APM
WORKING DOCUMENTS

FIRST EDITION
REVISION 1.0

May 1983
Reprinted Jan 2006 with corrections May 2020

Table of Contents

PREFACE

This document is a working manual for the Advanced Personal Computer under development at Edinburgh University Department of Computer Science. As such, it is subject to revision and change. Users are therefore advised to check that they have the latest version when working with the APM.

REPRINTED VERSION

This reprinted edition was generated by OCR from hard copy of the original document but includes additional sections concerning the Level 1 graphics board and the mouse. The original text is unaltered barring correction of some spelling errors and expansion of the external references including a new section. If using this document as a reference, ALWAYS check against the scanned originals since there WILL still be transcription errors.
[JHB 2006 and 2020]

Published by Department of Computer Science,
Edinburgh University,
JCMB, Mayfield Road,
Edinburgh, EH9 3JZ.
telephone 031-667-1081

Printed on 26 May 1983 at 11:46
Reprinted on 12 Sep 2005

1. INTRODUCTION

This document describes the Department of Computer Science's Advanced Personal Machine, (APM). It is part of a general computing environment that has been evolved over a number of years. There already is an extensive communications network in the department which has a filestore, print server, VAX 11/780, Perkin Elmer 3220 and a number of Interdatas attached to it. The services and facilities offered by these systems are described elsewhere and the APM makes extensive use of them.

The Department's Advanced Personal Machine is one version of a modular computer system designed to allow easy experimentation with both software and hardware. A major aim of this development has been to provide maximum flexibility of configuration, so that it would be possible within the overall framework to experiment with a variety of architectures and processors. Accordingly the system is constructed around general-purpose a moderate-performance bus, shareable by several processors and permitting easy expansion of memory capability. The bus supports full 32-bit data operands and 32-bit (byte) addresses, with separate data and address lines.

The basic version of the system has a single processor board utilising the Motorola 68000 microprocessor and one or more 1/2 megabyte memory boards. The processor board interfaces to a standard video terminal and to the Department's Ethernet-type network. There is no permanent local storage in the basic machine, all files being held on remote file servers accessed over the network.

A system configured with more than one active processor(bus master) also requires to have a arbitration board. It is desirable, but not essential, to have this board on the single processor configuration.

Optionally, a system may incorporate either of two levels of graphics capability. The lower level provides a passive frame-store memory which allows direct access from the ordinary processor to individual pixels; the second level puts the frame-store under the control of a programmable graphics processor.

In due course the processing capability of the system will be enhanced by the provision of user-level processor boards, incorporating a virtual memory capability, based on the latest fashionable parts coming onto the market, such as the Motorola 68010, the National Semiconductor 16032, and the Intel 80286. The present control processor boards will be retailed as input/output controllers and system monitors in this multi-process or configuration.

The existing operating system supports a single-process environaent and is an interim a development vehicle for hardware testing and evolution of the full system software. In due course the operating system will consist of a small nucleus concerned with process creation and synchronisation, together with an open-ended set of modules, selectable at will according to configuration and user needs.

There are a number of other related projects in various stages of development which include:

- the development of a Winchester disk controller, which with a large drive will enable the APM to become a network file server and with smaller disks have local file storage.

- A dynamically microprogrammable data path which will interpret abstract machine code of Meta Language (ML), an applicative language developed at Edinburgh. This scheme also provides for an experimental environment in which the support of other languages can be investigated.

- The development of a graphics processor, together with an associated frame store having a resolution of 4096 x 4096 pixels; enough for a high quality laser printer driver.

- Replacement of the current 2MHz network with a 10MHz Ethernet when integrated components become available.

2. EUCSD BUS

2.1. Introduction

The EDCSD bus is an interconnection system for computer components. Its design goals were those of simplicity, flexibility and high performance. It was assumed that the components to be connected would fall into three categories :

1. Memories - accessible from all other devices.

2. Main processors - with no I/O capability.

3. I/O processors - providing I/O facilities for the system.

The two significant features of the system are that there are multiple processors sharing common resources and that there are no simple I/O devices connected directly to the bus.

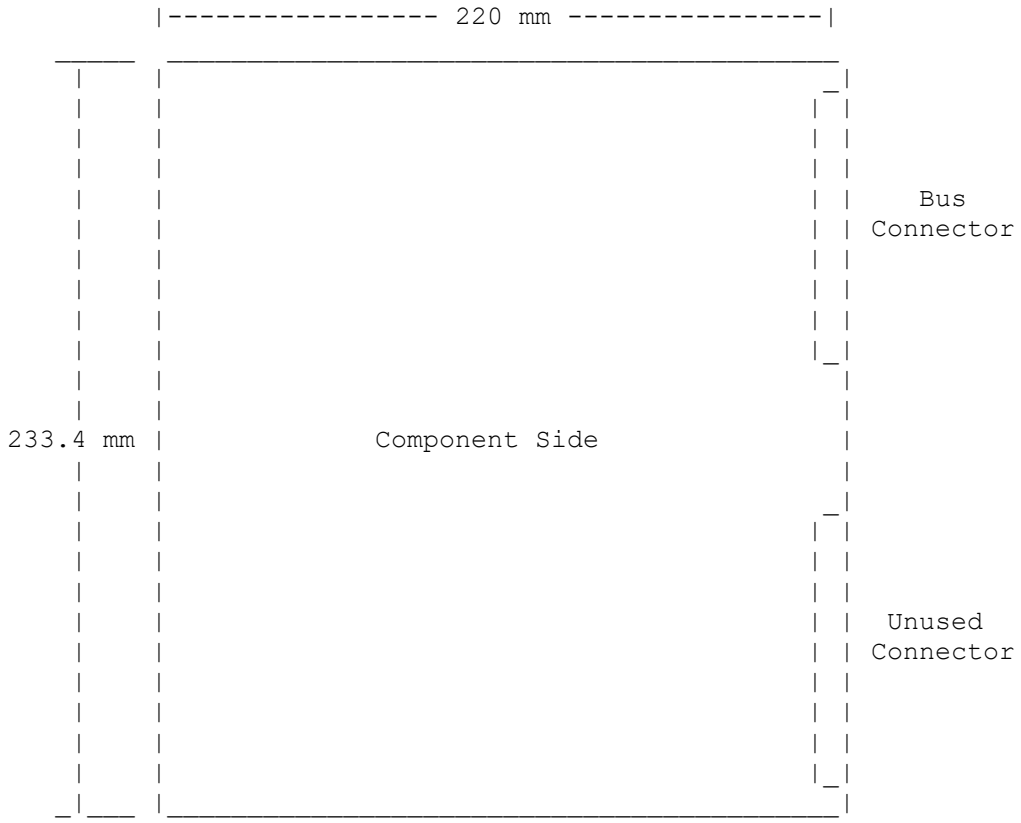
Multiple processors imply that the bus must be shared fairly among them. To achieve this, a bus controller that gives bus arbitration facilities must be provided as part of the bus structure. This is implemented as a central facility that has separate connections to each of the potential bus masters.

As there are no simple I/O devices on the bus, it is possible to avoid providing a fixed interrupt structure. These facilities are provided by control registers associated with processors. When these control registers are accessed from the bus, the effect is to cause an interrupt request tc the corresponding processor. The form and number of these registers are determined by the nature of the processor with which they are associated. As only 'intelligent’ interrupting devices are attached directly to the bus, differences between the processors can be handled in software. As a result, the only transfers on the bus comprise fully addressed exchanges of data between 'active’ masters and 'passive' slaves.

2.2. Aims of this document

2.3. Mechanical Details

All boards are Double height extended depth Eurocards (233.4mm X 220mm). The Bus is connected by a single DIN41612 C 96 way connector positioned as in the following diagram :



The unused connector position may be used freely to provide off-board or inter-board connections. It is intended that this facility should be used to communicate with other buses and systems.

2.4. Connector Pin Assignment

pin no	row a	b	c
1	GND	GND	GND
2	GND	GND	GND
3	+12V	+12V	+12V
4	-12V	-12V	-12V
5	AD2L	AD3L	AD4L
6	AD5L	AD6L	AD7L
7	AD8L	AD9L	AD10L
8	AD11L	AD12L	AD13L
9	AD14L	AD15L	AD16L
10	AD17L	AD18L	AD19L
11	AD20L	AD21L	AD22L
12	AD23L	AD24L	AD25L
13	AD26L	AD27L	AD28L
14	AD29L	AD30L	AD31L
15	CO0L	CO1L	CO2L
16	CO3L	reserved	R/WL
17	BRQiL	CBRL	BGRiL
18	RSTL	MCLK	PFLL
19	TACL	ATML	TRQL
20	ERRL	DA30L	DA31L
21	DA32L	DA33L	DA34L
22	DA35L	DA36L	DA37L
23	DA20L	DA21L	DA22L
24	DA23L	DA24L	DA25L
25	DA26L	DA27L	DA10L
26	DA11L	DA12L	DA13L
27	DA14L	DA15L	DA16L
28	DA17L	DA00L	DA01L
29	DA02L	DA03L	DA04L
30	DA05L	DA06L	DA07L
31	-5V	-5V	-5V
32	+5V	+5V	+5V

2.5. Signal Descriptions

2.5.1. Unused pins

Pin 16b is unused and reserved. Under NO circumstance should this pin be used.

2.5.2. Power supplies

GND Common signal and supply ground.

+12V Positive 12 volt supply.

-12V Negative 12 Volt supply.

+5V Positive 5 Volt supply.

-5V Negative 5 Volt supply.

2.5.3. Bus Signals

All bus signals are active low TTL level signals. Thus the Asserted (Ass) state is defined to be <=0.8V and the Negated (Neg) state to be >=2.0V.

The signal type specifies whether the signal driver is a TTL totem pole output (ttl), a TTL open collector output (o/c) or a TTL three state output (3-s).

