COMP-0 VALUE 50.
77  PARM2    PIC  99  COMP-0 VALUE 11.
77  PARM3    PIC  99  COMP-0 VALUE 0.
77  PAR1     PIC  99.
77  PAR2     PIC  99.
77  PAR3     PIC  99.
PROCEDURE DIVISION.
MAIN.
    CALL "MODULO" USING PARM1, PARM2, PARM3
    MOVE PARM1 TO PAR1.
    MOVE PARM2 TO PAR2.
    MOVE PARM3 TO PAR3.
    DISPLAY PAR1 "  MOD  " PAR2 "  =  " PAR3.
    STOP "Hit <cr> to return to system"
```

Assembly language module for use with CALL/USING statement

```
name    modulo

code    segment public 'code'
assume  cs:code, ds:code

public  modulo

; This module is called from Cobol with 3 parameters,
; using the CALL statement. It divides the first parameter by
; the second and returns the remainder in the third.
;
; Stack structure after saving BP
;
  parm1 equ     10                   ; pointer to dividend
  parm2 equ      8                   ; pointer to divisor
  parm3 equ      6                   ; pointer to return variable
; off   equ      4                   ; return offset
; seg   equ      2                   ; return segment
; bp    equ      0                   ; save BP
modulo  proc    far

        push    bp                   ; Save BP
        mov     bp, sp               ; BP used to get parameters
        mov     bx, [bp+parm1]       ; BX = pointer to dividend
        mov     ax, [bx]             ; AX = value of dividend
        mov     bx, [bp+parm2]       ; BX = pointer to divisor
        mov     cx, [bx]             ; CX = value of divisor
        mov     dx, 0                ; DX:AX = dividend
        idiv    cx                   ; AX = quotient, DX = remainder
        mov     bx, [bp+parm3]       ; BX = address of result
```

```
        mov     [bx], dx                ; return result to COBOL
        pop     bp                      ; Restore BP
        ret     6

modulo  endp
code    ends
        end
```

After compiling and assembling the various modules, use the Microsoft linker to create the executable program. The Cobol object modules should be listed before the names of the assembly language object modules. After your modules have been linked, your program is ready to run.

## 11.5 Interfacing Pascal Programs with Assembly Language Subroutines

Sometimes it may be desirable to write certain procedures or functions in assembly language instead of Pascal, because of speed, size, or other constraints. This section explains how to successfully link Pascal programs with assembly language subroutines so that parameters and function return values are passed correctly between the Pascal and assembly code modules. The terms subroutine and procedure used interchangably to mean either a Pascal procedure or function, while the term function applies specifically to functions. In order to best understand this section, knowledge of the following is desirable:

1). The Pascal compiler

2). The assembler

3). The linker

4). The register structure of the 8086/8088

### 11.5.1 Calling External Subroutines

To call an assembly language subroutine from a Pascal program, it is necessary to declare the assembly language subroutine as an external procedure or function. The format of an external procedure declaration in Pascal is exactly like that of a standard Pascal procedure declaration with the addition of the external directive and no procedure body. See the section on directives in the 'Reference Manual for MS-Pascal' for detailed information.

The assembly language subroutine must have the far attribute in its 'proc' statement, since Pascal assumes that all external procedures are far. Pascal also requires assembly language routines to have identical class names to Pascal routine class names. The acceptable class names can be found in the file ENTX6L.ASM, found on the Pascal compiler diskette. For example, code of an assembly language routine should have the class name 'CODE' and data used by an assembly language routine should have the class name 'DATA'. In addition, the name of the subroutine must be declared public in the assembler code. This must be the same name that is declared as an external procedure in Pascal. The two examples later in this chapter illustrate the relationship between the external declaration in Pascal and the public declaration in assembler.

The user written subroutine can modify the AX, BX, CX, DX, DI, SI, and ES registers. The SP, BP and DS registers can also be modified, but their values must be restored before returning to Pascal. The SS register should NEVER be modified. The user should also pop all parameters off of the stack by using a ret N statement, where N equals the number of bytes on the stack used by the parameters.

### 11.5.2 Passing Parameters

When a procedure is called in Pascal, either the address or the value of any parameters are passed to the procedure on the stack. The address of a parameter is pushed on the stack when the formal parameter is declared as a VAR, VARS, CONST, or CONSTS type. The value of a parameter is passed on the stack if the formal parameter in the procedure declaration does not have one of these types. The following example code should clarify this distinction.

------------------------- Figure 1 -------------------------

```
PROGRAM Sample (INPUT,OUTPUT);

VAR
  alpha, delta : INTEGER;

PROCEDURE Gamma(VARS x:INTEGER; z:INTEGER);
  BEGIN
    { body of procedure here }
  END;

BEGIN  { main program body }
  alpha := 10;
```

```
    delta := 21;
    Gamma(alpha, delta);
  END.
```

----------------------- End of Figure 1 -----------------------

The declaration for procedure gamma defines gamma as having two parameters: x and y. The first parameter is passed by address (this is also known as 'passing by reference'), while the second parameter is passed by value. When gamma is actually called, the address of alpha is pushed onto the stack, while delta is passed by having its value (21 in this case) pushed onto the stack.

In the 8086, addresses can be one of two types: near or far. A near address consists of the sixteen bit offset address in the current segment, while a far address consists of a full twenty bit address made up of specified segment and offset values. In Pascal, the user has the ability to specify which form of addressing to use for parameters (in relation to the data segment) when they are passed by reference. In the declaration of a procedure or function, near addresses are specified by declaring formal parameters as VAR or CONST, while far addresses are specified by declaring parameters as VARS or CONSTS. In the above example, the address of alpha is passed to the procedure by pushing the segment address (i.e. the value of the DS register) of alpha on the stack, followed by its offset address. The segment address of a parameter is the value contained in the DS register. If the formal parameter x in the procedure declaration had been declared as VAR instead of VARS, then only the offset address of alpha would have been pushed onto the stack. Note that the declared type of the variable does not affect the way that the parameter address is passed.

When a parameter is passed by value, the actual value of the variable at that time is pushed onto the stack. For variables of large size (such as large arrays, records, etc.), this causes the stack to grow quickly and potentially overflow. It is usually preferable to pass structured variables by reference to prevent this occurrence.

### 11.5.3 Pascal Data Types

1) Byte - simple 1 byte unsigned value.

2) Char - 1 byte ASCII character representation.

3) Boolean - 1 byte value. FALSE is represented with a 0 in the low order bit (bit 0). TRUE is represented with a 1

in bit 0. Bits 1-7 should be 0.

4) Word - Normally, a 2 byte unsigned value. However, Word subranges in the range 0..255 are represented by a one byte unsigned value.

5) Integer - Normally, a 2 byte two's complement number. Subranges in the range -128..127 are represented using one byte only.

6) Integer4 - 4 byte two's complement number.

7) Real4 - 4 byte IEEE standard real format. The most significant bit is the sign bit, followed by an 8 bit exponent with a bias of 127. This is followed by a 23 bit mantissa. The mantissa has a 'hidden' most significant bit that is always a 1, so the mantissa is actually a 24 bit quantity that represents a number greater than or equal to 1.0 but is less than 2.0.

8) Real8 - 8 byte IEEE format. The sign bit is followed by a 11 bit exponent with bias of 1023, followed by a 52 bit mantissa. As in Real4, the mantissa has a 'hidden' most significant bit that is always a 1.

9) arrays and records - the internal format of arrays and records is composed of the internal forms of the components, in the same order as in the array or record declaration.

10) super arrays - like arrays, super arrays are composed of the form of its declared component type. In a procedure declaration, a super array type can be defined as an address parameter. When the procedure is called, the actual parameter is substituted for the formal paramater and the size of the super array is pushed onto the stack before its address is pushed. This allows the procedure to be more general, as it can operate on arrays of different lengths. If the formal parameter is a dimensioned super array type, then the length of the super array is not passed on the stack. It is important to note that an undimensioned super array type cannot be passed by value and cannot be a function return type.

11) string - the string type is a predeclared super array, and as such has the same restrictions as a super array. The string itself is just a sequential array of type Char, with each element represented by its ASCII value.

12) lstring - the lstring type is also a predeclared super array. It is exactly like the string type, except that the first byte of the array contains the length of the string. This allows lstrings to have variable lengths but limits their maximum length to 255 characters.

13) Enumerated types - If there are 256 or fewer values in the enumerated type, 1 byte is required. If the enumerated type has more than 256 possible values, two bytes are required. The values are numbered from 0 to n-1, in the order declared (i.e. the value returned by the ord function on the type).

14) Address types - the ADR type is represented by a 2 byte value containing the offset address of the value. The ADS type is a 4 byte value containing both the segment and offset addresses of the value.

### 11.5.4 Returned Values

In Pascal, the returned value of a function is passed back to the calling module in specific registers. For small data types the actual value is returned, while for large data types the address of the result is returned. It is necessary for user written functions to follow the Pascal conventions.

Return values declared as one of the simple types Boolean, Byte, Char, Integer2, Word, or Integer4, or as one of the address types Adr or Ads, or an Enumerated type are returned by value in specified registers. The value of single byte types (Boolean, Byte, or Char) should be returned in the AL register. The value of single word types (Adr, Integer2, or Word) should be returned in the AX register. An enumerated type is returned in either the AL or AX register, depending on whether it has more or less than 256 declared elements. The value of double word types (Ads and Integer4) should be returned in the DX and AX registers, with the most significant word in DX.

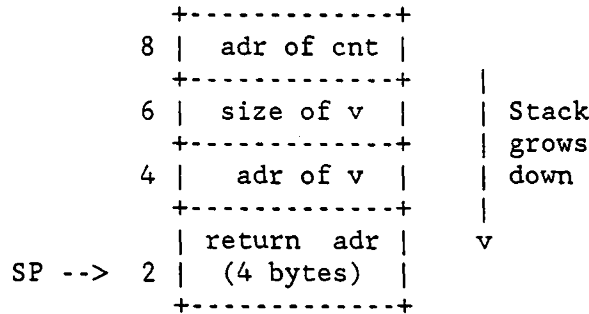
Function return values of any other type (such as Real, Record, and Super Array based types) are returned by address. This address should be returned in the AX register. When a function with one of these types is called, a temporary variable is allocated by Pascal. The near address of this temporary variable is pushed onto the stack just before the return address (i.e. after all the parameters have been pushed). The user function should put the value to be returned in this variable, and return the address in the AX register. The second example below shows how this mechanism works.

### 11.5.5 Example 1 - Sum function

This example (Figure 2) shows a user written assembly language routine that performs a sum function. The routine requires two input parameters - an integer super array of type vector and an integer value that contains the number of valid elements in the array. The function adds all of the valid elements in the array (which should be in array positions 1 to count) and returns the total of these values as an integer in the AX register.

It is important to note that the function has been declared external in the Pascal program and public in the assembler subroutine. It should also be noted that the return statement in Pascal pops six bytes off of the stack, even though there are only two parameters. This is due to the fact that one of the formal parameters is an undimensioned super array type, so that when the function is called, the size of the actual super array parameter is pushed onto the stack just before its address. The state of the stack just after the call to sum is shown here:

High addresses:




Low addresses:

In the assembly language function below, the value of the BP register is pushed on the stack, and then BP is used to access the values on the stack. Note that the offsets from BP are two greater than the offsets in the above diagram, because the value of BP has been saved on the stack, increasing the stack size by two bytes.

-------------------------- Figure 2 --------------------------

PASCAL PROGRAM:

```
PROGRAM Sumtest (input, output);

TYPE
  VECTOR = SUPER ARRAY [1..*] OF INTEGER;
```

```
VAR
  scores : VECTOR(10);
  total  : INTEGER;
  count  : INTEGER;

FUNCTION Sum (cnt:INTEGER; VAR v:VECTOR) : INTEGER; EXTERNAL;
  (* sum must be declared as an external function *)

BEGIN

  (* User code here sets values for count and scores array *)

  total := Sum(count, scores);
  WRITELN('The total is: ',total);
END.
```

ASSEMBLY LANGUAGE FUNCTION

```
name sum

; This subroutine is called from PASCAL.
;
; This subroutine requires two parameters:
; 1 - an integer containing the number of elements to sum
; 2 - a super array of integers containing the values to sum
;     (in elements 1..n)

cgroup  group code
dgroup  group data

assume  cs:cgroup, ds:dgroup
code    segment public 'code'   ; segment uses class name
                                ;  of 'code'

public  sum                     ; sum must be 'public'

sum     proc    far             ; sum must be 'far'
        push    bp
        mov     bp, sp
        mov     ax, 0           ; initialize sum
        mov     dx, ax          ; initialize array index
        mov     bx, [bp+6]      ; bx <- adr of array
        mov     cx, [bp+10]     ; cx <- length of array
sumloop:
        mov     di, dx          ; di <- index into array
        shl     di, 1           ; convert to word index
        add     ax, [bx+di]     ; add in next value of array
        inc     dx              ; increment array index
```

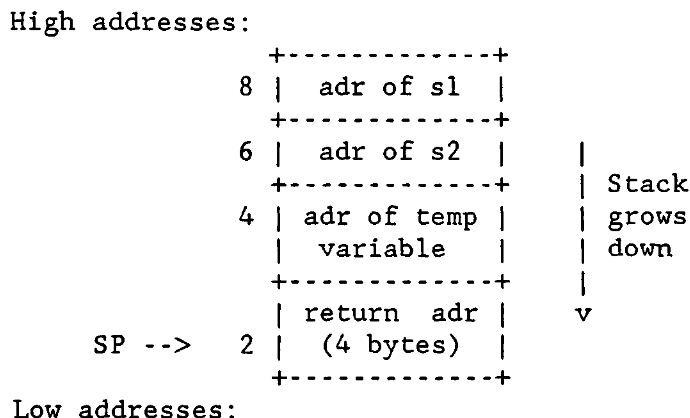
```
            loop    sumloop

            pop     bp
            ret     6               ; return - sum in ax
    sum     endp

    code    ends
            end
```

--------------------- End of Figure 2 -----------------------

### 11.5.6 Example 2: String concatenation function

This example illustrates how to return a value of a structured type to a Pascal program. The Pascal program passes two strings to the subroutine, which returns the string that results from cancatenating the second string to the first.

Note that the function requires two lstrings (declared as type shortstring) as parameters, and returns a third lstring (of type longstring) to the calling program. Since the returned value is a structured type, Pascal passes the address of a temporary variable of this type on the stack immediately before calling the user function. The assembly language routine uses this temporary variable to build the concatenated string, and then returns this address to the caller in the AX register. Just after the call, the stack is structured as follows:




As in the first example, the BP register is used to access parameters passed on the stack. The temporary variable passed on the stack is used to build the new string, and then the address is returned in the AX register to the Pascal program.

It is important that the programmer of the assembler func-

tion understands the data structures of any types being used by his routine. In this example, the structure of the lstring type needed to be known in order for the concatenated string to be correctly built and correctly interpreted by the Pascal program when it is returned.

------------------------- Figure 3 --------------------------

```
PROGRAM Myname (INPUT, OUTPUT);

TYPE
  SHORTSTRING = LSTRING(15);
  LONGSTRING  = LSTRING(30);

VAR
  first_name : SHORTSTRING;
  last_name  : SHORTSTRING;
  full_name  : LONGSTRING;

FUNCTION Concat(VAR s1,s2:SHORTSTRING) : LONGSTRING; EXTERNAL;
  (* Concat must be external *)

BEGIN
  first_name := 'Mortimer ';
  last_name  := 'Freeblekoff';
  full_name  := concat(first_name, last_name);
  writeln('My first name is ',first_name);
  writeln('My last name is ',last_name);
  writeln('My full name is ',full_name);
END.
```

ASSEMBLY LANGUAGE CONCATENATION ROUTINE:

```
name    concat

cgroup group code
assume cs:cgroup

code    segment public 'code'    ; segment uses class name
                                 ;  of 'code'

public  concat                   ; concat must be 'public'

concat  proc    far              ' concat must be 'far'
        push    bp
        mov     bp, sp
```

```
                push    ds
                pop     es                  ; set up ES for string moves
                cld

                mov     di, [bp+6]          ; di <- address of result string
                mov     bx, di
                inc     di                  ; advance to string field
                mov     si,[bp+10]          ; si <- address of 1st string
                mov     cl, [si]            ; cl <- length of 1st string
                mov     al, cl
                mov     ch, 0
                inc     si
                rep     movsb               ; mov 1st string to result

                mov     si, [bp+8]          ; si <- address of 2nd string
                mov     cl, [si]            ; cl <- length of 2nd string
                add     al, cl              ; al <- length of result string
                mov     ch, 0
                inc     si
                rep     movsb               ; add 2nd string to result

                mov     [bx], al            ; mov length to result string
                mov     ax, bx              ; return address of resultant
                pop     bp                  ;   string to Pascal in AX.
                ret     6

        concat  endp
        code    ends
                end
```

---------------------- End of Figure 3 ----------------------

### 11.5.7 Linking

After running the Pascal compiler and the macro assembler, it is necessary to link the object modules produced to create the executable .EXE file. The order of the object modules given to the linker is important - The Pascal program module must be the first module in the list of objects, with the assembly objects last. The order of the assembly objects does not matter, but they must come after all Pascal modules. Thus, the proper link command for the second example above is

```
A>link myname+concat
```

Refer to the "User's Guide for MS-DOS Utility Software" for more information on the Microsoft Linker.

# A. ASCII CODES

## A.1 ASCII CODES USED IN THE SIRIUS 1 COMPUTER

The American Standard Codes for Information Interchange (ASCII) has been defined to allow data communication between computers, their peripherals, and other computers. The other major code standard is the Extended Binary Coded-Decimal Interchange Code (EBCDIC) used on some mainframe computers. The Sirius 1 computer is designed to function in ASCII, but communication software is available that allows the Sirius 1 to receive EBCDIC data and have it translated into ASCII, and vice versa.

The following table contains the 7-ASCII codes and their meanings. It is called 7-ASCII as only 7-bits of the potential 8-bits are used to carry data; the "spare" bit is used in the Sirius 1 computer to support characters not otherwise available in the 7-ASCII set.

An Eight Bit Byte is pictured as follows:

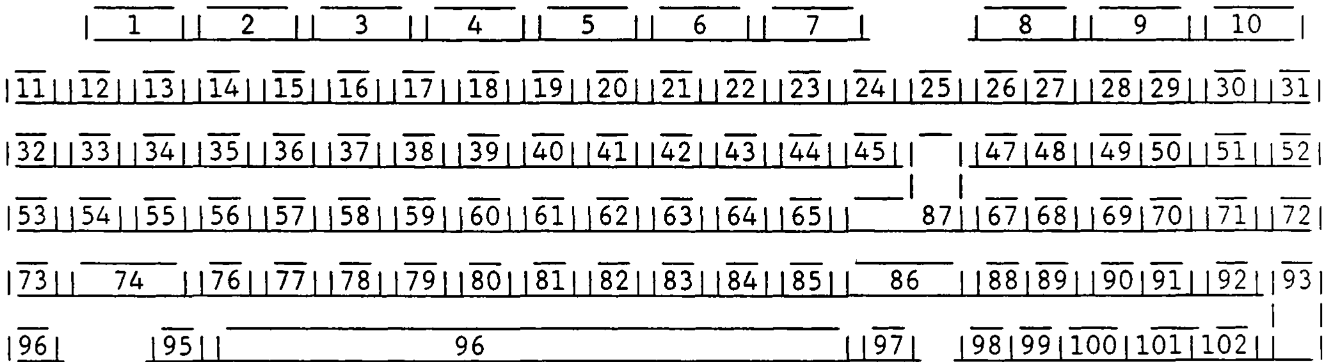
[ 7 ][ 6 ][ 5 ][ 4 ][ 3 ][ 2 ][ 1 ][ 0 ]

the bits are numbered 0 through 7 (which adds up to eight bits), and it is the 8th bit (bit 7 in computer jargon) which is not used in 7-ASCII.

## A.2 ASCII / HEXADECIMAL / DECIMAL Character Set

| ASCII | Hex | Dec | ASCII | Hex | Dec | ASCII | Hex | Dec | ASCII | Hex | Dec |
|---|---|---|---|---|---|---|---|---|---|---|---|
| NUL | 00 | 00 | space | 20 | 32 | @ | 40 | 64 | ` | 60 | 96 |
| SOH | 01 | 01 | ! | 21 | 33 | A | 41 | 65 | a | 61 | 97 |
| STX | 02 | 02 | " | 22 | 34 | B | 42 | 66 | b | 62 | 98 |
| ETX | 03 | 03 | # | 23 | 35 | C | 43 | 67 | c | 63 | 99 |
| EOT | 04 | 04 | $ | 24 | 36 | D | 44 | 68 | d | 64 | 100 |
| ENQ | 05 | 05 | % | 25 | 37 | E | 45 | 69 | e | 65 | 101 |
| ACK | 06 | 06 | & | 26 | 38 | F | 46 | 70 | f | 66 | 102 |
| BEL | 07 | 07 | ' | 27 | 39 | G | 47 | 71 | g | 67 | 103 |
| BS | 08 | 08 | ( | 28 | 40 | H | 48 | 72 | h | 68 | 104 |
| HT | 09 | 09 | ) | 29 | 41 | I | 49 | 73 | i | 69 | 105 |
| LF | 0A | 10 | * | 2A | 42 | J | 4A | 74 | j | 6A | 106 |
| VT | 0B | 11 | + | 2B | 43 | K | 4B | 75 | k | 6B | 107 |
| FF | 0C | 12 | , | 2C | 44 | L | 4C | 76 | l | 6C | 108 |
| CR | 0D | 13 | - | 2D | 45 | M | 4D | 77 | m | 6D | 109 |
| SO | 0E | 14 | . | 2E | 46 | N | 4E | 78 | n | 6E | 110 |
| SI | 0F | 15 | / | 2F | 47 | O | 4F | 79 | o | 6F | 111 |
| DLE | 10 | 16 | 0 | 30 | 48 | P | 50 | 80 | p | 70 | 112 |
| DC1 | 11 | 17 | 1 | 31 | 49 | Q | 51 | 81 | q | 71 | 113 |
| DC2 | 12 | 18 | 2 | 32 | 50 | R | 52 | 82 | r | 72 | 114 |
| DC3 | 13 | 19 | 3 | 33 | 51 | S | 53 | 83 | s | 73 | 115 |
| DC4 | 14 | 20 | 4 | 34 | 52 | T | 54 | 84 | t | 74 | 116 |
| NAK | 15 | 21 | 5 | 35 | 53 | U | 55 | 85 | u | 75 | 117 |
| SYN | 16 | 22 | 6 | 36 | 54 | V | 56 | 86 | v | 76 | 118 |
| ETB | 17 | 23 | 7 | 37 | 55 | W | 57 | 87 | w | 77 | 119 |
| CAN | 18 | 24 | 8 | 38 | 56 | X | 58 | 88 | x | 78 | 120 |
| EM | 19 | 25 | 9 | 39 | 57 | Y | 59 | 89 | y | 79 | 121 |
| SUB | 1A | 26 | : | 3A | 58 | Z | 5A | 90 | z | 7A | 122 |
| ESC | 1B | 27 | ; | 3B | 59 | [ | 5B | 91 | { | 7B | 123 |
| FS | 1C | 28 | < | 3C | 60 | \ | 5C | 92 | \| | 7C | 124 |
| GS | 1D | 29 | = | 3D | 61 | ] | 5D | 93 | } | 7D | 125 |
| RS | 1E | 30 | > | 3E | 62 | ^ | 5E | 94 | ~ | 7E | 126 |
| US | 1F | 31 | ? | 3F | 63 | _ | 5F | 95 | DEL | 7F | 127 |

## B.1 SIRIUS 1 KEYBOARD LAYOUT




**Sirius 1 Keyboard Configuration with Key Switch Positions and Logical Key Numbers**

APPENDIX B B-2

# C. PRINTER OUTPUT

## C.1 Switch Settings for C.Itoh Printers

### F10-40P Daisywheel

| Switch 40 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | C | O | O | C | O | O | C | C | | |
| **Switch 41** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| | O | O | C | O | C | O | O | C | O | O |

### 1550 Dot Matrix

| | Switch 1 | Switch 2 |
|---|---|---|
| Normal | 1 2 3 4 5 6 7 8<br>O C O O O C C O | 1 2 3 4 5 6 7 8<br>O O O O O C O O |
| Graphics | 1 2 3 4 5 6 7 8<br>C C O O O C C O | 1 2 3 4 5 6 7 8<br>O O O O O C O C |

Where O = Open and C = Closed

### Printing Graphics on the 1550 Printer

It is possible to demonstrate the graphics capability of the C.Itoh 1550 printer from the graphics demonstration disc. This is done by typing "P" when the graphics routine required to be printed is being displayed on the VDU, eg. pie charts. This causes the graphics program to produce pixel dump to the printer.

In order to achieve this the following adjustments must be made to the printer and cable

1. The switches on the printer should be set as follows:

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| SW 1 | C | C | O | O | O | C | C | C |
| SW 2 | O | O | O | O | O | C | O | C |

where C = Closed and O = Open

## C.2 Parallel Printer Cables

### Parallel Cable for Paper Tiger

| Sirius | | Printer |
|---|---|---|
| 1 | --------------- | 3 |
| 2 | --------------- | 14 |
| 3 | --------------- | 13 |
| 4 | --------------- | 12 |
| 5 | --------------- | 11 |
| 6 | --------------- | 10 |
| 7 | --------------- | 9 |
| 8 | --------------- | 15 |
| 10 | --------------- | 22 |
| 11 | --------------- | 19 |
| 14 | --------------- | 7 |
| 17 | --------------- | 1 |

### Parallel Cable for Olivetti Bytewriter

| Sirius | | Printer |
|---|---|---|
| 1 | --------------- | 1 |
| 2 | --------------- | 3 |
| 3 | --------------- | 5 |
| 4 | --------------- | 7 |
| 5 | --------------- | 9 |
| 6 | --------------- | 11 |
| 7 | --------------- | 13 |
| 8 | --------------- | 15 |
| 11 | --------------- | 17 |
| 14 | --------------- | 2 |
| 16 | --------------- | 4 |

## Parallel Cable for Crown Ranier Typewriter

| Sirius | | Printer |
|---|---|---|
| | | 1 |
| 1 | --------------- | 2 |
| | | 3 |
| 2 | --------------- | 4 |
| 10 | --------------- | 5 |
| 3 | --------------- | 6 |
| 11 | --------------- | 7 |
| 4 | --------------- | 8 |
| 9 | --------------- | 9 |
| 5 | --------------- | 10 |
| 8 | --------------- | 11 |
| 6 | --------------- | 12 |
| 7 | --------------- | 13 |
| 14 | --------------- | 14 |

## Parallel Cable for Trend Com-200

| Sirius | | Printer |
|---|---|---|
| 1 | --------------- | 8 |
| 2 | --------------- | 10 |
| 3 | --------------- | 11 |
| 4 | --------------- | 12 |
| 5 | --------------- | 13 |
| 6 | --------------- | 14 |
| 7 | --------------- | 15 |
| 8 | --------------- | 16 |
| 9 | --------------- | 17 |
| 11 | --------------- | 2 |
| 20 | --------------- | 1 |
| 21 | --------------- | 5 |

Parallel Cable for IBM

| Sirius | | Printer |
|---|---|---|
| 1 | --------------- | 1 |
| 2 | --------------- | 2 |
| 3 | --------------- | 3 |
| 4 | --------------- | 4 |
| 5 | --------------- | 5 |
| 6 | --------------- | 6 |
| 7 | --------------- | 7 |
| 8 | --------------- | 8 |
| 9 | --------------- | 9 |
| 10 | --------------- | 10 |
| 11 | --------------- | 11 |
| 13 | --------------- | 13 |
| 15 | --------------- | 16 |
| 16 | --------------- | 12 |
| 24 | --------------- | 24 |
| 25 | --------------- | 25 |

Parallel Cable for Epson MX80

| Sirius | | Printer |
|---|---|---|
| 1 | --------------- | 1 |
| 2 | --------------- | 2 |
| 3 | --------------- | 3 |
| 4 | --------------- | 4 |
| 5 | --------------- | 5 |
| 6 | --------------- | 6 |
| 7 | --------------- | 7 |
| 8 | --------------- | 8 |
| 9 | --------------- | 9 |
| 10 | --------------- | 10 |
| 11 | --------------- | 11 |
| 17 | --------------- | 17 |
| 19 | --------------- | 19 |
| 20 | --------------- | 20 |
| 21 | --------------- | 21 |
| 22 | --------------- | 22 |

## Parallel Cable for Oki Microline 84

```
 1 --------------- 1
 2 --------------- 2
 3 --------------- 3
 4 --------------- 4
 5 --------------- 5
 6 --------------- 6
 7 --------------- 7
 8 --------------- 8
 9 --------------- 9
27 ---|
10 --------------- 10
11 --------------- 11
17 --------------- 17
```

Switch Settings:

```
1 2 3 4 5 6 7 8
C C C C O C C C
```

## C.3 Serial Printer Cables

Serial Cable for QUME Sprint 5, DTC-300, Epson MX80, Datasouth DS180

```
Sirius              Printer
    1 --------------- 1
    2 --------------- 3
    3 --------------- 2
    5 --------------- 20
    7 --------------- 7
   20 --------------- 5
```

Serial Cable for Diablo 630

```
Sirius              Printer
    1 --------------- 1
    2 --------------- 3
    3 --------------- 2
    5 --------------- 20
    7 --------------- 7
    8 --------------- 5
   20 ---|      |---- 6
                |---- 8
```

Serial Cable for Mannesman-Tally (1805) and Anadex

```
Sirius              Printer
    1 --------------- 1
    2 --------------- 3
    3 --------------- 2
    5 --------------- 19
    7 --------------- 7
    8 ----|
   20 ----|
```

Serial Cable for Mannesman Tally MT-140 & MT-180

```
Sirius                          Tally
           2 ---------------- 3
           3 ---------------- 2
           5 ---------------- 11
           7 ---------------- 7
          11 ---------------- 5
           8 ----|-------|--- 8
          20 ----|       |--- 20
```

**Serial Cable for NEC Spinwriter 5520**

```
Sirius              Printer
    2 --------------- 3
    3 --------------- 2
    7 --------------- 7
    5 --------------- 19
                |------ 25
                |------ 6
                |------ 8
```

**Serial Cable for Oki Microline - Series (80/82A)**