The significance of assertion or negation of Address and Data signals on the bus is not specified as part of the bus protocol. This will be determined by mutual agreement of the masters and slaves in any given system. However all simple memory devices MUST present read data in the SAME state as it was written. By convention, address selection logic will assume TTL low on the bus to be binary 1 and TTL high on the bus to be binary 0, with AD2L the least significant bit and AD31L the most significant bit. In general, module address recognition logic will not decode more than the 16 most significant address lines.

Signal	Type	Driver	Validity	Description
BRQiL	ttl	Masters	always	Bus request from master to arbiter. Asserted whenever master wishes to use the bus and for the period while the master has control of the bus.
	NOT BUSSED			

BGRiL	ttl	Controller	always	Bus grant from arbiter to master. Asserted whenever master has requested and has been granted use of the bus.
	NOT BUSSED			

CBRL	o/c	Masters	always	Common bus request. Asserted whenever a master has requested but not yet been granted use of the bus. This indicates to the current bus master that other potential masters are requesting the bus.
RSTL	ttl	Controller	Always	System reset. Forces all devices into a known initial state.
MCLK	ttl	Controller	Always	Mains derived 100Hz clock. A train of 1ms wide low going pulses within 1ms of the zero crossing of the AC mains.
PFLL	ttl	Controller	Always	Mains supply failure. DC supplies should be stable while this signal is negated and for at least 5ms after the assertion of this signal.
ATML	o/c	Masters	always	transaction. Is negated only when TRQL is negated on the last transfer of the transaction. The transaction may consist of a single bus transfer.
TRQL	o/c	Masters	always	Asserted by the current master at the begining of an indivisible bus Asserted by the master in control of the bus when a transfer is required. This must occur only after AD2L-AD31L, R/WL, CO0L and if R/WL is asserted, DA00L-DA37L are valid and stable. It is negated only after these signals are deselected in response to TACL being asserted.

TACL	o/c	Slaves	Always	Asserted by the slave addressed in a transfer once the transfer is complete. This must occur only after ERRL and if R/WL is negated, DAOOL-DA37L are valid and stable. It is negated only after these signals are deselected in response to TRQL being negated.
			Controller	In the event of no slave asserting TACL within 2000ns of TRQL being asserted the controller will assert both TACL and ERRL to force unsuccessful termination of the requested transfer.
ERRL	o/c	Slaves	TRQL Ass TACL Ass	Slave error response. Asserted if the slave cannot succesfully complete the transfer.
		Controller		In the event of no slave asserting TACL within 2000ns of TRQL being asserted the controller will assert both ERRL and TACL to force unsuccessful termination of the
R/WL	3-s	Masters	TRQL Ass TACL Neg	Read/write line. Determines the direction of transfer on the bus. When asserted the direction is from Master to Slave. When negated is from Slave to Master.
AD2L - AD31L	3-s	Masters	TRQL Ass TACL Neg	Word address signals.
C00L - C03L	3-s	Masters	TRQL Ass TACL Neg	Byte strobes. Determines which data bus bytes are active in a transfer. Used as follows : If COOL asserted then DAOOL-DA07L are active. If C01L asserted then DA10L-DA17L are active. If C02L asserted then DA20L-DA27L are active. If C03L asserted then DA30L-DA37L are active. Otherwise the corresponding data lines are held in the high impedance state and take no part in the transfer.
DAOOL - DA07L	3-s	Masters (R/WL Ass) Slaves (R/WL Neg) COOL Ass ERRL Neg	TRQL Ass TACL Neg CO0L Ass TRQL Ass TACL Ass	byte 0 data signals.
DA10L - DA17L	3-s	Masters (R/WL Ass)	TRQL Ass TACL Neg	byte 1 data signals.

		C01L Ass	
	Slaves (R/WL Meg)	TRQL Ass TACL Ass C01L Ass ERRL Neg	
DA20L - 3-s DA27L	Masters (R/WL Ass)	TRQL Ass TACL Neg C02L Ass	byte 2 data signals.
	Slaves (R/WL Neg)	TRQL Ass TACL Ass C02L Ass ERRL Neg	
DA30L - 3-s DA37L	Masters (R/WL Ass)	TRQL Ass TACL Neg C03L Ass	byte 3 data signals.
	Slaves (R/WL Neg)	TRQL Ass TACL Ass C03L Ass ERRL Neg	

2.6. Bus Protocol

A master wishing to perform an exchange of data on the bus must obey a three level protocol. The first level deals with acquiring control of the bus. The next level deals with delimiting bus transactions, during which no information accessible from the bus can altered other than by the current master. This is included to deal with dual-ported memories which can be changed by sources other than the bus. At the third level the master requests a data transfer and acknowledges the slaves response. The protocol involved in each of these levels is given below.

2.6.1. Bus Aquistion Protocol

Step	Controller Action	Master i Action
1		Wait until BGRiL negated. Assert BRQiL. Assert CBRL.
2	Wait until it is Master i's turn to control the bus and previous master j has negated BRQjL. Negate BGRjL. Assert BGRiL.	
3		Note Assertion of BGRiL. Negate CBRL.
4		Perform one or more bus transactions according to the protocol below.
5		Negate BRQiL. Repeat Bus acquisition protocol as required.

2.6.2. Bus Transaction Protocol

Step	Master Action
1	Wait until TACL negated.
2	Assert ATML
3	Perform one or more bus transfers according to the protocol below.
4	Negate ATML
5	If CBRL is asserted then release bus otherwise retain control of bus and perform further transactions as required.

2.6.3 Bus Transfer Protocol

Step	Current Master Action	Slaves and Controller Action
1	Drive AD2L-AD31L, R/WL and COOL-C03L. If R/WL is asserted then drive DAOOL-DA37L as specified by COOL-C03L.	
2	Assert TRQL.	
3		Note TRQL asserted. Wait 25ns.
4		Decode AD2L-AD31L to see if selected. If selected then proceed otherwise wait until TRQL negated. If operation cannot be performed successfully, then assert ERRL, otherwise if R/WL is asserted then accept data on DAOOL-DA37L as specified by COOL-C03L. otherwise present data on DAOOL-DA37L as specified by COOL-C03L.
5		Assert TACL. If TACL not asserted by any slave within 2000ns of TRQL asserted then the Controller asserts TACL and ERRL.
6	Note TACL asserted. Wait 25ns.	
7	If ERRL asserted then note that transfer failed, otherwise If R/WL negated then accept data on DA00L-DA37L as specified by CO0L-C03L, Remove AD2-AD31L, R/WL and CO0L-C03L. If R/WL was asserted then remove DA00L-DA37L as specified by CO0L-C03L.	
8	Negate TRQL. If end of transaction then return to complete transaction protocol. otherwise continue.	
9		Note TRQL negated.
10		Negate ERRL. If R/WL was negated then remove DA00L-DA37L.

11 Negate TACL.

12 Wait until TACL negated.
Repeat bus transfer protocol.

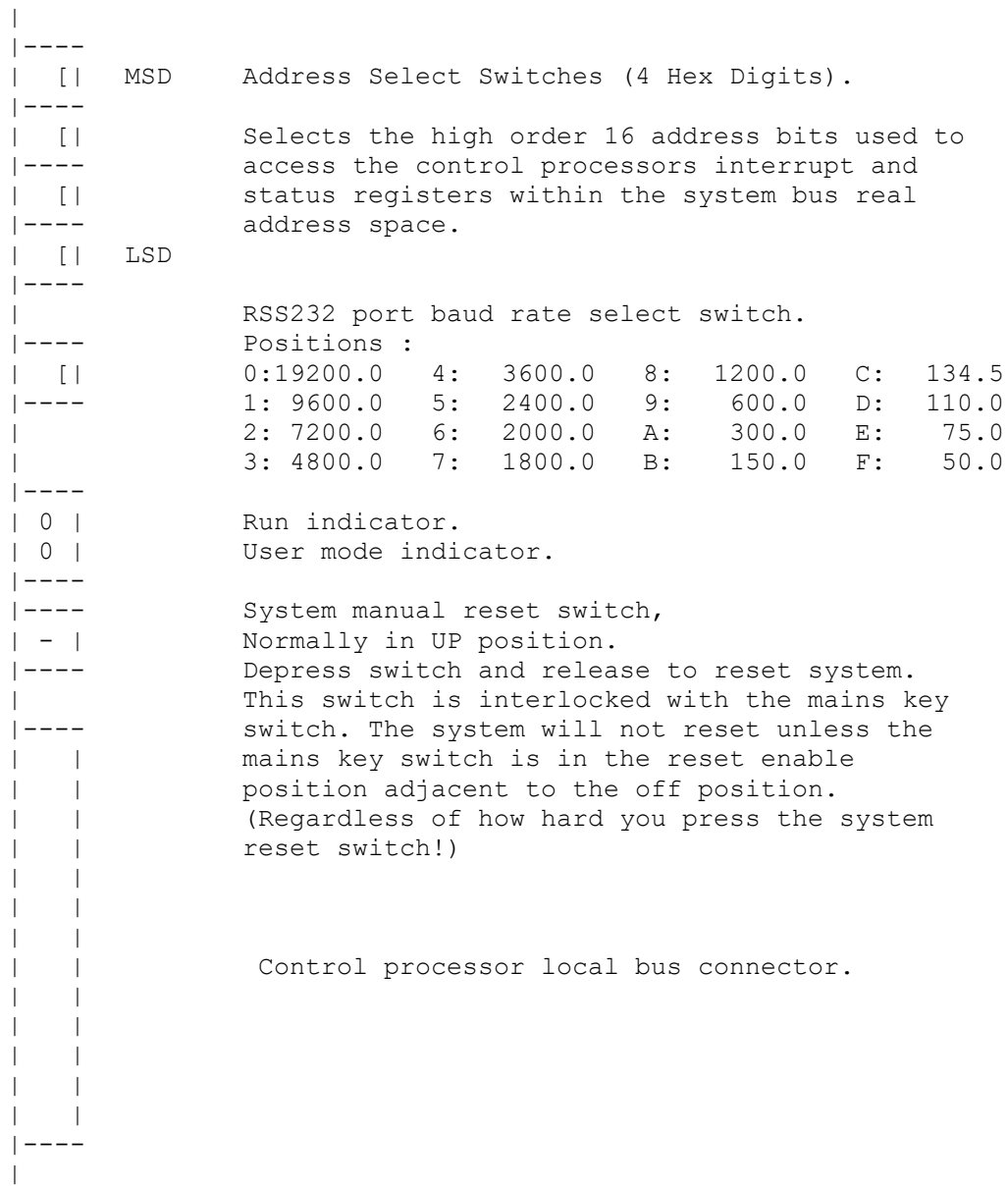
3. THE CONTROL PROCESSOR BOARD

3.1. Introduction

The control processor board (CSD134) provides a fully-programmed system control facility based on a Motorola MC68000 Clock rate micro-processor. It has a system bus interface that allows access to the complete address space of the system (EUCSD) bus. It also has a local bus through which it may access 16k of on board memory, an RS232 port and a programmable timer. This bus is also taken off board to allow expansion of the local resources. For further details of the MC68000 the reader should consult the manufacturer's documentation reference.

3.2. Switches and indicators

When viewed from the front of the system the layout of the control switches and indicators on the control processor board is as follows:



3.3. Control Processor Address Space

The address outputs from the MC68000 on the control processor are used to control access to the system bus, local bus and the on board memory and I/O devices. The 16MByte address apace of the processor is split into regions. Each region is associated with a group of devices each having a particular set of properties.

The most significant address bit (A23) is used to distinguish between access to local devices(A23=0) and the EDCSD system bus(A23=1). This is the most important division and the two cases are dealt with separately.

3.3.1. System Bus Address Mapping Registers

There are 8 registers which supply the 12 high order address bits of the system bus(AD31L-AD20L). These are indexed by address lines A22-A20 in both the system bus and the local device regions of the MC68000 address space. They can only be written. Writes to these registers take place only in the local device region. Software should maintain shadow locations in RAM to record their contents.

3.3.2. Local Devices

When accessing the local region of the address space, A19 determines whether the local bus or the address mapping registers are accessed. When accessing the mapping registers, bits A22-A20 select which register is written. Address bits A18-A1 are ignored. Only data bits 4-15 are written to these registers. The individual bytes of the address registers may be updated separately.

When accessing the local bus, A22 is used to determine the protocol to be used. Bits A18-A1 are used as the bus address lines. Bits A21-A20 are ignored. If A22=0 the MC68000 asynchronous handshake is used. A one micro-second time-out is provided to ensure all transfers complete. A M68000 bus error is generated on a local M68000 bus time-out. If A22=1 a synchronous protocol that is compatible with Motorola MC6300 family I/O devices is used. As A22 is used to gate the appropriate control signals the local bus appears as two logically distinct buses separated in time. They do, however, share both data and address lines. The UDS and LDS control signals from the MC68000 are used to determine which bytes of the local bus are active. Thus, they provide the least significant address line.

The on board memory and I/O devices are interfaced through these buses. The buses are also taken off board to allow expansion of the local address space. In particular the Ethernet local area communications network station is connected through this local bus.

3.3.2.1. Local Bus Pin Assignment
The local bus is taken off board on a DIN41612 C 96/64 way connector. This is positioned at the front of the board.

pin no	row a	c
1	GND	GND
2	GND	GND
3	unused	unused
4	unused	unused
5	unused	A1
6	A2	A3
7	A4	A5
8	A6	A7
9	A8	A9
10	A10	A11
11	A12	A13
12	A14	A15
13	A16	A17
14	A18	ADACK'
15	CLK	E
16	AS'	DTACK'
17	UDS’	LDS'
18	IRQ7’	IRQ6'
19	VMA'	R/W’
20	RST'	LOCK'
21	IRQ5'	IRQ4'
22	unused	unused
23	DO	D1
24	D2	D3
25	D4	D5
26	D6	D7
27	D8	D9
28	D10	D11
29	D12	D13
30	D14	D15
31	unused	unused
32	unused	unused

3.3.2.2. Local Bus Signal Description

signal	input output	description
A1-A18	O	MC68000 address lines A1-A18
D0-D15	I/O	MC68000 data lines D0-D15
AS’	O	MC68000 address strobe. Active only when M68000 bus in operation.
UDS'.LDS’	O	MC68000 Data strobes. Active only when M68000 bus in operation.
R/W	O	MC68000 read/write line.
VMA’	O	MC68000 valid memory address line. Active only when M6800 bus in operation.
CLK	O	MC68000 8 MHz clock.
E	O	MC68000 800KHz Enable clock.
RST'	O	MC68000 reset. Low during power-up, manual and software resets.
DTACK'	I	MC68000 data transfer acknoledge.
ADACK'	I	Address acknowledge. A TTL low on this line suppresses the local bus time-out. Used to enable slow response from slaves.
IRQ4’,IRQ5', IRQ6’,IRQ7'	I	Interrupt request lines. TTL low signals on these lines cause an auto-vectored interrupt at the corresponding priority level. IRQ7’ is a non maskable interrupt.
LOCK'	I	System bus lock. A TTL low on this line will cause the control processor to gain and retain control of the system bus. For use by diagnostic hardware.

3.3.2.3. Local Bus Timing

The timing of the signals is generally the same as an 8MHz MC68000. However, the bus is buffered and delays of approximately 20ns should be taken into account when analysing timing requirements.

ADACK' has similar timing characteristics to DTACK’, and must be asserted within 800ns of AS’ being asserted in order- to suppress the local sus timeout.

LOCK' is synchronised internally and there are few constraints on its timing.

3.3.3. System Bus

When A23=1 the system EUCSD bus is accessed. Address lines A22-A20 determine which address mapping register is used to provide the high order 12 bits of the system bus address(AD31L-AD20L). Address lines A19-A2 form the low order 18 bits of the address.

A1 is used to determine which pair of bytes is accessed on the system bus. When A1=0 bytes DA0XL and DA1XL are active. When A1 = 1 byte DA2XL and DA3ZL are active. Note that it is not possible to access bytes within both of these pairs simultaneously. UDS and LDS are used to determine which bytes within the pairs are accessed. UDS controls DA0XL and DA2ZL while LDS controls DA1XL and DA3XL. Thus to the MC68000 byte DA0XL has an address with two least significant bits on 0, DA1XL is accessed with low order address of 1 and so on.

3.3.4. Control Processor Address Space Summary

A23 A22 A21 A20 A19 A18-A2 A1 (A0)

```
 0 | 0 |---?---| 0 |----A----| 0  M68000 local space A high byte
 0 | 0 |---?---| 0 |----A----| 1  M68000 local space A low byte
 0 | 1 |---?---| 0 |----A----| 0  M6800 local space A high byte
 0 | 1 |---?---| 0 |----A----| 1  M6800 local space A low byte
 0 |-----M-----| 1 |----?----| 0  Map reg M high byte(write only)
 0 |-----M-----| 1 |----?----| 1  Map reg M  low byte(write only)
 1 |-----M-----|-----A----| 0  0  System bus (map reg M)A  DAX0L
 1 |-----M-----|-----A----| 0  1  System bus (map reg M)A  DAX1L
 1 |-----M-----|-----A----| 1  0  System bus (map reg M)A  DAX2L
 1 |-----M-----|-----A----| 1  1  System bus (map reg M)A  DAX3L
```

3.4. On board Facilities

The control processor board contains some memory and a number of I/O devices. These appear in the local address space of the MC68000 control processor.

3.4.1. Memory

The on board memory consists of 16K bytes organised as 8K of 16 bit words. It is link selectable as EPROM(2716) or RAM in 4K byte blocks. This occupies locations 00000-04000 of the local M68000 bus. Normally the first block is bootstrap EPROM and the remainder is RAM.

3.4.2. I/O Devices

The on board I/O devices consist of a RS232 port, a programmable counter timer and a mains key switch position sense register. They are located at addresses 00000-001FF of the local M6800 bus.

3.4.2.1. The Key Switch Register

Located at address 00031 of the local M6800 bus this is a simple read only register. The 4 low-order bits of this byte reflect the position of the mains switch. The coding is as follows:

Switch Position		Code
0	(furthest anti-clockwise)	Power off.
1	(Manual reset enable)	0
2		1
3		2
4		4
5	(furthest clockwise)	8

Bits 4-7 of the bus are not driven when this register is read. These bits may contain unpredictable data. Care should be taken to mask off these bits when using the contents of the register.

3.4.2.2. The RS232 Port

This is a Motorola MC6850 ACIA. For programming details see the appropriate data sheet. It appears as two low order byte locations within the M6800 bus. Both have a different meaning depending on whether it is being read or written. The command (write-only) and status (read-only) register is at 000C1. The data registers are at location 000C3. The ACIA interrupt output is connected to interrupt level 5 of the local bus and hence uses auto-vector 5 (longword at address 000074 of the local M68000 bus). Both the transmit and receive data rates are determined by the setting of a single hex coded switch visible at the front of the board. For details of their coding refer to the section on switches and indicators.

3.4.2.3. The Programmable Timer

This is a Motorola MC6840 PTM. For programming details see the appropriate data sheet. It appears as 8 high order byte locations within the M6800 bus. A? with the ACIA, each register has a different meaning depending on whetfcar it is being read or written. Their addresses within the M6800 address space are as follows:

Address	Read	Write
00100	Undefined	Control Register 1 or 3
00102	Interrupt Status	Control Register 2
00104	Counter 1 high byte	Constant Register 1 high byte
00106	Counter 1 low byte	Constant Register 1 low byte
00108	Counter 2 high byte	Constant Register 2 high byte
0010A	Counter 2 low byte	Constant Register 2 low byte
001OC	Counter 3 high byte	Constant Register 3 high byte
001OE	Counter 3 low byte	Constant Register 3 low byte

The PTM interrupt is connected to interrupt level 6 of the local bus and hence uses auto-vector 6 (longword at address 000078 of the local M68000 bus). The internal clock oscillates at 800KHz. Counter input 1 is derived from the system bus 100Hz mains clock. Counter input 2 is derived from counter 3's output. Counter input 3 is derived from counter 1fs output. This allows cascading of the counter channels. The gate inputs to all channels are permanently enabled.