```
Sirius              Printer
    1 --------------- 1
    2 --------------- 3
    7 --------------- 7
    5 --------------- 11
                |------ 6
                |------ 8
                |------ 20
```

```
SW1  (Busy polarity)
OFF

SW2  SW4  (1200 Baud)
ON   ON

SW6  (No Parity)
OFF
```

**Serial Cable for Qume Sprint 9**

```
Sirius              Printer
    1 --------------- 1
    2 --------------- 3
    3 --------------- 2
    7 --------------- 7
    5 --------------- 20
                |------ 4
                |------ 6
                |------ 8
```

```
A-1  A-2  A-3 (4800 baud)  A-4 (Self Test OFF)
OFF  ON   ON               OFF

A-5 (H/duplex) B-7  B-8 (H/ware handshake) C-3(WPS OFF)
OFF            OFF  OFF                    OFF
```

**Serial Cable for Olympia 103**

```
Sirius              Printer
    1 --------------- 1
    2 --------------- 3
    3 --------------- 2
    7 --------------- 7
    5 --------------- 4
```

```
                 Switch Settings for 4800 baud
1 2 3 4 5 6 7 8             1 2 3 4 5 6 7 8 9 10
C C C O C C O O             O C O C O O C C O O
```

**Serial Cable for TRD 170**

```
Sirius              Printer
    1 --------------- 1
    2 --------------- 3
    3 --------------- 2
    5 --------------- 4
    7 --------------- 7
```

**Serial Cable for IDS 560**

```
Sirius              Printer
    1 --------------- 1
    2 --------------- 3
    3 --------------- 2
    5 --------------- 20
    7 --------------- 7
    8 -----|
   20 -----|
```

**Serial Cable for Lear System 300**

```
Sirius              Printer
    1 --------------- 1
    2 --------------- 3
    3 --------------- 2
    7 --------------- 7
    5 --------------- 14
                |------ 6
                |------ 8
                |------ 20
```

**Serial Cable for Brother HR1**

```
Sirius                Printer
    1 --------------- 1
    2 --------------- 3
    3 --------------- 2
    7 --------------- 7
    5 --------------- 20
   20 --------------- 6
              |------ 8
```

**Serial Cable for Toshiba**

```
Sirius                Toshiba
    1 --------------- 1
    2 --------------- 3
    3 --------------- 2
                  |-- 5
    5 --------------- 20
    7 --------------- 7
```

**Serial Cable for Texas 810 with D.N.B. option**

```
Sirius                Texas
    1 --------------- 1
    2 --------------- 3
    3 --------------- 2
    7 --------------- 7
    5 --------------- 20
                 |--- 6
                 |--- 8
                 |--- 9
```

APPENDIX C C-10

# D. ASSEMBLER EXAMPLES

## D.1 EXAMPLE ASSEMBLER SHELL PROGRAM FOR MS-DOS INTERFACING

The Microsoft MACRO-86 assembler follows closely the Intel ASM-86 specifications. The operating system interfacing technique is via a straightforward interrupt (INT 21Hex), with the required operational parameter in the AH register. MS-DOS does not corrupt any registers other than the ones used for the sending or receiving of data. An example of the running and exiting program technique, plus the required assembler directives, follows. The program example is for the small memory model; but it will apply equally well to the compact or large memory model. The 8080 memory model is not recommended as it results in poor usage of the potential of the 8086/8088 processor. At link time, this programming example will generate an .EXE file - the header information on this file type will be found in E.1.

```
title   Example of MS-DOS/MACRO-86 Assembly Programming

dgroup  group   data
cgroup  group   code

msdos   equ     00021h                  ;interrupt to operating system

data    segment public    'data'
;######  insert your data here ######
data    ends

code    segment public    'code'
        assume  CS: cgroup, DS: dgroup

example proc    near                    ;origin of code

begin:
        push    ES                      ;save return segment address
        call    run_module              ;run the program
;
; run ends - select close down
;
exit    proc    far                     ;close down code
        xor     ax,ax                   ;zero for PSP:0
        push    ax                      ;save for far return
        ret                             ;and close down
exit    endp                            ;close down code ends
```

```
run_module:
        mov     ax,DATA                 ;get the data segment origin
        mov     DS,ax                   ; and initialise the segment
;##### insert your code at this point ######
        ret                             ;return to exit module
        example endp
        code    ends
        end
```

## D.2 Example Assembler Shell Program for CP/M-86 Interfacing

The Digital Research ASM-86 assembler does not follow the standard Intel ASM-86 structure - this makes for a more complex task when transferring assembler programs between the CP/M-86 and the MS-DOS operating systems. The operating system interfacing technique is via a straightforward interrupt (INT E0Hex), with the required operational parameter in the CL register. CP/M-86 corrupts all registers, excepting the CS and IP - it is, therefore, recommended that all registers be pushed prior to the INT E0Hex being issued. An example of the running and exiting program technique, plus the required assembly directives, follows. The program example follows that of the MS-DOS MACRO-86 example. At GENCMD time, this programming example will generate a .CMD file - the header information on this file type is shown in the System Guide for CP/M-86.

```
title   'Example of CP/M-86/ASM-86 Programming'

reset   equ     00000h                  ;system reset function
cpm     equ     000e0h                  ;interrupt to operating system

        cseg
begin:
        call    run_module              ;run the program
;
; run ends - select close down
;
        mov     cl,reset                ;select system reset
        mov     dl,00h                  ;select memory recovery
        int     cpm                     ;return to operating system
;
run_module:
;##### insert your code at this point ######
        ret                             ;return to exit module

        dseg
;##### insert your data here #####
        end
```

# E. MS-DOS - .EXE FILE HEADER STRUCTURE

The Microsoft linker outputs .EXE files in a relocatable format, suitable for quick loading into memory and relocation to any paragraph (16-byte) boundary. EXE files consist of the following parts:

- o Fixed length header
- o Relocation table
- o Memory image of resident program

EXE files are loaded in the following manner:

- o Read into RAM at any paragraph (16 byte) boundary
- o Relocation is then applied to all words described by the relocation table.

The resulting relocated program is then executable. Typically, programs using the PL/M small memory model have little or no relocation; programs using larger memory models have relocation for long calls, jumps, static long pointers, etc.

The following is a detailed description of the format of an EXE file:

### Microsoft .EXE File Main Header

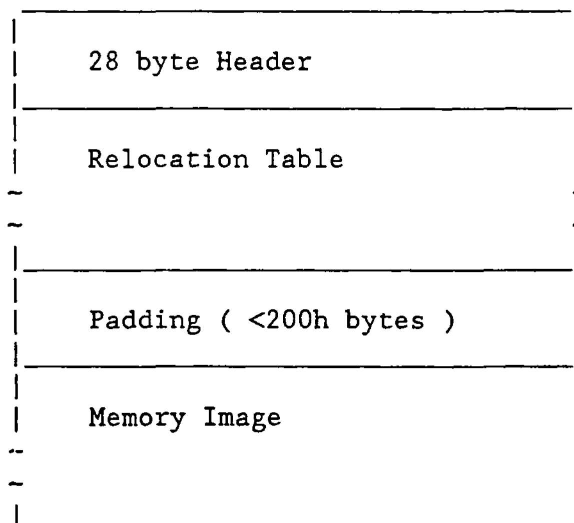
| Byte | Name | Function |
|---|---|---|
| 0+1 | wSignature | Must contain 4D5Ahex, this is the MS-LINK signature to mark the file as a valid .EXE file. |
| 2+3 | cbLastp | Number of bytes in the memory image modulo 512. If this is 0 then the last page is full, else it is the number of bytes in the last page. This is useful in reading overlays. |
| 4+5 | cpnRes | Size of the file in 512-byte pages including the end of the EXE file header. |
| 6+7 | irleMax | Number of relocation entries in the table. |
| 8+9 | cparDirectory | Number of paragraphs in EXE file header, used to locate the beginning of the memory image in the field. |
| A+B | cparMinAlloc | Minimum number of 16-byte paragraphs required above the end of the loaded program. |
| C+D | cparMaxAlloc | High/low loader switch, maximum number of 16-byte paragraphs required above the end of the loaded program. OFFFFh means that the program is located as low as possible into memory. |
| E+F | saStack | Initial value to be loaded into SS before starting program execution. |
| 10+11 | raStackInit | Initial value to be loaded into SP before starting program execution. |
| 12+13 | wchksum | Negative of the sum of all the words in the run file, ignoring overflow. |
| 14+15 | raStart | Initial value to be loaded into IP before starting program execution. |
| 16+17 | saStart | Initial value to be loaded into CS before starting program execution. |
| 18+19 | rbrgrle | Relative byte offset from beginning of run file to the relocation table. |
| 1A+1B | iov | Number of the overlay as generated by LINK-86. The resident part of a program will have iov = 0. |

The relocation table follows the fixed portion of the run file header and contains irleMax entries of type rleType, defined by:

```
rleType     bytes 0+1 ra
            bytes 2+3 sa
```

Taken together, the ra and sa fields are an 8086/8088 long pointer to a word in the EXE file to which the relocation factor is to be added. The relocation factor is expressed as the physical address of the first byte of the resident divided by 16. Note that the sa portion of an rle must first be relocated by the relocation factor before it in turn points to the actual word requiring relocation. For overlays, the rle is a long pointer from the beginning of the resident into the overlay area.

The format of the EXE file is:




The Memory Image begins at the first 512-byte boundary following the end of the Relocation table.

# SIRIUS 1 SPECIFICATIONS

## F.1 TECHNICAL SPECIFICATIONS

Processor
- o Intel 8088 16-bit microprocessor
- o 128k or 256k bytes standard (depending on model)
- o 335ns cycle time, 64k Dynamic RAM
- o 4 internal expansion slots for plug-in card options
- o 2 x RS232C serial communications ports. standard DB25-type connectors
- o 1 x Parallel (Centronics) or IEEE-488 port. 36 way amphenol connector
- o 2 x Parallel user port (50-way KK Connector on CPU board)
- o Expandable to 896K
- o 8k bytes of ROM (boot and diagnostic)
- o 4k bytes of screen RAM

Display System
- o 25 line x 80 column screen; 8 x 12 characters in a 10 x 16 cell
- o 50 line x 132 column screen
- o 300mm CRT, Green p39 phosphor
- o Adjustable horizontal viewing angle (± 45 degree swivel)
- o Adjustable vertical viewing angle (0 deg to +11 deg tilt)

Disc Drives
- o Standard 130cm, single-sided 96 TPI dual disc drives, with a maximum capacity of 600k bytes per drive.
- o Optional 130cm, double-sided 96 TPI dual disc drives, with a maximum capacity of 1200k bytes per drive.
- o Optional single 10,000k byte Hard Disc - non-removable; with single 130cm, double sided 96 TPI disc drive with a maximum capacity of 1200k bytes.
- o Single-sided floppy drive offers 80 tracks at 96 TPI
- o Double-sided floppy drive offers 160 tracks at 96 TPI
- o Floppy drives have 512 byte sectors; utilising a GCR, 10-bit recording technique.
- o Floppy access times:

  2 micro-seconds per bit data transfer rate, with an interleave factor of 3. Average seek time is approximately 94 milli-seconds.
  Track-to-track step time is 3ms. Latency = 100ms.
- o Hard Disc access times:

  0.2 micro-second per bit data transfer rate, with an interleave factor of 5. Average seek time is approximately 85 milli-seconds.
  Head settling time 15ms. Latency 8.33ms.

Keyboard
- Separate Intel 8048 microprocessor
- Fully software definable with 10 soft function keys
- Full IBM Selectric II (56 key) keyboard layout
- Type ahead buffering to 32 levels and full n-key rollover
- Capacitive keyswitches rated for 100 million operations
- Keyboard contains PCB with on-board drivers

Electrical
- Input voltage 90-137 VAC or 190-270 VAC (internal jumper)
- Input frequency 47-63 Hz

Environment
- Operating temperature 0oC to 40oC
- Operating humidity 20% to 80% (non-condensing)
- Storage temperature -20oC to 70oC
- Storage humidity 5% to 95% (non-condensing)

## F.2 Physical Specifications

Mainframe Assembly

| Height | Width | Depth | Weight (approx) |
|---|---|---|---|
| 178 mm | 422 mm | 356 mm | 12.6 kg |

Display Assembly

| Height | Width | Depth | Weight (approx) |
|---|---|---|---|
| 264 mm | 326 mm | 339 mm | 8.1 kg |

Keyboard Assembly

| Height | Width | Depth | Weight (approx) |
|---|---|---|---|
| 45 mm | 483 mm | 203 mm | 1.5 kg |

System Assembly

| Height | Width | Depth | Weight (approx) |
|---|---|---|---|
| 457 mm | 483 mm | 559 mm | 22.2 kg |

Width without the keyboard module is 396mm.

## G. GLOSSARY OF TERMS

The following table is a glossary of terms found in this manual:

BAUD — The term baud rate means the number of bits sent down a line per second. A baud rate of 300 will, therefore, be capable of transmitting data at 300 bits per second. Since a textual character is composed of 8 bits, then 37.5 characters could be sent per second at this baud rate.

BIOS — This means the Basic Input Output System. The BIOS is a fundamental portion of an Operating System, allowing the operating system to communicate correctly with any peripheral devices; typical BIOS modules include the disc driver; the keyboard input driver; the screen driver; the printer driver.

BIT — A bit is a binary digit. The bit can, therefore, contain either One or Zero. A One is bit HIGH or ON. A zero is bit LOW or OFF. A bit may be likened to a light-switch - the switch can only be on or off. See BYTE.

BOOT — This term comes from the phrase "the computer pulls itself up by its boot-strap". The term boot-strap means the same, but is no longer in such common use. To boot a computer is to load an operating system - the computer does this by means of a boot-strap program. The computer, when switched on, is not aware of its environment - but it automatically runs its boot-strap program. The Sirius 1 boot-strap program is stored in the boot PROM; it first causes the display of the little disc picture - it then searches for a disc with an operating system - when it finds this disc, it loads the operating system and begins to execute it. The boot-strap program is not used again until the reset switch is pressed, or the power is switched off and on.

BUS — A bus in computer jargon is not unlike a bus to carry passengers. When data is moved

around inside a computer it is moved along the bus wires. These bus wires connect the Sirius 1 microprocessor to its memory, disc(s) and screen.

BYTE — A byte is a collection of 8-bits or two nibbles. A byte may store one character of text, or a number from 0 to 255 in binary.

DOT MATRIX — A printed character on the screen or a dot-matrix printer may be viewed as a square containing dots. On the Sirius 1 screen a character has a square cell (matrix) of 16 dots high by 10 dots wide - within this box, the dot on/off patterns create a viewable character.

FONT CELL — In reference to DOT MATRIX, the font cell is the collection of bytes of data that make up the character dots that are to be displayed on the screen. Each character on the screen is composed of pre-defined patterns of dots to make the viewed dot matrix. These patterns of dots are stored in the Sirius 1 memory as data - the screen controller chip scans these data bytes and the resulting character image is displayed on the screen.

HEADER — A header on a file gives information to the operating system on where and how the file is to be loaded in to memory. Many files provided by Victor Technologies (such as keyboard and character set files) contain headers that are not used by the operating system, but are used by Victor Technologies utilities.

INTERRUPT — An interrupt is some event occurring in the computer's environment that the computer will stop all other activities for. An example of an interrupt is a key-press. If you press a key on the Sirius 1, an interrupt is generated; at this point the processor stores all information on its current task and gets and saves the value of the key pressed; it then picks up all the information it stored on its last task and continues where it left off. This whole series of events takes only a

few micro-seconds.

NIBBLE — Sometimes spelled NYBBLE; a nibble is half a byte or 4-bits. See BYTE and BIT.

OPERATING SYSTEM — An operating system allows the computer to be aware of its environment and gives the user the ability to enter and retrieve data from the computer.

PROM — Programmable Read Only Memory, PROM, is a chip or collection of chips that is used to store permanently a single computer program or collection of computer programs. The boot-prom, sometimes called boot-rom, contains all the information the Sirius 1 computer needs to read an operating system from disc. There are different types of prom; EPROM which is erasable prom, simply shine a high-powered ultra-violet lamp on the chip, and it can be re-programmed; etc.

RAM — Random Access Memory, RAM, is a chip or collection of chips that is used to store temporarily (until the power is removed) data, computer program(s), text, etc. This is the memory of a computer.

REGISTER — A computer register is a portion of the processor. The Sirius 1 uses the Intel 8088 micro-processor - there are several different types of registers within this chip; there are 8-bit registers, and 16-bit registers. Data is generally not manipulated in RAM, but is brought in to a register of the processor and manipulated there, then the result saved from the register back into RAM.

WORD — A word is a number of bits, generally greater than 8. The Sirius 1 has a 16-bit word - thus a word in the Sirius 1 is composed of two bytes. The DEC PDP-8 computer has a 12-bit word - on this machine, therefore, a word is composed of one byte and one nibble.

# H. DEALERS DEMONSTRATION PACKAGE

## H.1 Disc 1 'Latest Graphics Demo'

Disc 1 Menu contains:

| | | |
|---|---|---|
| 1. | SYSPEC | System Information<br>A written description of the business uses of the Sirius 1. Including a brief overview of the separate modules and system specifications. |
| 2. | BARS | Bar Graph<br>A demonstration of a bar chart depicting Snoocorp Profits. |
| 3. | PIE | Pie Chart<br>2 sets of pie charts depicting the National Budget. |
| 4. | CALENDAR | 1982 Calendar. |
| 5. | NOVEMBER | Sirius calendar 1981. |
| 6. | PLOT | Multiple Function Plots<br>Demonstrating the capabilities of the Sirius 1 for drawing frequency curves. |
| 7. | SCIENCE | Scientific Demonstration<br>Showing how the Sirius 1 can cope with scientific and mathematical notation and diagrams (eg. circuit diagrams). |
| 8. | GRAPHICS | Graphics Show<br>Gives a small graphics demonstration. |
| 9. | KEYS | Soft Keys - You Draw<br>Using the soft keys, numbers 1 to 7 - the operator can draw circles of different radii or draw lines given start and end positions. |

## Operating Instructions

To run this demonstration package type:

SUBMIT START <CR>

Before reaching the Menu options a voice message will be heard.

When the Menu is displayed, type a number between 1 and 9, selecting the options you require.

N.B. To proceed on certain options hit any key on the keyboard.

Options 1 to 8 are not interactive except for proceeding with the next display.

**Option 9 - Keys - Soft Keys - You draw**

Having selected this option, on line 25 will appear 7 functions which relate to the soft keys 1 - 7 on the keyboard. When used together, these keys allow the user to draw their own simple graphics ie. drawing lines and circles. The user can state where on the screen to construct the graphics. This is in relation to a pixel (matrix dot) which is originally positioned in the centre of the screen. The screen is divided into 800 pixels by 400 pixels.

Function keys are:

1. END
   This will finish the option and return to the menu.

2. PRINT (on/off switch)
   Allows the printing of any keyboard character.

3. DRAW CIRCLE
   When pressed, the command "Enter Centre" will appear. You move the illuminated pixel to the required centre then press soft key 3 again. "Enter Radius" will come up. You enter the required radius and press function key 3 again.

4. CONTINUOUS (on/off switch)
   This allows a solid line to be drawn when moving the illuminated pixel.

5. X = 0; Y = 0
This displays the position of the illuminated pixel on the screen as co-ordinates (to a maximum of + x = 400, x = -399, y = 190, y = 190).

Pressing function key 5 return the illuminated pixel to the centre of the screen.

6. STEP = 1
This shows the amount of pixels that will be covered when the highlighted pixel is moved. It has 4 settings 1,4,16 and 64.

7. ERASE BLOCK
This operation allows the user to erase a mistake or clear the screen by putting a block (rectangle) round it. All inside the block will be erased. When function key 7 is pressed "upper left" appears. You need to move the illuminated pixel to the top left position of an (imaginary) rectangle. When in position, press function key 7 again, "lower right" will be displayed. Position the illuminated pixel to the bottom right hand corner of the (imaginary) rectangle. Pressing function key 7 will erase the block.