3.5. Control Processor Address Space Index

This is a complete guide to the devices within the M68000 address space.

Addresses	Device
000000——000FFF	Local bootstrap EPROM
001000——003FFF	Local RAM (switchable to EPROM)
004000——07FFFB	Unused (available for local M68000 bus expansion)
07FFFC——07FFFF	Ethernet Station Interface (if fitted)
080000——0FFFFF	Address map register 0
100000——17FFFF	Same as 000000-7FFFFF
180000——1FFFFF	Address map register 1
200000——27FFFF	Same as 000000-7FFFFF
280000——2FFFFF	Address map register 2
300000——37FFFF	Same as 000000-7FFFFF
380000——3FFFFF	Address map register 3
400000——400030	Unused
400030(2)40003E	Unused
400031(2)40003F	Mains switch register (low byte)
400040——4000BF	Unsafe to access
4000CO(2)4000CE	Unused
4000C1(2)4000CF	ACIA (low bytes)
4000DO——4000FF	Unsafe to access
400100(2)40010E	PTM (high bytes)
400101(2)40010F	Unused
400110——1001FF	Unsafe to access
400200——47FFFF	Unused (available for local M6800 bus expainsion)
480000——4FFFFF	Address map register 4
500000——57FFFF	Same as 400000-47FFFF
580000——5FFFFF	address map register 5
600000——67FFFF	Same as 400000-47FFFF
680000——6FFFFF	Address map register 6
700000——77FFFF	Same as 400000-47FFFF
780000——7FFFFF	Address map register 7
800000——8FFFFF	System Bus RRROOOOO-RRRFFFFF, RRR = Map Reg 0
900000——9FFFFF	System Bus RRROOOOO-RRRFFFFF, RRR = Map Reg 1
AOOOOO——AFFFFF	System Bus RRROOOOO-RRRFFFFF, RHR = Map Reg 2
BOOOOO——BFFFFF	System Bus RRROOOOO-RRRFFFFF, RRR = Map Reg 3
COOOOO——CFFFFF	System Bus RRROOOOO-RRRFFFFF, RRR = Map Reg 4
DOOOOO——DFFFFF	System Bus RRROOOOO-RRRFFFFF, RRR = Map Reg 5
EOOOOO——EFFFFF	System Bus RRROOOOO-RRRFFFFF, RRR = Map Reg 6
FOOOOO——FFFFFF	System Bus RRROOOOO-RRRFFFFF, RRR = Map Reg 7

3.6. System Bus Control and Status Registers

The control processor provides a number of registers that can be accessed as slave devices from the system EUCSD bus. These enable the processor to be interrupted by any master on the system bus. They also provide information about the current state of the control processor.

The locations they occupy within the system bus address space are determined by 4 hex coded switches that are visible from the front of the board. These specify the high order address bits (AD16L-AD31L) used to access the registers.

The registers are 16 bits wide and connected to bytes DA2ZL and DA3IL of the system bus. Bytes DA0XL and DA1XL are unconnected and access to these bytes alone does not cause the slave logic to acknowledge an access. The two bytes are individually accessible. The DA2XL byte is common to each of the registers and is read only. The 8 distinct DA3XL bytes are selected by address lines AD15L-AD13L. Note that this allows fairly coarse segmentation to provide controlled access to the individual registers. At present the status bits are active high, but may be made active low by a trivial hardware modification.

3.6.1. Control and Status Register Address Map
To summarise the registers appear in the system address space as follows:

DA2XL byte

SSSS0002(4)SSSSFFFE	Combined interrupt state register (read only)

DA3XL bytes

SSSS0003(4)SSSS1FFF (read only)	Processor State register
SSSS2003(4)SSSS3FFF	Level 1 interrupt register
SSSS4003(4)SSSS5FFF	Level 2 interrupt register
SSSS6003(4)SSSS7FFF	Level 3 interrupt register
SSSS8003(4}SSSS9FFF	Level 4 interrupt register
SSSSA003(4}SSSSBFFF	Level 5 interrupt register
SSSSC003(4)SSSSDFFF	Level 6 interrupt register
SSSSE003(4)SSSSFFFF	Level 7 interrupt Register

(SSSS is the setting if the address select switches)

Note that the binary encoding of bits AD15L-AD13L give the level of the interrupt register.

3.6.2. Register Bit Encodings
The meaning of each of the register bits when read are as follows:

3.6.2.1. Combined Interrupt State Register

bit	
7	Priority level 7 interrupt pending
6	Priority level 6 interrupt pending
5	Priority level 5 interrupt pending
4	Priority level 4 interrupt pending
3	Priority level 3 interrupt pending
2	Priority level 2 interrupt pending
1	Priority level 1 interrupt pending
0	Always 0

3.6.2.2. Processor State Register

bit	
7	Function code FC2 for last cycle
6	Function code FC1 for last cycle
5	Function code FC0 for last cycle
4	Always 0
3	Processor halted after fatal bus error.
2	Always 0
1	Always 1
0	Always 1

3.6.2.3. Interrupt Registers

bit	
7	Interrupt pending at corresponding leve
0-6	Always 0

3.6.3. Requesting Interrupts

When any data is written to an Interrupt register the data Is discarded and an Interrupt at the corresponding M63000 priority level Is requested. At the same time Bit 7 of the interrupt register is set, together with the corresponding bit in the combined interrupt state register. Once the processor has acknowledged an interrupt at that level the interrupt request is removed, it the same time bit 7 of the interrupt register is cleared, together with the corresponding bit in the combined interrupt state register. During the interrupt acknowledge the processor is forced into the auto-vector mode of operation.

Note that care must be taken to ensure that system bus interrupts are not lost. In general, this will require that details of the interrupt request are stored in shared system memory. These details can then be examined by interrupt handling software on the control processor. This is particularly true for interrupt levels 4-7 which may also be generated by I/O devices attached to the control processor's local bus. The situation requires even more care when there is more than one source of interrupts from the system bus. Steps must be taken to ensure mutual exclusion on the Interrupt information held in shared memory. This might consist of a semaphore that is locked prior to storing the interrupt details and before issuing the interrupt request. The semaphore is subsequently released by the control processor on completion of the interrupt service routine.

4. MEMORY SYSTEMS

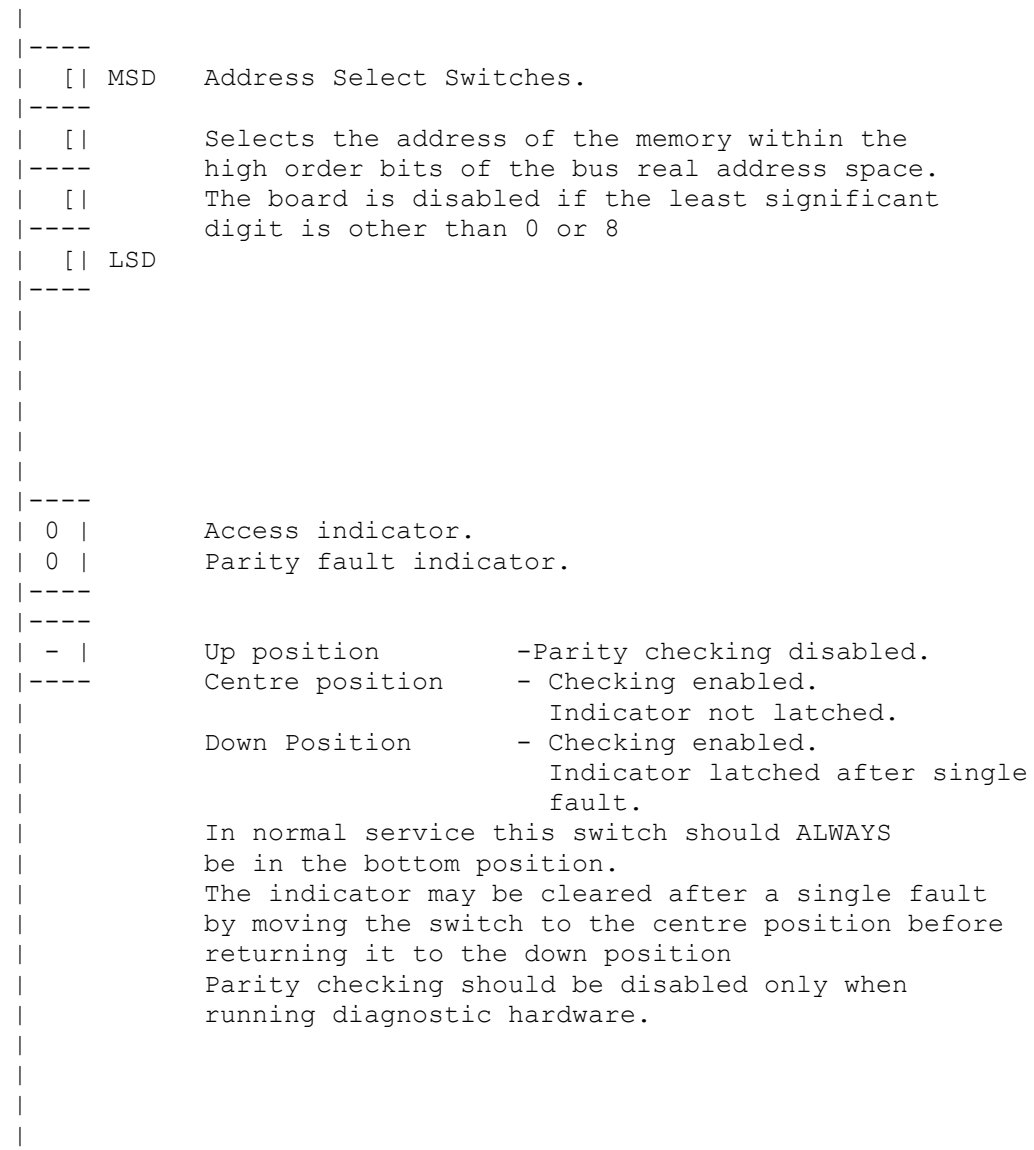
4.1. Introduction

4.2. The Memory Board (CSD136)

The standard memory board provides 512K bytes of parity checked volatile memory. The memory is based on 64K dynamic memory chips. To minimise the effect of cycle times and refresh overheads the board is internally organised as two interleaved banks. Approximate performance is as follows:

Average access time (read and write)	:<230 ns
Average cycle time	:<260 ns
Worst case access time	:<550 ns

Switches and indicators.



5. CONSTRUCTION

5.1. The Cabinet (CSD144)

The APM cabinet is an attractive standard 19 inch equipment case. It contains a 19 slot card cage to the EDCSD bus specification. Power is supplied by a 400 watt switched mode unit. Sufficient cooling for the boards and power supplies is provided by external quiet fans.

A Perspex door at the front of the cabinet gives access to the card cage and allows board indicator lights to be seen at all times. Mains switching is on a 6 position key switch. The position of the key switch may be sensed by software in the control processor. This switch also interlocks with the system manual reset switch on the control processor. The backplane may be accessed by a door at the back which also carries the power supply. A fixed panel at the rear of the cabinet carries external connectors such as the RS232 port and local area network interface. Specifications :

cabinet size : ? x ? x ?

power supply maximum ratings :	+5V	45A
	+12V	?A
	+24V	?A
	-5V	?A
	-12V	?A

SUBJECT TO AN OVERALL RATING OF 400 WATTS TOTAL

Although +24V is not available on the backplane direct connections may be made to the power supply for disc drive stepper motors.

I. Vendors and Services

6. LOCAL AREA COMMUNICATIONS

6.1. Introduction

This chapter describes the APM component of the Edinburgh University Computer Science Department's local area network. The network is based upon a carrier sense, multi-access broadcast discipline with collision detection which was first developed within the Xerox PARC establishment. There are some major differences between the Department's Ethernet and the Xerox model. Essentially, the EU Ethernet controllers have their own processing and memory resources and can therefore offer a range of network services which absorb the overheads of protocol handling that would otherwise fall upon the network's hosts. It follows that the scope of protocol design is more extensive, incorporating higher layers of protocol than the existing Xerox proposals. An obvious consequence of this is that network stations are comprehensive independent entities likely to appear in many devices. Indeed, versions of network stations were designed and constructed before the APM existed. The issues for the APM are how these stations are incorporated into the basic system framework and how they are used. A full discussion of the Edinburgh Ethernet is presented elsewhere [reference 3.13].

6.2. The Ether net Station Interface (07FFFC-07FFFF)

The station for the circuit (2mhz) ethernet is interfaced to the control processor through a full duplex bi-directional 9-bit wide character stream. Associated with each transfer direction is a single 8-bit character buffer register, plus various control flags which determine whether the register is full or empty and whether it contains data or a control character. When the ether station interface interrupts, it uses auto-vector 4 (longword at address 000070).

The interface appears to software as six registers (command register, status register, 2 Data Character Registers, 2 Control Character Registers). The command and status registers share the same address (and are distinguished by read/write), and both appear at two locations in the address space. The RX and TX data registers share an address, and so do the RX and TX control registers, as shown below. This mapping makes it possible to read either the data or the control register using a word, rather than byte, access, transferring the data in the low-order byte and command/status in the high-order byte.

Reference:

3.13 Edinburgh Local Area Network (W. Enos, I. Hansen, R. Thonnes, undated)

```
07FFFC    Command and Status
07FFFD    Data Character
07FFFE    Command and Status
07FFFF    Control Character
```

Status Register bits have the following meaning:

```
Bit 7:    Interrupting
Bit 6:    Being Reset
Bits 5,4: Unused (0 when read)
Bit 3:    Transmit Register Empty
Bit 2:    Received Control Character
Bit 1:    Received Data Character
Bit 0:    Received Control or Data Character
```

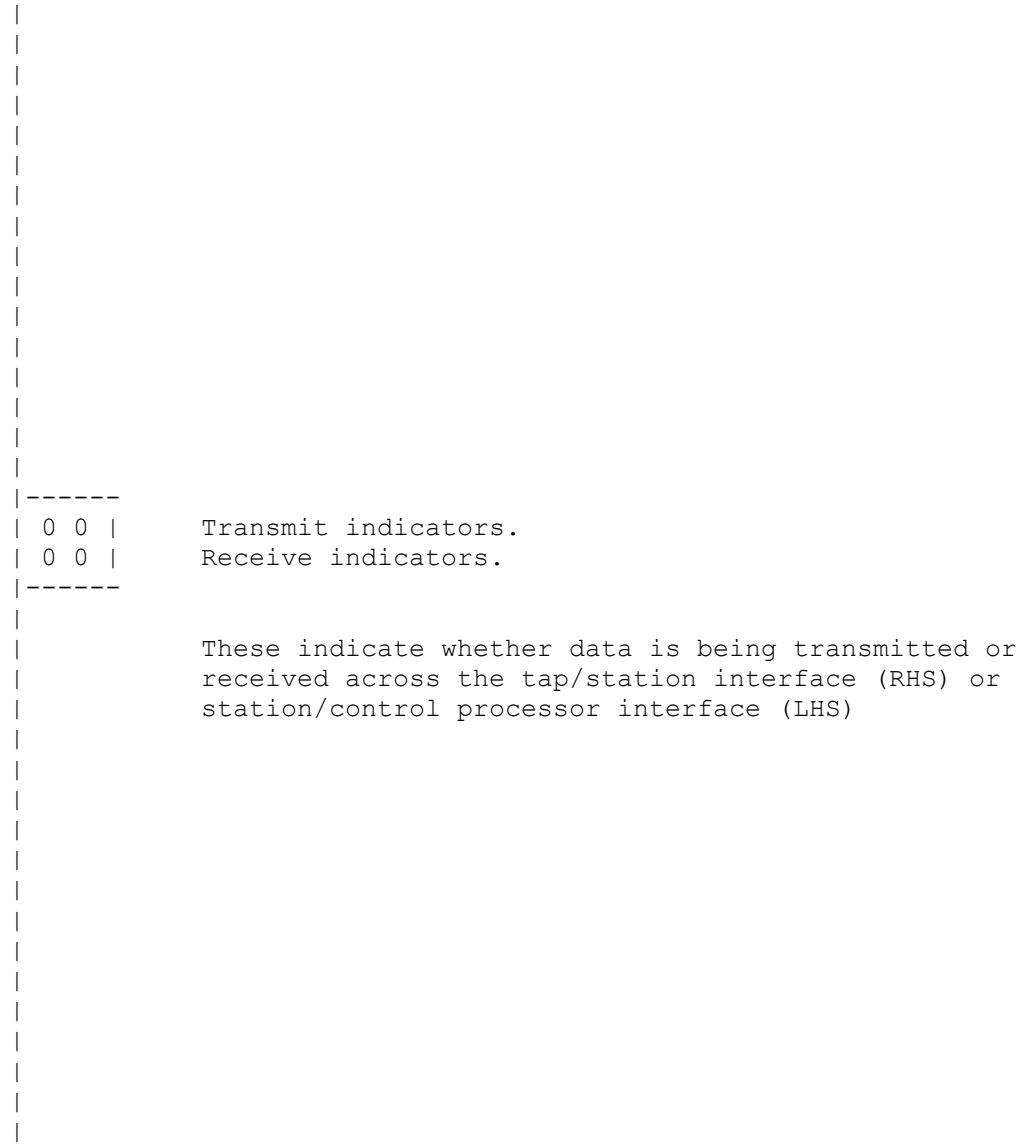
Command Register bits have the following meaning:

```
Bit 7:    Unused
Bit 6:    Reset the Station
Bits 5,4: Unused
Bit 3:    Interrupt on Transmitter Empty
Bit 2:    Interrupt on Received Control
Bit 1:    Interrupt on Received Data
Bit 0:    Unused
```

Interface operation is straightforward on the transmitter side. The software writes its characters either into the data or the control register, provided the TSE bit in the status register is set. Receiver operation is such that a read from the data register will clear the RDC bit, but not the RCC bit. It is necessary to read from the control register to clear the RCC bit. Likewise, a read from the control register will not clear the RDC bit. RCDC will remain set as long as either of RCC or RDC is set.

6.3. The 2 MHz Ethernet Board (CSD---)

Switches and indicators.



7. SYSTEM SOFTWARE

This chapter describes the facilities provided by the interim development operating system. In line with earlier practice, very little is built into the system itself, the great majority of the commands being implemented by calling individual programs.

7.1. Loading the System

To load the system turn the power-on key clockwise to the first position. It already switched on, the system may be reset and reloaded at any time by (gently!) pressing the small spring-loaded switch located on the control processor board. This causes the operating system software to be loaded over the network from the Filestore. NB: The memory boards have similar-looking switches and should not be confused with the control processor board. The latter can be distinguished by the ribbon cable connecting it to its neighbour.

7.1.1. Communicating with the System

The video terminal is the basic means of communicating with the operating system. After initial loading, the system announces itself on the screen and outputs a curly bracket, inviting the user to enter a command at the keyboard. Usually the first command that a user will give in a session will be the L command to log on to a remote filestore. However, it is not essential to do so in order to use the system. Every time a command is issued by the user, the system executes it and then prompts for another command.

7.2. Command Language

The simplest commands consist of a single word (a verb or abbreviation for a verb); others consist of a number of words, separated by prescribed punctuation symbols. Commands are always terminated by typing RETURN, and until this is done, they can be corrected or revoked at will.

In general, the effect of typing a command is that the command verb is used to locate a file which implements the command.

Examples of some of the simple commands which require no additional information beside the command verb itself are KELP (to elicit help information) and TOD (to find out the time of day)

Examples of more elaborate commands (discussed below) are:

```
COMPARE OLDFILE,NEWFILE/DUMPFILE
T TESTPROG/ CURRPROG
```

7.2.1. Parameters

As the more elaborate examples illustrate, it is, in general, necessary fax- the user to be able to specify not only what operation is to be performed but also what data is to be used, where the results are to go, and what options are to be selected. This is done by supplying additional parameter words following the command verb.