```
UPPER LEFT  X------
            |     |
            |     |
            |     |
            -------X LOWER RIGHT
```

## Pixel Movement

To move the illuminated pixel around the screen use the numeric block numbers 1 to 9.

| | | |
|---|---|---|
| 7 | 8 | 9 |
| 4 | CHANGE<br>5<br>STEP | 6 |
| 1 | 2 | 3 |

## Graphics Dump

Options 2 to 8 can be dumped to various printers eg. Epson, Prism - but not the C.Itoh 1550.

In order to use the Epson MX80 Type 2 or 3 the operator would type:

```
        SUBMIT START MX80
or      SUBMIT START MX8D
```

the MX80 is for sideways print out and MX8D for upright print.

In order to use a Prism printer the operator would type:

```
        SUBMIT START PRSM
```

To dump any screen type P.

## H.2 Disc 2 - 'Sliding Picture Show'

Disc 2 contains the following demonstration programs:

1. DEMO — Displays pictures to show different aspects of the hi-res graphics.

   (cannot exit from program - need to reboot the system by pressing the reset switch at the back of the machine).

2. DEMOT — Shows the same pictures but slides them over the pixels (matrix dots) to appropriate position. (ALT-C to terminate program).

3. DEMOS — Slides the pictures in using complete blocks - also includes page 3 ladies (ALT-C to terminate program).

4. DEMON — Same as DEMOS except no nudes - cleaned up version. (Cannot exit program - need to reboot the system).

5. DEMO3D — Shows a revolving outline of the Sirius. The picture can be altered using the following keys:

   Y Lowers the Sirius
   shift Y Raises the Sirius

   B Tilts the Sirius backwards
   shift B Tilts the Sirius forwards

   R Reduces the Sirius
   shift R Enlarges the Sirius

   (ALT-C to terminate program).

## H.3 Disc 3 - 1550 (C.Itoh) Graphics

This disc contains the same menu as the disc 1 - 'Latest Graphics Demo' when the operator types:

SUBMIT START <CR>

For a complete explanation of the Menu options see pages H-1 to H-3.

The difference is that 1550 graphics will allow options 2 to 8 inclusive to be printed on the printer, whereas disc 1 dumps to Epson printers.

Type "P" when the graphics routine required to be printed is being displayed on the CRT, eg. pie charts. This causes the program to produce a pixel dump to the printer.

In order to achieve this the printer switches should be re-set to the graphics set shown on page , and the printer cable should be modified to the graphics cable shown on page .

Alternately the same demonstration without the voice can be started

SUBMIT DEMO <CR>.

The disc contains DEMO3D which is also on sliding picture show.

Instructions for moving the display are:

| | |
|---|---|
| B | Tilts the Sirius backwards |
| shift B | Tilts the Sirius forwards |
| R | Reduces the Sirius |
| shift R | Enlarges the Sirius |

A full screen picture of the Sirius logo and the word Sirius which fades and glows can be obtained by typing LOGO<CR>.

## H.4 Disc 4 - 'Arabic Demonstration'

As you would imagine instructions on how to obtain the best out of this demonstration are limited. The submit file START.SUB is automatically called in on booting up the machine. Unfortunately there is no picture show.

For those dealers who want to impress French customers there is a French Vocal file.

Instructions

1. Boot up the machine.

2. When BASIC86 is first called in (easy to guess when) hit control and C.

3. Type SYSTEM<CR>.

4. Type VOCF<CR>.

# I. INTERRUPT DRIVEN SERIAL INPUT/OUTPUT

## I.1 Introduction

This appendix is designed to show the methodology involved in driving the Sirius 1 in interrupt mode when communicating via the serial port(s). Some pitfalls are described, and tested sample routines are included. There are, currently, no system level facilities that enable this task to be accomplished easily, and some chips, namely the PIC 8259, PIT 8253, SIO 7201 and the VIA 6522 will require re-programming. It is up to the programmer to reset the machine to the original state prior to exiting the interrupt driven application.

A typical interrupt driven application will normally follow the steps outlined below:

1. Save the original vector, set the new vector.
2. Set the direction bits.
3. Enable clocks (internal or external).
4. Reset SIO 7201 device, define your communication characteristics.
5. Set the baud rate.
6. Set the PIC 8259 to enable SIO interrupts.

These steps will be discussed in more detail below.

## I.2 Interrupt Vectors

There are 256 software interrupts available to the Sirius 1. Most are reserved for system functions, and diagnostics. A block of vectors from 80H through BFH are set aside for applications.

### I.2.1 Vectors available on the Sirius 1

| | |
|---|---|
| 00-1Fhex | Intel reserved |
| 20-3Fhex | Microsoft reserved |
| 40-7Fhex | Victor reserved |
| 80-BFhex | Applications reserved |
| C0-FFhex | Victor reserved |

Vectors 40H through 47H are those belonging to devices controlled by the Programmable Interrupt Controller (PIC).

| | |
|---|---|
| 40hex | Sync IRQ |
| 41hex | SIO 7201 |
| 42hex | Timer 8253 |
| 43hex | General Interrupt Handler (all 6522 IRQ's) |

44hex IRQ4
45hex IRQ5
46hex Keyboard - keystroke
47hex 8087 math processor

### I.2.2 Location of Vectors

Vectors consist of a long pointer (double word) to an interrupt service routine. This pointer is a 4 byte entry consisting of the Segment and Offset of the Interrupt Service Routine. The vectors are stored in a table that has its origin at 0000:0000. The first entry in this table is, therefore, Interrupt 0; the vector for Interrupt 1 is the second, with its vector having an origin of 0000:0004. the interrupt vector for Interrupt 41hex (the SIO 7201) will be found at location 0000:0104 (4*41hex).

To set a vector into this table, the MS-DOS function 25hex can be used, but since it is desirable to restore the old vector prior to the application program exiting, it is less cumbersome to simply set the new vector "by hand", and restore the old vector when the application terminates.

### I.2.3 Set Vector - Assembler Example

```
;store old vector, and set new vector for SIO
     cli                                ;clear interrupts
     xor  ax,ax                         ;AX=0000
     mov  ES,ax                         ;access table via ES
     mov  ax,word ptr ES:[104h]         ;get old offset
     mov  word ptr old_offset,ax        ;save old offset in DS
     mov  ax,word ptr ES:[106h]         ;old segment
     mov  word ptr old_segment,ax       ;save old segment
     mov  ax,my_sio_isr                 ;get offset to my code
     mov  word ptr ES:[104h],ax         ;set vector offset
     mov  word ptr ES:[106h],CS         ; and the new segment
     sti                                ;enable interrupts
     ret                                ;all done, exit
;to replace the old vector prior to exit
     cli                                ;clear interrupts
     xor  ax,ax                         ;AX=0000
     mov  ES,ax                         ;access table via ES
     mov  ax,word ptr old_offset        ;get old offset
     mov  word ptr ES:[104h],ax         ;restore old offset
     mov  ax,word ptr old_segment       ;get old segment
     mov  word ptr ES:[106h],ax         ;restore old segment
     sti                                ;enable interrupts
     ret                                ;all done, exit
```

## I.3 Enabling Internal and External Clocks

In an asynchronous environment the transmit clock is generated internally, as opposed to a synchronous environment where the transmit clock is typically provided by an external source.

Internal clocking is selected by masking off the appropriate bit in register 1 of the keyboard Versatile Interface Adaptor (VIA).

The keyboard VIA, register 1, is located at E804:0001.

The appropriate bits are:

Bit 0 (PA0) for Port A
Bit 1 (PA1) for Port B

Thus, by setting PA0 to zero, the internal clock is enabled for port A; setting PA1 to zero will enable the internal clock for port B. Setting PA0 or PA1 to one will enable the external clock disabling the internal clock. CAUTION: Care must be taken to leave the other bits in the pre-selected state.

To enable internal clocks for ports A and B then mask off the two least significant bits in register 1:

```
mov  ax,0e804h                  ;keyboard VIA segment
mov  ES,ax                      ;select the segment register
and  byte ptr ES:[0001],0fch    ;A & B internal clocks done
```

To enable external clocks on either channel then set the relevant bit by OR'ing the bit in. The following sample sets the external clocks for both ports A and B:

```
mov  ax,0e804h                  ;keyboard VIA segment
mov  ES,ax                      ;select the segment register
or   byte ptr ES:[0001],03h     ;A & B external clocks done
```

### I.3.1 Providing Clocks

In a synchronous environment it sometimes becomes necessary to provide transmit and receive clocks from the Sirius 1. This requires that the cable used has to have pins 15, 17 and 24 jumpered at the Sirius 1 end. The Sirus 1 always has a clock on pin 24, this being provided by the internal baud rate generator; thus by jumpering pin 24 to both pins 15 and 17, this clock becomes available for both the transmitter and the receiver, at

both ends of the cable.

When providing clocks from the Sirius 1, the external clock must be set as well as a baud rate selected. In synchronous mode, the "divide by rate" of the PIT 8253 is 1, therefore the values used to set the required baud rate is 1/16 the values used in an asynchronous environment. (see section 3.8.2 for values).

## I.4 Initialising the SIO

There is little magic used in this step, but it is recommended that the programmer read the entire Intel/NEC 7201 chip data sheet. The SIO segment is found in segment location E004hex. The offsets for the data ports A and B and control ports A and B are at 0, 1, 2, 3 respectively.

The following example of initialising the SIO 7201 is for Port A:

```
     cli                              ;disable interrupts
     mov  ax,0e004h                   ;the SIO segment
     mov  ES,ax                       ; using ES
     mov  byte ptr ES:[0002h],18h     ;channel reset

;now delay at least 4 system clock cycles

     nop
     nop                              ;delay for 7201

     mov  byte ptr ES:[0002h],12h     ;reset external/status
                                      ; interrupts

;and select register 2

     mov  byte ptr ES:[0002h],14h     ;non-vectored
     mov  byte ptr ES:[0003h],02h     ;select CR2 B
     mov  byte ptr ES:[0003h],00h     ;set vector to 0

;set for clock rate of 16*; 1 stop bit; parity disabled

     move byte ptr ES:[0002h],04h     ;select CR4 A
     move byte ptr ES:[0002h],44h     ;

;this register defines the operation of the receiver:
;7 data bits; auto enable and receive enable

     mov  byte ptr ES:[0002h],03h     ;select CR3 A
     mov  byte ptr ES:[0002h],61h     ;
```

```
;CR5 controls the operation of the transmitter
;7 data bits, dtr; assumes half-duplex

     mov  byte ptr ES:[0002h],05h  ;select CR5 A
     mov  byte ptr ES:[0002h],0a0h ;