When a parameter is used to denote a data source or destination, it is termed a stream-name. A stream-name may take either of two forms: a file-name or a device-name. When parameters are present, they are separated from the command word by at least one space. Some conventions governing the use of punctuation symbols in the rest of the command are illustrated in the following examples:

```
T TESTPROG / CDRRPROG
```

Here the command word is T, indicating the Transfer operation,which copies data from any file or device to any other file or device. The source (TESTPROG) is specified first and then the destination (CDRRPROG) following an oblique stroke.

```
COMPARE OLDFILE , NEWFILE / DUMPFILE.
```

In this case, the command word is COMPARE, indicating file comparison. For this operation the two files to be compared (OLDFILE and NEWFILE) have to be specified as inputs; these file names are separated by a comma. Then the output file to which a list of discrepancies is to be sent (DUMPFILE) is specified after the oblique stroke.

```
COMPARE OLDFILE , NEWFILE
```

In many cases, there are default parameters for a command which are understood if the user does not supply a parameter explicitly. In the case of COMPARE, the default for output is the terminal.

```
IMP CURRPHOG -NOASS –LIST
```

In this example invoking the IMP compiler, the source program is specified as the first parameter and then two options are selected: NO ASS to suppress unassigned-variable checking and LIST to cause a listing to be produced.

In this example, the dash (minus) is used to mark what follows as an option specification.

```
IMP -NOASS -LIST CURRPROG
```

This command is identical in effect to the previous one.

the point that the placing of option specifiers is open, unlike the placing of other parameters.

7.2.2. Punctuation

General conventions governing the format of most commands are as follows

(a) output stream-names are separated from input stream-names by an oblique-stroke;

(b) within a group of either input or output stream-names the separator is a comma;

(c) keyword parameters are preceded by a dash (minus), and may be followed by an equals-sign and a value;

(d) the complete command is terminated by RETURN;

(e) spaces are optional before and after the punctuation symbols mentioned above.

7.3. Input and Output

7.3.1. Files

File-names denote files held on a remote file server; file naming conventions are those imposed by the file server. At present, the only such server is the original central Departmental Filestore introduced in 1976. Full information about the management of files is contained in the Filestore manual [reference 4.1].

The naming conventions are as follows. The full form of a filename consists of an owner-name, followed by a colon, followed by a file-identifier proper. In many cases the owner-name is implicit, so that only the file-identifier proper need be typed. The latter consists or up to 12 characters, of which the first must be a letter or a dollar-sign, and the remainder may be any combination of letters, digits and dots. Conventionally a dot is used to separate a file-name extension (like LIS or OBJ) from the file- name proper. A file-identifier starting with a dollar-sign denotes a temporary file, which is automatically deleted when the user logs off. The special file name LP: (that is, owner- name LP followed simply by a colon) is used to denote the line-printer connected to the Filestore.

7.3.2. Devices

Device-cases are distinguished by starting with a colon. On the basic machine, the only device is the video terminal, which is denoted by :T (or simply a colon by itself). Also reckoned as a device is the null data stream (denoted by :N), which it is sometimes convenient to specify in place of a file-name; for input, this implies no data and for output, results to be discarded.

7.4. Command verbs

The interpretation of a command verb proceeds along the following lines:

If the word is a defined symbol (see below), it is replaced by its definition and the resulting command is interpreted;

Otherwise, a file with the same name as the command verb and extension H3B is searched for. If a directory is included in the name (for example, APM:PROG), the search is confined to the directory indicated. Otherwise, the search is made, first, in the user's selected directory and, failing that, in the system directory FMAC. If so such file can be found, an error report is made.

Reference:

4.1 “The Filestore” (H. Dewar, V. Eachus, K. Humphry, P. McLellan, Dept. of Computer Science Oct 1977, revised Sep 1981 (draft))

The located file may be:

a Motorola 68000 object program to be executed.
a file of commands to be obeyed

7.4.1. Object files

These are files containing directly executable programs, generated by one of the assemblers or compilers available on the system.

If the first character of an object file is 'S', the file is assumed to contain object code in Motorola format. The code is loaded into free memory, with location- counter addresses interpreted as relative to the start of free memory.

Otherwise an object file, is assumed to be a binary executable image, i.e. a contiguous chunk of position-independent code. [At present the first character should be binary 254 (16_FE)].

Binary and Motorola format code files, once loaded, are entered by means of a JSH to the last longword of the file, which will normally contain a word-displacement jump instruction of some sort.

At present, by default, the rest of the command line is net read by the command interpreter. When the loaded program reads from stream 0, it will get the rest of the command line and is free to interpret it in whichever way it chooses.

7.4.2. Command files

A command file consists of a sequence of commands and data exactly as they would be typed at the terminal, except that the first character in the file must be a left curly bracket ('{'). The commands are successively executed until either the end of the file is reached or one of the commands fails. The command file is closed after the last (and possibly only) command in the file is executed. One command file may call another, but presently only on a chained (not nested) basis.

7.4.3. Symbol Dictionary

The system maintains a dictionary of names or 'symbols' which have been defined to stand for other names or partial command sequences. A number of system definitions are established when the system is first loaded. The user may add others as required by means of symbol definition commands. The form of this type of command is:

```
<newname> = <oldname>
```

For example:

```
GO = UTIL:TESTPROG1
```

When a program is loaded for which there is an abbreviation in the command dictionary, the system remembers where it has been loaded. The space occupied by the binary image is not released when the program finishes. This saves time when the same program is subsequently required again, which is especially useful for large frequently used programs like compilers.

When symbols are looked up in the command dictionary, approximate (leading substring) matches are taken to be sufficient. For example, given the abbreviation NIMP=HMD:NEWCOMP the command NIMP, but also the commands NI or Just N, may be used to refer to HMD:NEWCOMP, provided there has not been a more recent symbol definition starting with N or HI.

Because the rest of the command line is normally read from the control stream directly by the loaded program, it is not possible, as on VAX/YMS, to include preferred default options in the symbol definition (like IMP=I:C-DIAG-LIST), as there is no way the command interpreter can inject these into the control stream. It is, however, possible to include flag characters in a definition (e.g. COMPARE=FMAC:COMPARE!).

7.4.4. Flag Characters

A command verb may be terminated by one or more flag characters, which cause the command to be interpreted slightly differently.

Flag character '_' suppresses translation of the command verb via the command dictionary.

Flag character '!' instructs the command interpreter to read the rest of the command line, and to interpret it as a set of input-output stream definitions as noted above (Up to 3 input file names separated by commas, optionally followed by a '/' and up to 3 output file names separated by commas).

Flag character ‘?' causes the loaded program to be entered in trace-mode for single step execution under control of the software front panel (see below).

7.5. Terminals

On the basic configuration, the main device for interaction with the system is a standard VDU. In fact, it figures as two devices: a keyboard for input, and a display screen for output. Terminals are always driven in full duplex mode, which means that information typed at the keyboard does not automatically appear on the screen; it does so only if the software dealing with keyboard input (the terminal handler) echoes it. Most of the time, characters are echoed as they are typed in, but in some cases echoing is suppressed or other characters substituted.

The general principle for keyboard input is that a sequence of printing characters are not acted on until the RETURN key (or other terminator key) is pressed, although some highly interactive programs override this provision. All the printing characters of the ASCII character set may be entered in the normal way, but the control characters are subject to special interpretation, as described below. Again, some programs override the conventional system interpretation. Control characters are generated by pressing a normal key while holding down the CTRL key. For example, TAB (though usually provided as a single key) can also be generated by CTRL together with I; this is indicated below as ^I.

Many programs issue a prompt message when input is required, as a guide to the user. It is usually permitted to type ahead of input requests, and it may be convenient to do so when a lengthy operation is in progress. Information typed ahead is not echoed until the running program tries to read it. It is discarded in the event of a total failure in the execution of a previously issued command.

7.5.1. Terminal Control Characters

DEL Erase the last extant character typed on the current line.

BS or ^X Erase all characters typed on the current line. RETURN Terminate the current line.

^S Set auto-freeze mode.
When enabled, output to the terminal is halted every time the screen becomes full and remains in this frozen state until something is typed on the keyboard; LF 'unfreezes' the display for one line, RETURN or BS unfreeze the display for 1 page, and ^Q changes the mode to 'continuous scrolling'.

^Q Quit auto-freeze mode.

^Y Abandon the current activity, and return to command level.

^D or ^Z Terminate input from the terminal,
creating an 'end of file' condition.

^P Pass the next character typed as data to the program currently running. For example, ^P^Y allows ^Y to be input without stopping the program.

^T Escape to trace-mode
(See below: Software Front Panel).

7.6. Summary of Individual Commands

Logging on or off:
```
L <user>
L
```

This command is used to log on to the Filestore. The purpose of logging on is to establish the default directory to be used in referencing files, and to demonstrate authority to use those files by citing the appropriate password. The user is prompted for the password (union is not echoed). If the command file LOGIN.MDB exists, the commands in it are obeyed before the system accepts further commands from the terminal. If another user is already logged on at this machine, the previous user is automatically logged off.

Since this command always causes any existing user to be logged off, it can be used without any parameter simply to log the current user off.

Transfer:
```
T <in1>,<in2>,<in3>/<out>
```

The Transfer command is used to copy information from one or more (up to three) source streams to a destination stream. If more than one source stream is specified, the data is concatenated in the order gives. The defaults for <in1> and <out> are the terminal.

Print:
```
P <file>,<file>,...
```

The specified files are sent to the printer spooler. Any copying operations this involves are done remotely and do not impose a load on the local processor or on the network.

Edit command:
```
E <old>/<new>
E :N/<new>
E <old>
E <old>,<secondary>/<new>
E <old>/:N
```

The Edit command is used to invoke the VECCE context editor, which is fully described in a separate manual (the revised screen-oriented specification).