;set status: affects the vector, interrupt on every character,
;enable transmitter interrupt

     mov  byte ptr ES:[0002h],01h  ;select CR1 A
     mov  byte ptr ES:[0002h],17h  ;
     sti                           ;enable interrupts
```

### I.4.1 Baud Rate for SIO

At this point, baud rate must be selected. In an asynchronous environment the PIT 8253 divides the supplied baud rate by 16; but in a synchronous environment the baud rate is divided by 1. Thus, to set the baud rate in an asynchronous environment, the value written to the PIT 8253 is 16 times the desired baud rate value. The common baud rate values, and the method of establishing the baud rates, are shown in section 3.8.2 of this manual.

### I.4.2 Set the PIC to Enable SIO Interrupts

In the Sirius 1 the PIC is normally initialised to operate the SIO in a polled environment. The following lines of code sets the PIC to operate the SIO in an interrupt environment:

The PIC resides at segment E000hex and the register required here is at offset 0001:

```
     cli                           ;disable interrupts
     mov  ax,0e000h                ;get the PIC segment
     mov  ES,ax                    ;
     and  byte ptr ES:[0001h],(not 02h) ;mask off bit 1
     .
     .
     sti                           ;allow interrupts
```

Prior to exiting the interrupt driven application, the PIC should be returned to operating the SIO in polled mode. This is done by setting bit 1:

```
     cli                           ;disable interrupts
     mov  ax,0e000h                ;get the PIC segment
```

```
    mov  ES,ax                     ;
    or   byte ptr ES:[0001h],02h  ;set polled
    sti                           ;allow interrupts
```

## I.5 Interrupt Service Routine - ISR

When an interrupt occurs in non-vectored mode, SIO register CR2 B contains the vector number of the interrupting device. Assuming the SIO was initialised as earlier described in this appendix, CR2 B contains a value in the range 0-7, which serves as the index to the following interrupt vector table

### I.5.1 Sample Interrupt Service Routine

```
data segment    public    'data'
int_vectors     dw   tx_int_b              ;tx int for port B
                dw   ext_status_b          ;external status changed
                dw   recv_int_b            ;recv int port B
                dw   recv_err_b            ;recv error port B
                dw   tx_int_a              ;tx in for port A
                dw   ext_status_a          ;external status changed
                dw   recv_int_a            ;recv in port A
                dw   recv_err_a            ;recv error port A
data ends

code segment    public    'code'
     assume     CS:cgroup, DS:dgroup
sio_isr:
     mov  word ptr CS:current ss,SS        ;save stack seg
     mov  word ptr CS:current_sp,SP        ; and stack pointer
     mov  SS,word ptr CS:ss_origin         ;internal stack
     mov  SP,offset dgroup:stack top       ;defined in DS (dgroup)
     push ax                               ;save environment
     push bx
     push cx
     push dx
     push bp
     push DS
     push ES

     mov  DS,dgroup                        ;set to internal data
     mov  ax,0e004h                        ;set SIO segment
     mov  ES,ax                            ;
     mov  byte ptr ES:[003h],02h           ;select CR2 B
     mov  al,ES:[0003h]                    ;read int device
     add  al,al                            ;word align
     mov  ah,0                             ; hi = 0
     mov  bx,offset int_vectors            ;get vector table
```

```
     add  bx,ax                         ;point to entry
     call [bx]                          ;service routine
     cli                                ;keep disabled

;now an "end of interrupt" (EOI) must be issued to the
;SIO (port A) and to the PIC.

     mov  ax,0e000h                     ;PIC segment
     mov  DS,ax                         ;
     mov  byte ptr [0042h],38h          ;EOI to ctrl A of SIO
     mov  byte ptr [0000h],61h          ;EOI to PIC ctrl port A

     pop  ES                            ;restore environment
     pop  DS
     pop  bp
     pop  dx
     pop  cx
     pop  bx
     pop  ax

     mov  SS,word ptr CS:current_ss     ;get SS
     mov  SP,word ptr CS:current_sp     ;get SP
     iret                               ;interrupt return

;the SS origin is stored here during initialisation
ss_origin      dw   0                   ;stack segment origin
current_sp     dw   0                   ;SP on ISR entry
current_ss     dw   0                   ;SS on ISR entry
```

NOTE: Some variables are stored within the code segment, as the CS register is the only register containing a known value at the time of interrupt.

## I.6 Setting Direction Bits

This function need only be performed once, and is performed by the operating system BIOS following a hardware reset. This step need not be implemented, therefore, if a standard Sirius operating system is used. If a standard operating system is not used, then this step needs to be performed immediately prior to the enable clock code.

```
;The offset to the data direction register is 0003hex.

     cli                                ;disable interrupts
     mov  ax,0e804h                     ;kbd VIA segment
     mov  ES,ax                         ;
```

```
     mov  al,byte ptr ES:[0003h]       ;get the old value
     or   al,03h                       ;set for output
;
;set the PA2-5 to zero, to enable DSR and RI input
;
     and  al,0c3h                      ;mask in
     mov  byte ptr ES:[0003h],al       ;rewrite new value
     .
     .
     sti                               ;enable interrupts
```

# J. FILE HEADER INFORMATION

## J.1 Character Set Header

All files with the extension .CHR are Character Set table files. These files contain data corresponding to the actual dot matrix displayed for each character on the console. These files also contain information regarding the character set name, version number, origin, date of creation, and display class. The Character Set table file header is a 128 byte field, structured as follows:

| Byte No. Hex | Dec | Function |
|---|---|---|
| 00 | 00 | Character Set type, ASCII 'C'=character |
| 01 | 01 | Character Set Version No. (ASCII 0 through 9) |
| 02-0D | 02-13 | Display Class |
| 0E-15 | 14-21 | Character Set Name |
| 16 | 22 | Filler (ASCII Space) |
| 17-19 | 23-25 | Banner Class |
| 1A | 26 | Filler (ASCII Space) |
| 1B-3D | 27-61 | Comment |
| 3E-4D | 62-77 | Originator |
| 4E-55 | 78-55 | Creation Date - arranged as YY/MM/DD |
| 56-59 | 86-89 | Number of records in the file in ASCII. A character set file of 128 characters has 32 records; a character set file of 256 characters has 64 records. The record count for a 32 record file is stored as 30 30 33 32 (0032) |
| 5A-5B | 90-91 | Reserved |

| Byte No. Hex | Dec | Function |
|---|---|---|
| 5C | 92 | This byte is used to house three variables. Bit 7 is used to show the Horizontal/Vertical alignment of the character set - bit 7 ON infers a Vertical character set. Bits 6 through 4 of the high nibble is used to store the binary Super/Subscript value (which may be 1 through 7) offset from 1 - thus a Super/Subscript value of two would be stored as binary 2. The low nibble is used to store the binary Character Height offset from 0 - thus a Character Height value of 16 would be stored as binary F. The Character height is a function of the number of vertical pixels the character will occupy in the 16x10 pixel matrix available for each character on the screen. If the Horizontal/Vertical bit, the Super/Subscript value and the Character Height value was as stated above, then this byte would read AF. The byte appears: |

| Bit | [7] | [ 6 5 4 ] | [ 3 2 1 0 ] |
|---|---|---|---|
| Function | Horiz/Vert | Super/Sub | Character Height |

| Byte No. Hex | Dec | Function |
|---|---|---|
| 5D | 93 | This byte contains two values; the User/System character set toggle, bit 0 stores this value; and the Stock/Special character set toggle, bit 1 stores this value. Bit 0 ON infers that the character set is a special character set. |
| 5E | 94 | This byte contains information on the character set width. If the high nibble is 0, then the low nibble contains the binary information, offset from 0, of all the characters in the character set - thus a character set width value of 16 would be stored as F. If the high nibble is F, then the character set is a proportional one - the proportional |

| | | |
|---|---|---|
| | | character set has a trailing record containing information on the width of each individual character in the character set. A proportional character set is designed to be used in high-resolution mode as it requires a 16x16 screen cell. |
| 5F-7F | 95-127 | Reserved. |
| 80- | 128- | The character set font information. |

### J.1.1 Sample Character Set Table File Header

Following is an actual header taken from the Character Set table file for the character set PROP.CHR. PROP contains 128 characters, and is a proportional character set:

| Hex Offset | Value in Hex | |
|---|---|---|
| 0: | 43 30 49 6E 74 27 6C 20 | 20 20 20 20 20 20 50 52 |
| 10: | 4F 50 20 20 20 20 20 43 | 48 52 20 54 68 69 6E 20 |
| 20: | 70 72 6F 70 6F 72 74 69 | 6F 6E 61 6C 20 63 68 61 |
| 30: | 72 61 63 74 65 72 20 73 | 65 74 20 20 20 20 53 69 |
| 40: | 72 69 75 73 20 53 79 73 | 74 65 6D 73 20 20 38 32 |
| 50: | 2F 30 37 2F 31 36 30 30 | 33 30 00 00 7F 00 FF 00 |
| 60: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
| 70: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |

## J.2 Proportional Character Set Trailer Information

In the case of a proportional character set, the trailing 128 bytes of the character set file contains information on the proportional width of each of the characters in the file. A proportional character set may not, therefore, contain more than 256 characters.

The following is a sample taken from the character set PROP.CHR; the hex figures represent the width for each proportional character starting with the space character. Note that each width value is offset from 0, with a value range of 1 through 16 decimal. Each byte is stored, and represented below, in low/high order; the two nibbles would be exchanged to give the value to the character(s) in high/low order. Each character is mapped from the proportional width as follows:

29 95 98 49 77 88 84 93

The above figures are for the first 16 display characters including the space character - they correspond as follows:

| | | |
|---|---|---|
| space | = 10 | (corresponding to 9) |
| ! | = 3 | (corresponding to 2) |
| " | = 6 | (corresponding to 5) |
| # | = 10 | (corresponding to 9) |
| $ | = 9 | (corresponding to 8) |
| % | = 10 | (corresponding to 9) |
| & | = 10 | (corresponding to 9) |
| ' | = 5 | (corresponding to 4) |
| ( | = 8 | (corresponding to 7) |
| ) | = 8 | (corresponding to 7) |
| * | = 9 | (corresponding to 8) |

etc.

## J.3 Keyboard Table Header

All files with the extension .KB are Keyboard Table files. these files contain information regarding keyboard code generated when a key on the keyboard is pressed. These files also contain information regarding the Keyboard Table name, version number, origin, date of creation, and display class. The Keyboard Table table file header is a 128 byte field, structured as follows:

| Byte No. Hex | Dec | Function |
|---|---|---|
| 00 | 00 | Keyboard table type, ASCII 'K'=character |
| 01 | 01 | Keyboard table Version No. (ASCII 0 through 9) |
| 02-0D | 02-13 | Display Class |
| 0E-15 | 14-21 | Keyboard Table Name |
| 16 | 22 | Filler (ASCII Space) |
| 17-19 | 23-25 | Banner Class |
| 1A | 26 | Filler (ASCII Space) |
| 1B-3D | 27-61 | Comment |
| 3E-4D | 62-77 | Originator |
| 4E-55 | 78-55 | Creation Date - arranged as YY/MM/DD |

| | | |
|---|---|---|
| 56-59 | 86-89 | Number of records in the file in ASCII. A character set file of 128 characters has 32 records; a character set file of 256 characters has 64 records. The record count for a 32 record file is stored as 30 30 33 32 (0032) |
| 5A-7F | 90-127 | Reserved |
| 80- | 128- | Keyboard table information |

### J.4 Banner Skeleton Files

Files with the extension .BAN are banner skeleton files. The banner is information printed on the screen during system boot. The banner also prints the Logo (if selected) along with other information regarding configuration. The banner is a set of ASCII strings containing the escape sequences and characters necessary to print the logo and configuration information on the console.

The first 128 bytes of the Banner Skeleton has the following format. The first byte is zero followed by 0Dh, 0Ah. This is followed by the length of the file in ASCII decimal with a leading and trailing space, and followed by 0Dh, 0Ah.

The location of the keyboard name and character set name follow in the same format as the file name length. If the file length is 639 characters, the keyboard name is at byte 502, and the character set name is at 541, then the first 24 bytes of the banner file would be as follows:

```
30 0D 0A 20 36 33 39 20 0D 0A 20 35 30 32 20 0D 0A 20
35 34 31 20 0D 0A
```

### J.5 Banner Customisation

Would you like your company name included on the Sirius banner? Syselect permits the selection to be made, but first you need to create a banner.

This may be achieved by using ED (under CP/M-86), EDLIN (under MS-DOS) or WordStar.

1. CP/M-86

```
A>PIP YOUR.BAN=SIRIUS.BAN<CR>
A>ED YOUR.BAN<CR>
```

```
:*#a<CR>
1:*2:0tt<cr>
2: xxxxx SIRIUS I  MICROCOMPUTER xxxxx

   (must be exactly as shown)    (must be exactly 24 chrs)
2:*sSIRIUS I MICROCOMPUTER^Z    PUT YOUR NAME HERE  ^Z0tt<cr>
2:*e
A>
```

2. MS-DOS

```
A>COPY SIRIUS.BAN YOUR.BAN<CR>
A>EDLIN YOUR.BAN<CR>

EDLIN version 1.01
End of input file

*alt clr                    (clear screen)

rSIRIUS I  MICROCOMPUTER^Z    PUT YOUR NAME HERE  ^Z0tt<cr>
                              PUT YOUR NAME HERE  (EDLIN
                                                  repeats)
*E
A>
```

Syselect may now be used to include your banner file in an operating system.

## J.6 Logo Creation

A logo editor is available. Contact Barson Computers for details.

## J.7 Normal File Control Block

The normal file control block is structured as follows - with offsets in decimal:

| Byte | Contents |
|---|---|
| 0 | The drive number. The drives are numbered as follows:<br><br>Before opening file: 0 = default drive<br>1 = drive A<br>2 = drive B<br>3 = drive C, etc. |

After opening file: 1 = drive A
2 = drive B, etc.

MS-DOS replaces the default drive prefix of 0 with the correct drive number after the open is processed.

1-8 Filename, left justified with trailing ASCII space(s). If a device name is placed in this region, the trailing colon should be omitted.

9-11 Extent, left justified with trailing ASCII space(s).

12-13 Current block number relative to the beginning of the file, starting with zero (automatically set to zero by the open function request). A block consists of 128 records, each record being of the size specified in the logical record size field. The current block number is used with the current record field for sequential reads/writes.

14-15 Logical record size in bytes. Set to 80H by the open function request.

16-19 File size in bytes. The first word represents the low-order part of the file size.

20-21 Date the file was created or last updated. The date is set by the open function request. The date is formatted as follows:

```
<          21          > <         20    >
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
 y  y  y  y  y  y y m m m m d d d d d
```

where m = month 1-12
d = day 1-31
y = year 0-119 (1980 - 2099)

22-23 Time the file was created or last updated. The time is set by the open function request. The time is formatted as follows:

```
<          23         > <        22   >
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
 h  h  h  h  h  m m m m m m s s s s s

where h = hours      0-23
      m = minutes    0-59
      s = seconds*2  0-59
```

24-31 Reserved for system use.

32 Current relative record number (0-127) within the current block. This must be set before doing sequential read/write operations on the file. The open function request does not set this field.

33-36 Relative record number, relative to the origin of the file, starting at zero. This field must be set prior to doing random read/write operations on the file. The open function request does not set this field.

If the record size is less than 64 bytes, both words are used. If the record size is greater than 64 bytes, then only the first three bytes are used.

Notes: The File Control Block at 5CH in the Base Page overlaps both the File Control Block at 6CH and the first byte of the command line area/disc transfer area at 80H.

Bytes 0 - 15 and 32 - 36 must be set by the user program. Bytes 16 - 31 are set by MS-DOS and may only be changed at the programmers own risk.
In the 8086/8088 all word fields are stored least significant byte first - this is true in setting the record length, etc.

## J.8 Extended File Control Block

The extended FCB is used to create or search for files having special attributes. The extended FCB adds an additional 7 bytes preceeding the normal FCB. The extended FCB is structured as follows:

| Byte | Contents |
| --- | --- |
| FCB-7 | Set to FFH indicates that an extended FCB is being used. |
| FCB-6 - FCB-2 | Are reserved. |
| FCB-1 | Attribute byte to include hidden files (02H) or system files (04H) in directory searches. |
| FCB-0 | Origin of normal FCB (drive byte). |

## K. COMPARISONS BETWEEN MS-DOS AND CP/M-86

| FUNCTION | NR | MSDOS(AH) | CPM-86(CL) |
|---|---|---|---|
| Terminate | 0 | NC | DL=0 =>back to CCP<br>DL=1 =>remain in memory |
| Conin | 1 | Returns:<br>AL=ascii char<br>checks are made<br>input is echoed | Returns:<br>AL=ascii char<br>checks are made<br>input is echoed |
| Conout | 2 | DL is char | DL is char |
| Aux /rdr | 3 | Returns:<br>AL is char | Returns:<br>AL is char |
| Aux/pun | 4 | DL is char | DL is char |
| Print | 5 | DL is char | DL is char |
| Dir Con I/O | 6 | DL=FF=>Cons input<br>Returns:<br>AL=00=>no char<br>AL<>00=>Cons char<br>DL<>FF=>Cons output<br>DL is char | DL=FF=>Cons input<br>Returns:<br>AL=cons char<br>DL=FE=>Cons stat<br>Returns:<br>AL=00=>no char<br>DL<FE=>cons output<br>DL is char |
| Dir con Input /<br>Get I/O byte | 7 | Return AL=char | Returns:<br>AL=IOBYTE |
| Conin,No Echo/<br>Set I/O byte | 8 | See func 1 | DL=IOBYTE |
| Print String | 9 | DS:DX=[charstr$]<br>uses func 2 | DX=[charstr$] |
| Read console<br>Buffer | A | DS:DX=[datalen,<br>inlen,data..]<br>last char always CR<br>inlen doesn't<br>include CR.<br>Editing functions<br>apply. | DX=[datalen,<br>inlen, data..]<br>CR not in data |

| FUNCTION | NR | MSDOS(AH) | CPM-86(CL) |
|---|---|---|---|
| Constat | B | Returns:<br>AL=FF=>char<br>=00=>no char | Returns:<br>AL=01=>char<br>=00=>no char |
| Conin with flush/<br>Get version # | C | Flush buffer then<br>do function 1,6,7,8<br>or A if in AL.<br>See above for<br>parameters | Returns:<br>BX=version |
| Disc reset (system) | D | Set A, flushes<br>buffers | Select A and<br>makes all R/W |
| Select disc | E | DL=drive<br>Returns:<br>AL=number of drives | DL=drive |
| Open file | F | DS:DX=[FCBu]<br>FCB (E,F,20)=00H<br>Returns:<br>AL=00=>Okay<br>=FF=>not found | DX=[FCBu]<br>FCB(C)=0<br>Returns:<br>AL=00=>Okay<br>=FF=>not found |
| Close file | 10 | DS:DX=[FCBo]<br>Returns:<br>AL=00=>Okay<br>=FF=>not found | DX=[FCBo]<br>Returns:<br>AL=(0,1,2,3)=>Ok<br>=FF=>not found |
| Search 1st entry | 11 | DS:DX=[FCBu]<br>Return:<br>AL=FF=>not found<br>2 FCB's: extended,<br>normal | DX=[FCBu]<br>Return:<br>AL=FF=>not found<br>AL=(0,1,2,3)=> Okay<br>Curr DMA=dir record |
| Search next entry | 12 | DS:DX=[FCB] see 11<br>Return:<br>AL=FF=>not found | Must be preceded<br>by 11 or 12.<br>See 11 for return |
| Delete file | 13 | DS:DX=[FCBu]<br>Return:<br>AL=FF=>not found<br>=00=>okay | DX=[FCB]<br>Return:<br>AL=FF=not found<br>=00=>okay |

| FUNCTION | NR | MSDOS(AH) | CPM-86(CL) |
|---|---|---|---|
| Read sequential | 14 | DS:DX=[FCBo]<br>Return:<br>AL=00=>okay<br>=01=>no data<br>=02=>not enough mem<br>=03=>partial record<br>Into current transfer address | DX=[FCBo]<br>Return:<br>AL=00=>okay<br>=01=>eof-no data<br>Into current DMA |
| Write seg. | 15 | DS:DX=[FCBo]<br>Return:<br>AL=00<br>=01=>disc full<br>=02=>not enough memory | DX=[FCBo]<br>Return:<br>AL=00=>okay<br>=01=>dir full<br>=02=>data full |
| Make file | 16 | DS:DX=[FCBu]<br>Return:<br>AL=00<br>=FF=>no dir.space<br>Inits to zero len file | DX=[FCBu]<br>Return:<br>AL=00,01,02,03=>Ok<br>Inits to zero len file |
| Rename file | 17 | DS:DX=[FCBr]<br>Return:<br>AL=00>ok<br>=FF=>not found<br>DS:DX+11H=new filename<br>"?"supported | DX=[FCBr]<br>Return:<br>AL=00=>ok<br>=FF=>not found<br>DX+10H=new filename<br>unambiguous filename |
| Not defined / Return login vect | 18 | ND | Return:<br>BX=login vector for discs |
| Current disc | 19 | Return:<br>AL=current sel drive<br>0=>A,... | Return:<br>AL=current disc<br>0=>A,... |
| Set disc xfr / addr set DMA offset | 1A | DS:DX=disc transfer addr | DX=DMA offset |
| Allocation table/ vector | 1B | Return:<br>DS:BX=allocation table<br>DX=alloc unit count<br>AL=records per alloc unit<br>CX=sector size | Return:<br>ES:BX=allocation vector |

| FUNCTION | NR | MSDOS(AH) | CPM-86(CL) |
| --- | --- | --- | --- |
| Not defined /<br>writ prot. disc | 1C | ND | No parameters |
| /Get R/O vector | 1D | ND | BX=R/O Vector |
| /Set file<br>attributes | 1E | ND | DX=[FCB]<br>with attributes set<br>Return:<br>AL=00=>ok<br>=FF=>not found |
| /Get disc<br>parameter | 1F | ND | ES:BX=[disc parameters] |
| /Set/get user<br>code | 20 | ND | DL=FF=>gets user code<br>0->F=>set user code<br>Return:<br>AL=current code<br>if FF in DL |
| Random read | 21 | DS:DX=[FCBo]<br>Return:<br>AL=00=>ok<br>01=>no more data<br>02=>not enough mem<br>03=>partial | DX=[FCBo]<br>Return:<br>AL=00=>ok<br>01=>read unwritten.<br>02=>NC<br>03=>can't close ext.<br>04=>unwritten extent<br>05=>NC<br>06=>out of range |
| Random write | 22 | DS:DX[FCBo]<br>Return:<br>AL=00=>ok<br>01=>disc full<br>02=>not enough mem | DX=[FCBo]<br>Return:<br>AL=00=>ok<br>01=>NC<br>02=>no avail data<br>03=>can't close ext<br>04=>NC<br>05=>no dir space<br>06=>out of range |
| File size | 23 | DS:DX=[FCBu]<br>Return:<br>AL=00=>ok<br>=FF=>not found<br>Random record set | DX=[FCB]random<br>Return:<br>AL=00=>ok<br>FF=>not found<br>Random record set |

| FUNCTION | NR | MSDOS(AH) | CPM-86(CL) |
|---|---|---|---|
| Set random rec | 24 | DS:DX=[FCBo]<br>Set random record<br>in record FCB=<br>current blocks | DX=[FCBo]<br>Set random record<br>in record FCB=<br>next sequential |
| Set vector /<br>Reset drive | 25 | AL=interrupt to set<br>DS:DX=vector values | DX=drive vector<br>return AL=00 |
| Create program/ | 26 | DX=new prog segment | ND |
| Random block rd/ | 27 | DS:DX=[FCBo]<br>CX=record count<br>Return:<br>01=>eof, complete<br>02=>not enough mem.<br>03=>eof,partial<br>CX=record count of read<br>FCB updated | ND |
| Random block wrt | 28 | See 27 except<br>Return:<br>AL=01=>no disc space<br>If CX=0=>set file<br>size to random record | See 22 DX=[FCBo]<br>Zero fills data<br>block |
| Parse file name/ | 29 | DS:SI=[string,cr]source<br>ES:DI=[FCBu] destination<br>AL=0=>no scan off<br>reading gaps<br>1=>scan off<br>FCBu is created<br>Return:<br>AL=00=>ok<br>AL=01=>ok, ? Or * present<br>DS:SI=[name]<br>ES:DI=[blank] if no name | ND |
| Get date / | 2A | Return:<br>CX=year<br>DH=month<br>DL=day | ND |
| Set date/ | 2B | CX,DX as 2A | ND |

| FUNCTION | NR | MSDOS(AH) | CPM-86(CL) |
|---|---|---|---|
| Get time/ | 2C | CH=hour<br>CL=min<br>DH=sec<br>DL=1/100ths | ND |
| Set time/ | 2D | CX,DX as 2C | ND |
| Set/Reset verify flag/ | 2E | DL=0 | ND |
| /Direct Bios call | 32 | ND | DX=[parameter block] |
| /Set DMA base | 33 | ND | DX=DMA base |
| /Get DMA base | 34 | ND | Return:<br>ES:BX=DMA base |
| /Get max mem | 35 | ND | DX=[MCB]<br>Return:<br>AL=00=>Ok<br>=FF=>no mem. |
| /Get ABS max | 36 | ND | DX=[MCB]<br>Return:<br>as for 35 |
| /Alloc. mem | 37 | ND | DX=[MCB]<br>Return:<br>AL=00=>Ok<br>=FF=>insuff. mem. |
| /Alloc ABS mem | 38 | ND | DX=[MCB]<br>Return:<br>as for 37 |
| /Free mem | 39 | ND | DX=[MCB] |
| /Free all mem | 3A | ND | No parameters |
| /Program Load | 3B | ND | DX=[FCBo]<br>ret AX=FFFFH=><br>bad load else<br>AX=base page<br>BX=base page. |

## L. FEATURES TO BE INCLUDED IN MS-DOS VERSION 2

| ITEM | DESCRIPTION |
|---|---|
| 1 | A DOS function call will be added to the DOS to allow a program to load/give control to a ".EXE" file beginning at the current DMA address. The calling program would simply provide an FCB formatted with the called program filename, any parameters, the segment at which to set up the PSP, and a flag (more on flag later). The function would (1) set up the program segment and parameters, (2) load the program (any file can be loaded this way, but if its extension is .EXE, it is relocated on the way in), and (3) optionally give the program control. |
| 2 | A volume identification will be internally stored on a diskette and a mechanism for specification and display of this identification will be provided. |
| 3 | Function key assignments will be made to facilitate use of the most commonly used DOS functions. An external command will allow character strings to be assigned by the user, with special keyword or parameter to reset keys back to DOS defaults. |
| 4 | XENIX-compatible function calls. |
| 5 | The DOS will be modified to allow the use of more disc I/O buffers as memory allows. |
| 6 | A spooling utility will be incorporated for the printer that will allow processing to continue with printing performed in a background environment. A mechanism will be provided to disable this function for users that cannot spare the necessary spooling disc space. |
| 7 | A conditional continuation will be added to the BATCH facility that allows the user to conditionally continue with a batch file based upon the results of preceeding programs and utilities. As a part of this capability a function call will be added to allow user programs to terminate and store the user program termination condition code. |
| 8 | The I/O system is redefined to perform character I/O using named device drivers. When input or output with a named device is requested by a program, the device name table is searched for the needed driver. If no matching name is found as a character device, the name is assumed to refer to a disc file. |

| ITEM | DESCRIPTION |
|---|---|
| 9 | Device error trapping will be expanded such that at the option of the program, I/O device errors are trappable, in the same manner (but using a different interrupt vector) as disc errors are trapped. This approach is consistent with XENIX Signals. |
| 10 | Console input and/or console output may be assigned to a file before execution of a command or program. Or, the output of one command or program may be directed as the input to the next. This approximates the XENIX Pipe. This goes beyond the CP/M XSUBMIT facility. |
| 11 | The utility EDLIN will be enhanced and modified with the following:<br>a. The backup file is not deleted until necessary.<br>b. It is possible to recover from a full disc.<br>c. Multiple commands are allowed on one line.<br>d. Search and Replace commands continue searching with the line after the previous match (ie. default first parameter is current line plus one).<br>e. Search and Replace default to the previous search string.<br>f. A new listing command "P" moves the current line as it lists.<br>g. Lines may be referenced relative to the current line.<br>h. Allow the user to move text around within the file.<br>i. Allow the user to copy outside files into his text. |
| 12 | Microsoft will consolidate MS-DOS messages within the DOS and utilities. |
| 13 | DEBUG - add an ASSEMBLE option using M86 assembler syntax, and change UNASSEMBLE to use the same syntax. |
| 14 | CLS command to clear the screen from the console or batch file - this could also be a useful function call. |
| 15 | Disc catalogue - holds names/volids of all files, used for prompting user to insert required diskette when correct volume not currently mounted. |
| 16 | Security scheme for data integrity/protection (file ?). Details to be determined. |

# M. SIRIUS 1 DEALER SPARE PARTS KIT

| SIRIUS P/N | DESCRIPTION |
|---|---|
| 100617-01 | PCB, Video |
| 100078-02 | PCB Assy, Power Supply 220/240V |
| 100353-01 (5) | Fuse, 20/240V |
| 100092-01 | Disc Drive Floppy |
| 100670-01 | PCB Assy,Disc Drive |
| 100812-01 | Fan Assy,220/240V |
| 100470-01 | PCB Assy,CPU |
| 100480-01 | Switch,Rock |
| 100516-01 | Switch Assy, Reset |
| 100792-01 (5) | Switch, Keyboard |
| 100820-01 | Suitcase,Spares Kit |
| 100036-02 | Cable, Keyboard |
| 100007-05 | Keyboard, |

# N. DOUBLE SIDED DISKETTES

## N.1 DOUBLE SIDED DISKETTES

Release 2.2 of the CP/M-86 operating system can use double-sided floppy drives. If you are using a machine with double-sided drives, you can create diskettes which have roughly twice the storage capacity that was possible with earlier releases of the operating system.

### A Word of Caution about using Double-sided Diskettes

The great advantage in using double-sided diskettes is that the amount of data stored on the diskette is almost twice that of a standard single-sided diskette: 1.2 megabytes compared to 610 kilobytes. A single-sided diskette can be used on a computer with either single-or double-sided drives. However, once a diskette is formatted as double-sided NONE of the data on it can be accessed by a computer which has single-sided drives.

If you have a computer which has double-sided drives, you may not want to convert all of your diskettes into double-sided diskettes right away, since a majority of the computers currently in use are only single-sided and you will not be able to use your diskettes on them.

### Creating a Double-sided Diskette

You can only create a double-sided diskette on a computer which has double-sided floppy drives. To create a double-sided diskette, you need to use version 2.8 of the FORMAT program. If your computer has double-sided drives, the FORMAT program asks the following question on the 25th line of the screen:

Format both sides of the diskette (y/n) ?

If you answer by typing a 'Y' the program creates a double-sided diskette. If you answer with a 'N' just the first side of the diskette is formatted and you have a single-sided diskette which can be used on both single- and double-sided machines.

Remember that you can only create double-sided diskettes on a computer with double-sided floppy drives, having created the double-sided diskette you can copy the operating system across using either BOOTCOPY (CP/M-86) or SYSCOPY (MS-DOS).

Copying a Double-sided Diskette

To make a copy of a double-sided diskette, you must use Version 2.5 or later, of the DCOPY program. You do not have to tell the DCOPY program whether or not the diskette to be copied is double-sided: the program automatically knows. To copy a double-sided diskette, you must have double-sided drives in your computer. If you try to DCOPY a double-sided diskette on a computer which has only single-sided drives, DCOPY prints an error message and aborts the copy.

Converting Single- and Double-sided Diskettes

If you have a computer that has double-sided floppy drives, you may want to convert some of your single-sided diskettes into double-sided diskettes (see the caution above). To do this, create a double-sided diskette by using the FORMAT program as described above. Then use PIP (CPM/86) or COPY (MS-DOS) to copy the desired files from the single-sided diskette to the double-sided diskette you just created.

At times you may need to convert a double-sided diskette back into a single-sided diskette. Using the FORMAT program create a single-sided diskette. Then use PIP (CPM/86) or COPY (MS-DOS) to copy the desired files from the double-sided diskette to the single-sided diskette. You may need two or more diskettes to hold all of the data from a double-sided diskette.

Diskette LED's

The LED associated with each diskette is lit during diskette I/O. Specifically, it is lit at the initiation of a read or write and remains lit for approximately 1/2 second after the I/O is completed. The LED also stays lit as long as there is a "dirty buffer" in RAM since this is considered as being "in the middle" of a write operation.

This implementation permits the following simple rule for end users: **"do not remove the diskette when the LED is lit."**

If a diskette is removed while there is a "dirty buffer" in RAM, the drive is marked Read Only (R/O). This prevents destroying disc data in case the diskette is replaced by another diskette.

Diskette Label

Track 0 of each disc has a label which contains information

relating to the structure of the diskette (such as location of the directory).

Prior to the 2.0 release space was reserved for the label, but the label was not used by the system.

With release 2.0, two types of diskette may exist. One type contains boot tracks, starting with track 1, with the directory and data following. The other type has the directory and data starting at track 1.

## N.2 Boot Disc Label Format

**Track 0 Sector 0**

| Byte Offset | Name | Description |
|---|---|---|
| 0 | System disc ID | literally, ff,00h for a system disc |
| 2 | Load address | paragraph to load booted program at. If zero then boot loads in high memory. |
| 4 | Length | paragraph count to load. |
| 6 | Entry offset | I.P. value for transfer of control. |
| 8 | Entry segment | C.S. value for transfer of control. |
| 10 | I.D. | disc identifier. |
| 18 | Part number | system identifier - displayed by early versions of boot. |
| 26 | Sector size | byte count for sectors. |
| 28 | Data start | first data sector on disc (absolute sectors). |
| 30 | Boot start | first absolute sector of program for boot to load at 'load address' for 'length' paragraphs. |

| | | |
|---|---|---|
| 32 | Flags | indicators:<br>bit meaning<br>15-12 interleave factor (0-15)<br>0 0=single sided<br>1=double sided |
| 34 | Disc type | 00 = CP/M<br>01 = MS-DOS |
| 35 | Reserved | |
| 38 | Speed table | information for speed control proc. |
| 56 | Zone table | high track for each zone. |
| 71 | Sector/track | sectors per track for each zone. |

# O. FUNCTIONAL SPECIFICATIONS OF THE BOOT ROM

## O.1 Diagnostic ROM Board Support

1. Access to disc routines in boot ROM provided.

2. CPU registers are saved in low memory prior to memory test. Low memory from 0 to 0ffh is not tested or cleared.

The boot ROM loads diskette resident systems from a description contained on track 0 sector 0 of a diskette. A logical flow follows:

1. Disable interrupts.

2. Clear the CRT controller.

3. If a memory test of the first 16K of memory fails then halt the boot. Else set the first 16K of memory to zero.

4. If a test of the screen RAM fails then halt the boot. Else set the screen RAM to zero.

5. Load the character set into the dot matrix.

6. Initialise, clear and set to high intensity the CRT.

7. Turn on the arrow display.

8. Initialise the diskette and display the diskette image at middle screen.

9. Turn off all disc drives.

10. Select drive A.

11. If a drive door closes, select that drive as the boot disc otherwise every 32nd time blink the arrow display.

12. When a drive has been selected then turn off the display and turn on the disc drive motor.

13. Read a header record from track 0 sector 0 and if an error go to the disc error output.

14. Otherwise display the clock and memory determination icons and test for end of memory.

15. Save the memory size for comparison with system requirements and display it on the CRT with the clock off.

16. Turn the clock on and read the diskette definitions and if a disc read error occurs report it via the disc display.

17. Check the disc label and if invalid go to the not a system disc error display.

18. Bring the disc online with all disc parameters loaded.

19. Compute the parameters to load the system and then load the system into memory. After successfully loading it transfers to the system via a programmed interrupt 255.

20. On the event of a reportable error display it then waits for the disc door to open at which time control is passed to 6. above.

### O.2 ICONS for boot ROM Version P1

At power on in the middle of the bottom line a flashing arrow and a diskette are displayed if both disc drive doors are open. When a drive door is detected to have closed that disc is selected for loading a system. A memory symbol, a large 'M' at the left of middle is displayed and a clock image replaces the arrow/disc formerly displayed. If the memory required cannot be read from the disc record then an image containing, diskette, large 'X', and an error code is displayed on right of middle. If the memory required by the target system is more than the processors capacity then an image with a large 'X' and the size of memory in paragraphs follows the 'M' at left of middle.

Normal load sequence

| | | | | |
|---|---|---|---|---|
| 1. | | | arrow/disc | |
| 2. | M | | clock | |
| 3. | M | pppp | clock | [] |
| 4. | M | pppp | | [] |
| 5. | | | clock | |

6. Target system display.

## O.3 Exception Displays

1. M X pppp
   Means that the disc system requested more memory than the processor has.

2. [] X ee where [] = diskette image.
   Means that a diskette error occurred of type 'ee'.

'ee' Error Codes

| ee value | error description |
|---|---|
| 01 | no sync pulse detected |
| 02 | no header track |
| 03 | checksum error in header |
| 04 | not right track |
| 05 | not right sector |
| 06 | not a data block |
| 07 | data checksum error |
| 08 | sync too long |
| 99 | not a system disc |

## O.4 Universal Boot EPROMS

New boot EPROMs have been developed which allow the Sirus 1 to boot off any available device; ie. floppy, hard disc, network, etc. These EPROMs not only eliminate the need for specialised sets which were used in the past, but also provide some primitive yet effective diagnostic capabilities to field service personnel.

A system equipped with Universal EPROMs can be easily identified by the different ICONS which are displayed following a power on reset or push button reset. The memory sizing ICON is reported in kilo-bytes rather than Segments (M 128K instead of M 2000). The type of device ICON from which the CPU is trying to

boot from is displayed along with the specific number (0 for floppy A or hard disc 0, and 1 for floppy B or hard disc 1). A new ICON has been installed and will be presented to the screen while the system tries to boot off the Network.

When the CPU is powered on, or reset, the Universal EPROMs will execute diagnostic tests on the screen RAM, boot ROM checksum, dynamic RAM (DRAM), programmable interrupt controller (PIC), and some of the I/O devices. If the CPU encounters an error during these diagnostic tests, it will report the error either to the screen (assuming enough RAM is functional), or via an OUTPUT instruction to I/O port 0FFFF hex.

In the case where there is enough functional circuitry to report the error to the screen, the error code will be reported on the 25th line along with ICON display.

The method for more catastrophic errors is to report, via the output instruction, the type of error in the UPPER NIBBLE of the data byte, and if possible the failing device in the LOWER NIBBLE of the data byte. This is done by doing a write to I/O port 0FFFF hex. The boot code then loops on this instruction allowing a technician to use an oscilloscope to analyse the failure.

| ERROR CODE | | TYPE OF ERROR | BAD DEVICE |
|---|---|---|---|
| UPPER | LOWER | | |
| 0 | 1 | Screen Ram, not reproduced | Undetermined |
| 0 | 2 | Rom Checksum Error | Boot Rom |
| 0 | 3 | DRAM, not reproduced | Undetermined |
| 0 | 4 | Internal CPU Error | 8088 Failure |
| 1 | X | Screen Ram, Single Bit | X=failing bit |
| 2 | X | Screen Ram, Multiple Bits | X=1st fail bit |
| 3 | X | DRAM, Single Bit | X=failing bit |
| 4 | X | DRAM, Multiple Bits | X=1st fail bit |

Two examples of CPU error detection where there is sufficient circuitry available to report failure to the screen, are listed below.

M 16K 3X

In this example, the CPU has found the first 16K bytes of dynamic ram to be functional but found a faulty ram location in an area above the 16K bytes. The code 3X hex is defined as

follows; the upper nibble 3 indicates a single bit failure in dynamic ram, and the X would be in the range of 0 - F to indicate which ram bit contained the failure.

**M 16K 4X**

This error code is defined as follows; the upper nibble 4 indicates a multiple bit dynamic ram failure, and the lower nibble X indicates the first failing ram bit (starting with the most significant bit). Replace this device and repeat test until system boots or other error code is present.

As stated previously the Universal Boot EPROM also tests the PIC and the three 6522's resident on the CPU board. The CPU will write to some of the registers within the devices and then attempt to read back the value written to that register. If the CPU cannot read back the same value written, then the faulty I/O device will be reported to the 25th line on the CRT screen. This error code will appear to the right of the Device ICON, and is described below.

(25th line) M 128K X 1234

1 programmable interrupt controller
2 parallel port interface
3 keyboard interface
4 user port interface

Example:

**M 128K X0030**

This indicates a diagnostic fault while trying to access the keyboard interface.

## P. TRANSFERRING ASCII FILES FROM COMMODORE TO SIRIUS

ASCII files including programs may be transferred from C.B.M. computers to SIRIUS via the C.B.M. user port and the SIRIUS 1 control port.

CABLE - a download cable is attached to the 50-pin internal user port (Jack-5) on the SIRIUS CPU board. The other end of the cable is connected to the user port (J2) of the Commodore.

| C.B.M. (J2) | | | SIRIUS (J5) | |
|---|---|---|---|---|
| GND | A | ------------------------- | 11 | GND |
| PA0 | C | ------------------------- | 16 | PA0 |
| PA1 | D | ------------------------- | 18 | PA1 |
| PA2 | E | ------------------------- | 20 | PA2 |
| PA3 | F | ------------------------- | 22 | PA3 |
| PA4 | H | ------------------------- | 24 | PA4 |
| PA5 | J | ------------------------- | 26 | PA5 |
| PA6 | K | ------------------------- | 28 | PA6 |
| PA7 | L | ------------------------- | 30 | PA7 |
| CB2 | M | ------------------------- | 12 | DSTB |

SOFTWARE - For Sirius, a utility called PIPPIN.CMD is available from most dealers and distributers. It will work under CP/M-86 or MS-DOS (with emulation) perfectly.

By invoking A>PIPPIN FILENAME=INP:[E] <cr>

PIPPIN can also be invoked by PIPPIN <cr> and a '*' will appear to accept the remaining command, the difference is that PIPPIN will stay active if invoked this way and allow for multiple transfer of files without it returning to the operating system and having to be re-loaded each time.

The byte-stream transmitted from the source machine can be stored in the file, FILENAME, on the SIRIUS. This byte-stream must be terminated with an ASCII end-of-file character, ^Z (1AH). The [E] option causes PIPPIN to echo the transfer to the screen. "Wildcard" file designators may not be used in this operation.

SOFTWARE - For C.B.M., the following program may be used for the Commodore computer. If compiled it will run significantly faster.

Note: If you wish to transfer **programs** from the Commodore, you must first convert them to ASCII. This may be done as follows:

```
DLOAD"PROGRAM NAME",D0  <cr>
DOPEN#5,"ASCII NAME",W,D0:CMD5:LIST  <cr>
PRINT#5,:DCLOSE  <cr>
```

You now have an ASCII version of your program on disc.

```
10 REM ****************************************
20 REM *                                      *
30 REM * C.B.M. TO SIRIUS ASCII FILE TRANSFER *
40 REM *                                      *
50 REM *  BY KEITH PICKUP - BARSON COMPUTERS  *
55 REM *                                      *
60 REM *    this version for C.B.M. 8032      *
65 REM *                                      *
70 REM *  s=home  ^=clr/home  q=cursor down   *
80 REM *        r=rvs/on     o=rvs/off        *
85 REM *                                      *
90 REM ****************************************
100 PRINT"sss^";
110 SP$="                                                  ":SP$=SP$+SP$
120 PRINT"^qr C.B.M. TO SIRIUS 1 FILE TRANSFER PROGRAM - (C) KEITH PICKUP";
130 PRINT" [BARSON COMPUTERS] o";
140 PRINT:PRINT:INPUT"WHAT IS THE NAME OF THE PROGRAM TO TRANSFER ";PG$
150 DOPEN#5,(PG$),R:SC=33487
160 IFDS=62THENPRINT"qrFILE - NOT FOUNDo":FORY=1TO1500:NEXT:DCLOSE:GOTO100
170 PRINT:PRINT"TRANSFERRING FILE r";PG$;"o TO SIRIUS 1"
175 PRINT CHR$(15);
180 GET#5,A$
190 IFST=64THEN460                            :REM DO END OF FILE 1A HEX
200 X=ASC(A$):IF X>127 THEN X=X-128           :REM KEEP TO ASCII
205 :                                         :REM <CR> & <LF>
210 IFX=13THENGOSUB1000:X=10:GOSUB1000:CT=0:PRINT"^";:GOTO180
270 IFX<1THEN X=32
280 POKESC+CT,X
300 GOSUB 1000:GOTO 180                       :GO TO MAIN OUTPUT ROUTINE
460 REM END OF JOB
470 X=13:GOSUB1000:X=10:GOSUB1000:X=26:GOSUB1000:X=0:GOSUB1000:DCLOSE
480 PRINT"qqqqqqqJOB COMPLETED - O.K. ":FORK=1TO3000:NEXT:CLR:GOTO100
1000 POKE59459,255                            :REM SET DDR 0-7 AS OUTPUTS
1010 POKE59467,PEEK(59467)AND227              :REM DISABLE SHIFT REG
1020 POKE59468,PEEK(59468)AND31OR192          :REM SET CB2 LOW
1030 POKE 59471,X:CT=CT+1                     :REM WRITE VAR. X TO ORA
1040 POKE59468,PEEK(59468)AND31OR224          :REM SET CB2 HIGH
1050 RETURN
```

# Q. UNPROTECTING DISCS

If you have ever accidentally re-saved your basic program onto disc in protected mode, overwriting your existing source program, it is possible to recover it. Below is a sample session.