1. The first form of command is used to edit an existing file to a new file.
2. The second form is used to create a file from scratch.
3. The third form is used to edit an existing file. In this case, the new file is given the same name as the old, (but the original file is not deleted until the editing has been finished successfully).
4. The fourth form is used to specify a secondary input file.

5. The last form is used to examine a file without altering it.

File enquiry: F <dir>

The file enquiry command F provides the facility to list the names of the files in a given directory.

The command FILES provides fuller information about each file, including the permissions and date of creation.

The current default directory is used if <dir> is omitted.

Directory Operations: D <dir>

This command displays the names of the files in the specified directory and allows the user to ask for any of them to be Shown, Renamed, Deleted, or Permitted. It is self-documenting. Type H within it for assistance.

The current default directory is used if <dir> is omitted.

Change Directory: SET <dir>

The SET command is used to change to another default directory, to be used in subsequent file references. If <dir> is omitted, the default reverts to that established at log-on.

Quote Password: QUOTE

The QUOTE command is used to quote a password in order to demonstrate authority additional to that established at log-on, so that protected files in directories owned by someone else may be accessed.

Delete file(s): DELETE <file>,<file>,...

The DELETE command causes -the specified file(s) to be destroyed. Ths user must have owner-authority over the directory involved in order to delete files in it.

Rename file: RENAME <old>/<new>

The RENAME command is used to rename a file. The user must have owner authority with respect to the directory containing the file in order to rename it. Other attributes of the file, like date of creation and permissions, are unaffected by renaming.

It is not possible to alter the directory part of a file-specification. The parameter <old> may contain a directory name, as in LIB:TEST2f but <new> may not.

Change file protection: PERMIT <file>/<code>
or PERMIT <dir>:/code

The first form changes the protection of <file> to <code>. The second font changes the default protection associated with the directory <dir>; this applies to all files subsequently created in that directory.

The interpretation of <code> on the 1976 Filestore is as follows:

The first letter denotes the permission given to anyone with owner-authority; the second letter denotes permissions given to everyone else. These letters may be:

F : Free (i.e. Read/Write/Delete)
R : Read-only
or H : no access

The third letter, if present, affects the archive flag associated with the file and may be:

A : Archive (i.e. back this file up regularly)
or V : Vulnerable (do not back this file up).

View help information: HELP <topic>

The HELP command is used to browse through documentation relating to the commands available on the system. HELP by itself provides general information about the system and the command language. HELP followed by a topic-name like ECCE provides information about a particular topic. The system command F VIEW may be used to establish what topics may be specified.

This facility emulates the EMAS VIEW facility, which provides access to information using a Prestel-style tree-structured approach.

Type H within HELP for a summary of the operations provided.

Compare files: COMPARE <file1>,<file2>/<diff>

The program COMPARE compares two files and produces a list of the lines which differ in each file. The matching is done in such a way that short insertions are recognised without generating a lot of false mismatches. The differing lines are printed out preceded by 1 or 2 to indicate which file they are from. The differing lines are followed by the first line which matches in both files; preceded by an equals.

Produce documents LAYOUT <doc>/<file>

This command invokes the document production program LAYOUT to format a file prepared in LAYOUT source.

The program is described in a separate manual [references 5.1, 5.2].

Assemble program: M68000 <file> {options}

This invokes the M68000 assembler. The program in <file> (with assumed extension ASM) is assembled. The object code (in Motorola format) goes into a file with extension MOB, the listing into a file with extension LIS. By default, production of object code is enabled, of a listing disabled.

Compile Imp program: IMP <file> {options}

This invokes the IMP compiler for the M68000. The program in <file> (no assumed extension) is compiled, with object code (in binary image form) going to a file with extension MOB. By default, production of object code is enabled, but is suppressed if the program contains compiler-detected errors. By default, production of a listing is disabled. The main options are:

-LIST{=file}	Produce a listing (default file is LP:, so that listings are sent straight to the network printer
-DIAG	Include line number diagnostic
-TRACE	Generate code which lets the program be excuted one line at a time under control of the Software Front Panel.

Interpret S-Algol program: SR <SA-command-line>

References:

5.1 “Layout”, P. McLellan, Dept. of Computer Science Internal Report CSR-21-78, Feb 1978

5.2 “Layout 1.5 User’s Guide (provisional)” (Hamish Dewar, Dept. of Computer Science, Feb 1984)

The SR command invokes the S-Algol interpreter. The first thing on the command line is taken to be the name of an S-code file, which is loaded and interpreted.

Compile S-Algol program: SR SA:SC <file>

This invokes the S-Algol interpreter to load and interpret the S-Algol compiler, which is documented elsewhere.

Test New System: BOOT <file>

The BOOT command loads the file specified starting at location 16_1000, logs the current user off the Filestore, and jumps to the start-address specified by the pseudo-reset vector contained in the loaded file. It is used to load trial versions of new operating systems.

Convert to Binary: TOBIN <infile>/<outfile>

The standard Motorola M68000 assembler produces a form of output which, aside froa being space-inefficient, is incompatible with the ROM bootstrap loader, which only loads straight binary images. TOBIN converts from Motorola format to binary image form.

Convert to hex: HEX! <in>/<out>

The stream (usually a file) <in> is output to <out> one byte at a time, in hexadecimal.

Dissemble machine-code: DIS <from> <to> /<out>

Code in the machine address space, bewteen the specified addresses <from> and <to> (both in hex) are decoded and output to stream <out> in M68000 assembler notation. If /<out> is omitted, output goes to the terminal.

Log on to VAX: VAX

The VAX command attempts to establish a Network Terminal connection to the Department's VAX, so that the terminal can be used as a VAX terminal. This facility should be used sparingly, because the mode of connection puts a heavier load on Vax than ordinary terminals. It is not intended for general public use, but rather as

a convenience terminal for those who are working in parallel on VAX and the APMs. The VAX program terminates (i.e. returns to command level) when the VAX process to which you are connected stops (normally when you log off VAX).

Access files on VAX: VFS

The VFS (Vax File Store) command makes it possible to transfer files between VAX and the 1976 Filestore. Like the VAX command, it is only a convenience utility and should be used with care. It is self-documenting (type H in reply to the '>>' prompt).

7.7. The Software Front Panel .

This debugging tool is invoked by appending the flag-character '?' to a command verb, or by pressing ^T at the keyboard. When invoked, the SFP displays (near the top of the screen) the contents of processor registers D0 to D7, A0 to A6, the current stack pointer A7, the status register and program counter SR and PC, the line number (if an IMP program compiled with the -DIAG or -TRACE qualifiers is running), and the previous PC (if known). In response to the "Now what?" prompt, single-character commands are accepted, which are not subject to normal line-editing conventions. Unknown command characters (such as H for help) cause a list of valid commands to be displayed. These are:

S — Execute one single instruction of the running program, then re-invoke the SFP (Implemented by setting the T bit in SR).

C — Continue. Resume normal execution of the running program.

Bx — Set breakpoint at x. Execute instructions until PC=x. x is an absolute PC value and should be entered in hexadecimal immediately after the B (no leading spaces), the first non-hex character terminates the number, sets the breakpoint and resumes execution of the running program (implemented by replacing the instructions at the target PC by a TRAP TRAP 0 instruction,not by setting the T bit in SR).

R — Reset and reload the system.

N — Execute next statement or line. Only relevant to programs compiled with the -TRACE qualifier.

Ln — Execute statements until line n is reached (n is in decimal and should come immediately after the L). Only relevant to IMP programs compiled with the TRACE qualifier

X — Examine store (Spy). This issues a prompt to which the reply is TWO hex numbers (no leading spaces,separated by one non-hex character). The first is an address, the second a byte count (in the range 0 (meaning 256) to 255). The specified number of bytes (say k) are displayed on the screen. Ihree actions are then valid: Press RETURN:The next k bytes are displayed (repeatedly). Press '=': The last byte displayed is overwritten with a hex value, typed immediately following the'='. Press any of the above commands (including X

to examine a different area of store).

7.8. System Environment

7.8.1. Initial System Loading

After the system has been reset (as a result of pressing the reset button, of typing 'R' to the Software Front Panel, or of first switching on), the ROM bootstrap fetches the operating system from the filestore. The file FMAC.STS on the 1976 filestore is read, and assumed to contain a binary image of an operating system suitable for loading into memory starting at address 16_1000. Because in the M68000 the interrupt table occupies the 256 longwords (1024 bytes) of store starting at address 0, and this area is actually ROM, the ROM bootstrap contains code to re-direct all interrupts through a corresponding table of pseudo-vectors starting at address 16_1000. The first two interrupt "vectors" represent initial values for the stack pointer and program counter and, consequently, the ROM bootstrap will use the corresponding values (at 16_1000 and 16_1004) to enter the loaded operating system.

Once the basic system is loaded, it will load the command interpreter. If by this time any key on the keyboard has been struck, the system will ask for the name of an image file to load instead of the command interpreter. Otherwise it proceeds to load file FMAC:EZEC.MOB. Once this is loaded, it obeys command file FMAC:STARTUP.MOB, which contains symbol definitions for some of the utility commands detailed above.

7.8.2. Software Environment

Programs are entered at the address of the last byte loaded minus three. The last longword is usually a jump with word displacement to the actual start of the program. The area of free store available for use by the program as stack or heap or general work space is contiguous and is delimited at the low-address end by D6-256, and at the high-address end by SP. The contents of all other registers on entry to the program are undefined. Normally (except for programs which remain resident because they have symbolic abbreviations) this contiguous area of free store will be adjacent to the area into which the code was loaded.

In the development system, a number of general-purpose support routines are incorporated in the operating system itself and are called indirectly through a fixed-site table of entry points. [The M68000 does not have indirect addressing, so indirect calls are done as calls into a branch table]. A list of the routines available follows, with a brief indication of what they do. "Include"-files suitable for u-e with the assembler and IMP compiler which define the entry-points and their parameter specifications where applicable are contained in filestore files FMACS:SPECS.ASM (assembler) and FMACS:SPECS (IMP).