```
A>MSBASIC
BASIC-86 Rev. 5.27
[MS-DOS Version]
Copyright 1977-1982 (C) by Microsoft
Created: 8-Nov-82
62281 Bytes free
Ok
load"pinstall"          ** load in protected program MSBASIC will
                           find the first byte read is 'FE', this
                           tells BASIC that it must unscramble the
                           following program
Ok
new                     ** type new, this does not erase the program
                           but inserts 3 bytes '00 00 00' at the
                           start of memory for program storage.
                           These 3 bytes normally are at the end of
                           your program to signify it's end mark.
Ok
def seg:bsave"prog",2700,50000
                        ** this sets the default segment and saves a
                           block of memory to disc. Within this block
                           is your program in it's unscrambled form
Ok
system                  ** exit to the operating system

A>debug prog            ** load DEBUG and your saved memory block
```

```
DEBUG-86  version 1.07
>d                                  ** display memory from 0100
05A7:0100  FD 79 0A 8C 0A 50 C3 00-00 00 00 00 00 00 00 00   }y...PC.........
05A7:0110  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
05A7:0120  64 00 3A 8F DB 00 B3 0A-6E 00 3A 8F DB 00 C4 0A   d.:.[.3.n.:.[.D.
05A7:0130  78 00 3A 8F DB 20 50 49-4E 53 54 41 4C 4C 00 05   x.:.[ PINSTALL..
05A7:0140  0B 82 00 3A 8F DB 20 54-68 69 73 20 70 72 6F 67   ...:.[ This prog
05A7:0150  72 61 6D 20 69 6E 73 74-61 6C 6C 73 20 76 61 72   ram installs var
05A7:0160  69 6F 75 73 20 70 72 69-6E 74 65 72 73 20 66 6F   ious printers fo
05A7:0170  72 20 57 6F 72 64 53 74-61 72 20 33 2E 30 32 00   r WordStar 3.02./

                                             ** at address 011D are the 3 bytes that were
                                                inserted after the 'new' command, so by
                                                placing a 'FF' byte at 011E we are setting
                                                the normal unprotected first byte
         >e011e
         05A7:011E  00.ff
         >nprog.bas                          ** rename the program with a .BAS extension
```

# R. ASYNC PROTOCOL

## R.1 Data Block

Async transmits data in 133-byte blocks with headers and trailers as follows. All byte values are in hexadecimal.

| header byte 01h | block # 1 to FFh then 0 to FFh | block # XOR FFh | 128 data bytes | checksum | NAK 15h |
|---|---|---|---|---|---|

Notes:

1. The first byte in the block is a header byte. A value of 01h indicates that this block is a data block.

2. The second byte in the block is the block number which is 1 for the first block transmitted, 2 for the second block, etc. When the value passes FFh the block numbers start again at zero.

3. The third byte is the value of block number exclusive-or'd with FFh. This provides a check on the integrity of the block number.

4. Following the third byte is 128 bytes of data.

5. Following the data is a checksum byte. The checksum is formed by taking the algebraic sum of all the data bytes, any carry or overflow is discarded.

6. The last byte in the block is the NAK character (15h).

7. The receiver commences by sending NAK characters. When the transmitter receives a NAK character it starts transmitting.

8. The receiver responds with the following single characters:

   ACK (06h) - block received correctly
   NAK (15h) - incorrect frame, checksum not correct
   CAN (18h) - receiver wishes to abort

9. The transmitter may send the following single characters:

   CAN (18h) - abort
   EOT (04h) - end of transmission

## R.2 File Name Transmission Blocks

Multiple files may be transmitted by the sender, in which case the data is preceded by a file name block.

There are two types of file name blocks:

1. Block containing a file name.

| header byte<br><br>02h | file name indicator<br><br>24h | file name<br>8-byte name plus<br>3-byte extension | checksum on file name<br>1 byte | EOT<br><br>04h |
|---|---|---|---|---|

2. Block indicating no more files.

| header byte<br><br>02h | no-more-files byte<br><br>25h | EOT<br><br>04h |
|---|---|---|

Notes:

1. A header byte of 02h indicates a file name block.

2. The second byte has the value 24h if the block contains a file name or the value 25h if there are no more files names to come.

3. The file name is an 11-byte string containing 8 bytes for the file name (packed with spaces if necessary) and 3 bytes for the extension.

4. The checksum is on the 11-byte file name.

5. The last byte is the EOT character (04h).

# S. COMMUNICATIONS

## S.1 IBM REMOTE BATCH EMULATION

### Product Description

1. Half Duplex, synchronous operation.

2. Requires no additional hardware.

3. 1200 to 9600 Baud.

4. Device types supported by the package are 2770, 2780, 3741 or 3780.

5. Can be used to transfer files between two Sirius computers.

6. Files can be transmitted from the Sirius and both line printer and punch data can be received either to disc files or direct to the printer.

7. Console support is included to allow interaction with the host processor and local control of the emulator functions.

8. Commands normally input by the operator at the console may be prepared in a disc file and submitted to the emulator with a single command, thereby making the package suitable for use by operators unfamiliar with the syntax required by the host machine.

9. Chaining can be invoked to enable multiple receive files to be written to individual disc files.

10. Transmission can be in either transparent or non-transparent mode.

11. The emulator will not operate in a multidrop environment (ie. point-to-point link only).

12. Operating System - CP/M-86 or CP/M-86 emulator under MS-DOS.

### Product Requirements:

To enable the remote batch emulation package to operate the following conditions must be satisfied on the host computer:

| | | tick |
|---|---|---|
| a. | Binary Synchronous communications port | [ ] |
| b. | Half Duplex | [ ] |
| c. | System is generated with: | |
| | (i) Only one terminal configured on the port | [ ] |
| | (ii) This terminal to be either: 2770 | [ ] |
| | or 2780 | [ ] |
| | or 3741 | [ ] |
| | or 3780 | [ ] |
| d. | The Sirius must be connected to the host by one of the following methods: | |
| | Synchronous modems | [ ] or |
| | A modem eliminator | [ ] or |
| | Short haul modems (limited distance) | [ ] |

S.2 IBM 3270 EMULATION PACKAGE

**Product Description**

1. Half Duplex, synchronous operation.

2. Requires no additional hardware.

3. 1200 to 9600 Baud.

4. The package is capable of interpreting screen formatting sequences, data link control and handling polling responses, time out control and cyclic redundancy checking to ensure the integrity of received data.

5. Printer support is provided to emulate IBM printer types 3284 or 3286. The printer definition can define the operating system list device as the printer or a disc file may be used.

6. Diagnostic aids incorporated in the package include an analog loopback test for terminal to modem interface

checking and a built-in trace buffer used to keep a cyclic record of all data and protocol traffic transmitted and received.

7. Device types supported by the package are 3271 controller with 3277 video display or optionally a 3275 video display may be emulated.

8. The Sirius keyboard can be configured to provide the special key functions found on the 3277 device (eg. PF1, PA1 etc).

9. The package is capable of communicating in either ASCII or EBCDIC codes. Built in translation tables are automatically invoked for EBCDIC transmissions.

10. Operating System - MS-DOS.

Product Requirements

To enable the 3270 emulation package to operate the following conditions must be satisfied on the host computer:

tick

a. Binary Synchronous communications port [ ]

b. Half Duplex [ ]

c. System is generated with:

(i) Only one terminal configured on the port [ ]

(ii) This terminal to be a 3271 controller with a 3277 video display [ ]

(iii) The control unit address (CUA) should be set to hex 40 * [ ]

(iv) The device unit address (DUA) should be set to hex 40 * [ ]

(v) The Sirius must be connected to the host by one of the following methods:

Synchronous modems [ ]
or
A modem eliminator [ ]

or

Short haul modems (limited distance) [ ]

* This package can be configured to use different control unit address or device unit address.

## S.3 ASYNCHRONOUS COMMUNICATIONS PACKAGE

**Product Description**

1. Full or Half Duplex.

2. 50 to 4800 Baud.

3. Terminal Emulation mode allows the Sirius to emulate a teletype compatible device for interactive sessions with a host computer.
   Files may also be transmitted and received in this mode subject to the limitations/coinditions detailed below.

4. The user may configure the package in terms of bits per character, parity checking and number of stop bits.

   Handshaking is defined by the rules outlined below.

5. Datalink mode allows the Sirius to transfer files to either another Sirius using the ASYNC package or to another computer capable of handling the special protocol framing and checksum sequences used for this transmission method. (See appendix R for details of the protocol).

6. Configuration options also allow the user to select how much memory is used within the Sirius for data buffers ie. how much data is received from the transmission line before handshaking procedures are invoked.

7. Requires no additional hardware.

8. Operating System - CP/M-86 or CP/M-86 emulator under MS-DOS.

**Product Requirements**

1. Terminal Emulation - Interactive Use

To enable the ASYNC comms package to communicate as a "teletype compatible" device in an interactive mode the host machine must have the following facilities:

tick

a. Asynchronous Communications Interface [ ]

b. Baud rate selectable in range 50-4800 Baud [ ]

c. Full or Half Duplex facilities [ ]

d. Eight bits per character [ ]

e. No parity [ ]

f. One stop bit [ ]

If all of the above conditions are satisfied then the package, as supplied, will function provided a suitable cable is used for connection.

N.B. Conditions d,e and f are re-configurable in the ASYNC package, if the host settings are different. Note that the terminal emulation only provides for teletype compatibility, and therefore will not function correctly if it receives special screen control characters eg. VT100 interfaces.

2. Terminal Emulation - File transfer

File transfer is possible in terminal emulation mode under the following conditions:

tick

a. Host ----> Sirius

(i) Host machine must obey CTL/S - CTL/Q (XON/XOFF) handshaking [ ]

(ii) The file to be received must fit on available disc space [ ]

(iii) No error checking is carried out [ ]

b. Sirius ----> Host

(i) Host must transmit a linefeed character (decimal 10) to the Sirius each time a carriage return (decimal 13) is received from the Sirius .

(This is the handshaking procedure

employed for file transfer from the Sirius) [ ]

(ii) No error checking is carried out [ ]

3. **Datalink Mode**

This mode employs an error checking protocol whereby the transmitting Sirius envelopes consecutive blocks of data with frame header and trailer characters and also appends a block check character to each block. The receiving Sirius strips off the frame characters and validates the block check character. The Datalink mode incorporates all error checking and automatic retransmission in the event of errors.

a. This mode can only be used to transfer files between two Sirius machines or between Sirius and Apricot [ ]

For a description of the datalink mode, see appendix R.

## S.4 ASYNC PACKAGE - REMOTE TERMINAL

An attractive feature of the Asynchronous Communications package allows for remote operation of another Sirius. When one Sirius computer is connected to another Sirius (or to another computer of similar capabilities), the second computer can be driven from the keyboard and screen of the first.

To use this technique, the 'IOBYTE' facility must be implemented on the second machine (allowing redirection of logical devices to any physical device). The console of the second computer is redirected to the communications link and therefore, to the screen and keyboard of the first Sirius 1 computer.

In the case of Sirius to Sirius this is supported under CP/M-86 version 1.1/2.4 or later and MS-DOS version 1.25/2.5 or later, using port A.

### Remote Operations

The operator of the second machine (known as remote terminal) redirects the console to the asynchronous communications port, port A.

Type:

STAT CON:=TTY: <CR>

The screen at the remote terminal is normally blank during the communication operations but everything that appears on the first Sirius' (local terminal) screen can be echoed remotely so before typing STAT CON:=TTY:

Type:

STAT LST:=CRT: <CR>
<ALT>P

To run the remote terminal from the keyboard of the first machine (local terminal) load the ASYNC package into the local terminal and put it into terminal emulation mode.

Type:

ASYNC T <CR>
press the <CR> key twice

The remote terminal operating system prompt should appear. The local terminal can now access files and programs from the remote terminal.

eg. DIR <CR>

Produces a directory listing on the local screen of the disc file contents of the remote Sirius.

**Transferring Files from one Computer to Another**

If both computers have ASYNC on line, no one needs to be at the remote terminal - the whole process can be run from your local terminal. Transfer of files is carried out in datalink mode.

**Remote Terminal to Local Terminal**

1. On remote terminal type:

STAT CON:=TTY: <CR>

2. On local terminal type:

   ASYNC T <CR>
   press <CR> once or twice

3. The operating system prompt of the remote terminal will appear.
   On local terminal invoke datalink mode for remote terminal, in preparation to send a file eg. TEST.DAT.

   Type:

   ASYNC SQ TEST.DAT <CR>

   (The secondary option, Q, keeps cursor control characters and status messages from disturbing transmission).

4. On local terminal exit from ASYNC.

   Type:

   <ALT> V
   <ALT> E

5. On local terminal invoke ASYNC in datalink mode in order to receive file eg. TEST.DAT. Then revert to Terminal mode when transfer is complete.

   Type:

   ASYNC RT TEST.DAT <CR>

**Local Terminal to Remote Terminal**

1. On remote terminal type:

   STAT CON:=TTY: <CR>

2. On local terminal type:

   ASYNC T <CR>
   press <CR> once or twice

3. The operating system prompt of the remote terminal should appear.
   On local terminal invoke datalink mode for remote terminal, in preparation to receive a file eg. DEMO.TXT

Type:

ASYNC RQ DEMO.TXT <CR>

4. On local terminal exit from ASYNC.

Type:

<ALT> V
<ALT> E

5. On local terminal invoke ASYNC in datalink mode in order to send a file eg. DEMO.TXT. Then reset to terminal mode when transfer is complete.

Type:

ASYNC ST DEMO.TXT <CR>

## READER'S COMMENT FORM

Your comments are our main source of ideas for improvement. Please use this form to provide us with feedback on this document.

DOCUMENT:
Title: Supplementary Technical Reference Manual

YOUR GENERAL REACTION:

| | | | |
|---|---|---|---|
| Overall quality: | [ ]Excellent | [ ]Adequate | [ ]Poor |
| Text Clarity: | [ ]Very clear | [ ]Adequate | [ ]Poor |
| Usefulness of format: | [ ]Helpful | [ ]Adequate | [ ]Poor |

YOUR SPECIFIC COMMENTS:

Did you find any errors in the document? ______________________
If so, describe: ______________________________________________
_______________________________________________________________
_______________________________________________________________

Was any important information omitted from the document? _______
If so, describe: ______________________________________________
_______________________________________________________________
_______________________________________________________________

What sections of the document were especially useful to you?
_______________________________________________________________
_______________________________________________________________

What sections were of no use to you? __________________________
_______________________________________________________________
_______________________________________________________________

How could the material be presented to be more helpful to you?
_______________________________________________________________
_______________________________________________________________

READER'S NAME: _______________________________________________
JOB TITLE: ___________________________________________________
COMPANY: _____________________________________________________
ADDRESS: _____________________________________________________
_______________________________________________________________
_______________________________________________________________

- O V E R -

Please complete and return this form to:
Attn: Sirius Technical Support

| Barson Computers Pty. Ltd. or | Barson Computers P/L., or | Barson Computers (NZ) |
|---|---|---|
| 335 Johnston St., | 331 Pacific Hwy., | P.O. Box 36045, |
| MELBOURNE VIC 3067 | CROWS NEST NSW 2065 | AUCKLAND NEW ZEALAND |