All these routines follow the conventions that parameters are passed in registers (values in D0 to D3, addresses in A0 to A3)» results (if any) come back in D0 or A0, all registers not conveying

results retain the values they had when the procedure was called.

It is emphasised that these functions are provided as an in-built part of the development system for convenience and that most of them will eventually be removed in favour of linked library routines. The continued availability of the low-level ether net procedures is not guaranteed in any form.

In the list which follows, the entry-point address (in hex) comes first, then the name of the procedure, then a brief description.

10C0	printsymbol	The character in D0 is sent to the current output stream.
10C4	printstring	The character string (pointed to by) A0 is sent to the current output stream.
10C6	readsymbol	The next character on the current input is read to D0.
10CC	nextsymbol	The next character on the current input stream is copied to D0 (but not read).
10D0	prompt	The character string A0 becomes the prompt string for terminal input.
10D4	testsymbol	The next character from the keyboard type-head buffer is read to D0, -1 is returned there is type-head. This operation is independent of the currently selected stream
10D8	selectinput	input stream D0 is selected (0:3).
10DC	selectoutput	output stream D0 is selected (0:3).
10E0	resetinput	The current input stream is reset.
10E4	resetoutput	The current output stream is reset.
10E8	closeinput	The current output stream is closed.
10EC	closeeoutput	The current output stream is closed.
10F0	openinput	Input stream D0 is selected, closed, and re-opened as the file with name A0. The null file, ".T" or "." denoted the terminal.
10F4	openoutput	Output stream D0 is selected, closed unsatisfactorily, and re-opened as file A0.
10F8	etheropen	Ether port D0 is opened to remote station D1>>8, port D1&15.

10FC	etherclose	Ether port D0 is closed.
1100	etherwrite	D1-byte buffer A0 is written to ether port D0.
1104	etherread	Up to D1 bytes are read from ether port D0 into buffer A0, the actual packet size is returned in D0.
1108	fcomm	Filestore command D0 with parameter A0 is sent to the filestore. The numeric response is returned in D0.
110C	fcommw	Filestore command D0 with parameter A0 and D1-byte buffer A1 is sent to the filestore. The numeric response is returned in D0.
1110	fcommr	Filestore command D0 with parameter A0 is sent to the filestore,the packet response is read into buffer A1 of size D1. The actual packet size is is returned in D0. NB: Filestore commands passed in D0 to fcomm/fcommr/fcommw are encoded as the command letter « 8 plus the user/transaction digit. If the user digit is null (as opposed to '0',), the user number of the currently logged-on user is substituted.

1114	signal	IMP event D0 is signalled. D0 is encoded as the sub-event number « 4 plus the event number. The whole value may be negated to indicate that the extra information and the message field in the the event record are to be cleared.
1118	read	A (decimal) number is read from the current input stream and returned in D0.
111C	write	D0 is written out in decimal on the current output stream, padded with with leading spaces to fill D1+1 character positions (not padded if D1<=0).
1120	mull	The product of 32-bit integer values D0 and D1 is returned in D0
1124	divl	The quotient of the division of 32-bit D0 by 32-bit D1 is returned in D0, the remainder in D1.
1128	scompu	(IMP perm:String compare A)
112C	scomp	(IMP perm: String compare B)
1130	cputime	Return (in D0) the (integer) number of milliseconds of elapsed time since the system was loaded.
1134	arraydef	(IMP perm: Define array dimension)
1138	arrayget	(IMP perm: Claim array space)
113C	arrayref	(IMP perm: Reference array element)
1140	settermmode	Set terminal mode as selected by bits in D0: 1: Suppress echo 2: Suppress terminator echo 4: Suppress line-buffering 8: Suppress auto-freeze
1144	intpcwer	Raise integer D0 to the integer power D1, result in D0.

1148	rplus	Floating-point D0 = D0 + D1
114C	rminus	Floating-point D0 = D0 - D1
1150	rmult	Floating-point D0 = D0 * D1
1154	rdiv	Floating-point D0 = D0 / D1
1158	rpower	Floating-point D0=D0 \\ Integer D1
115C	rnegate	Floating-point D0 = -D0
1160	float	Floating-point D0 = Integer D0
1164	fracpt	Floating-point D0 = fracpt(D0)
1168	intpt	Integer D0=intpt (Floating-point D0)
116C	sqrt	Floating-point D0 = sqrt (D0)
1170	line	Graphics Mk III: draw line from (d0|d1) to (d2|d3)
1174	trapeze	Graphics Mk III: draw a trapezium with sides (d0|d2) (d1|d2) and (d3|d5) (d4|d5) [not callable from IMP].
1178	triangle	Graphics Mk III: draw a triangle with vertices (d0|d1), (d2|d3), and (d4|d5) [not callable from IMP].
117C	defname	Define name A0 in dictionary A1, reserving D0 bytes. Return reference in A0.
1180	refname	Look up name A0 in dictionary A1,return reference in A0.
1184	transname	Perform reverse lookup on reference A0, returning name to string A1.
1188	defineeh	Define Event Handler (IMP support). The partial context (PC, A4, A5, A6, SP) is saved somewhere, and restored when any event is signalled.
118C	nhex	Print D0 as single-digit hex number (nibble)
1190	bhex	Print D0 as two-digit hex number (byte)
1194	whex	Print D0 as four-digit hex number (word)

1198	phex	Print D0 as eight-digit hex number
119C	rhex	Read hex number to D0

8. GRAPHIC SYSTEMS

8.0. Introduction

The information for this section is currently being re-typed into the machine. (Recovery from head crash).

A copy of what appears to be the missing section has been added to the reprinted edition. Originally this was Section 4, numbered 4.1 – 4.11

LEVEL 1 GRAPHICS SYSTEM

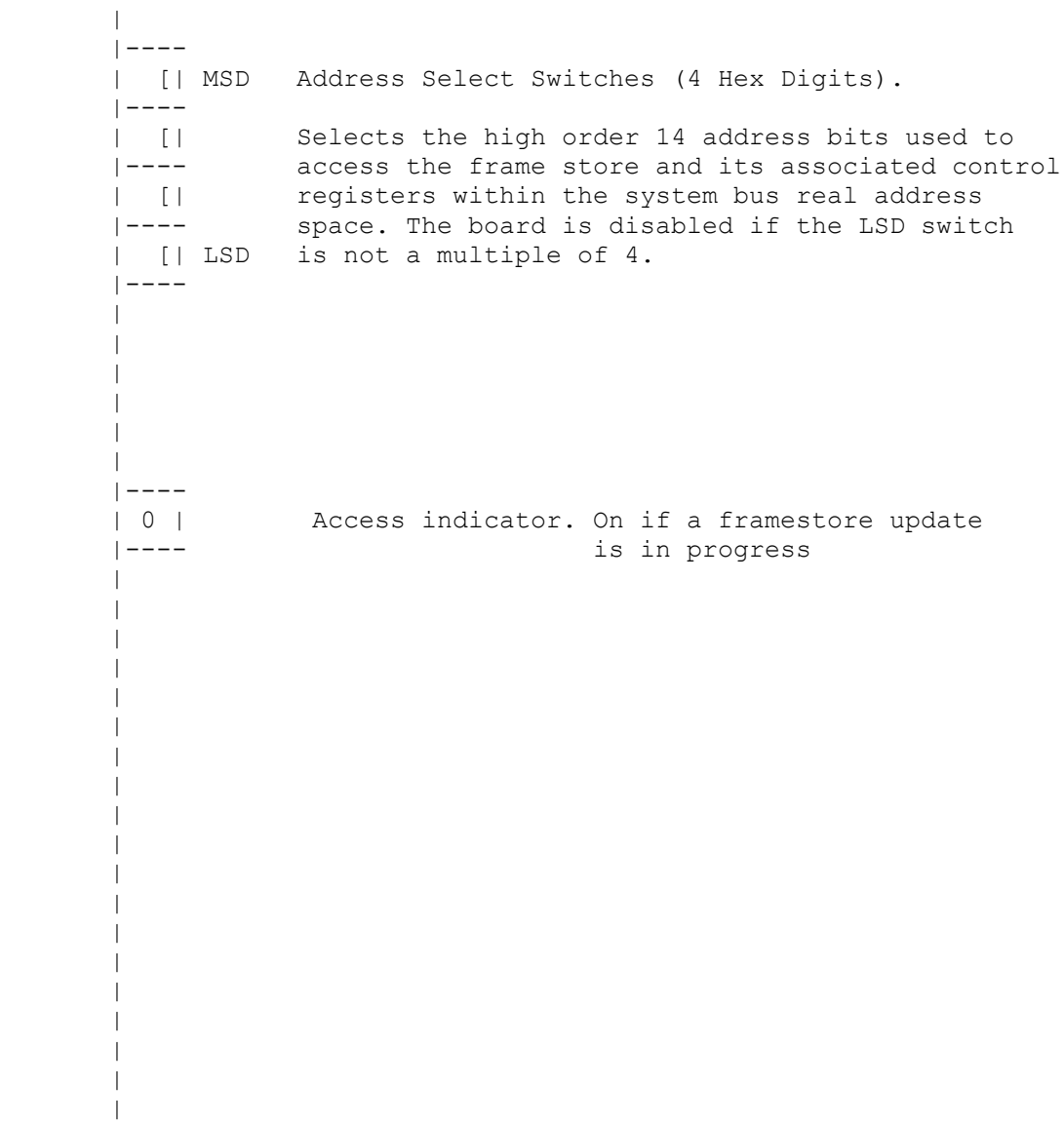
8.1 Introduction

The level 1 graphics board (CSD154) is a simple, write only, colour raster graphics controller and frame store. It provides a display of up to 1024*1024 pixels of four bits each. It can be configured to drive a large variety of monitors, both interlaced and non-interlaced and at different data, line and frame rates. It generates 3 TTL video outputs allowing 8 different colours to be displayed.

A further board (CSD155) can be added to give an additional four bits per pixel and a colour map suitable for driving analogue input colour monitors. Together they can display up to 256 different colours at one time.
These colours can be selected from a total of 32768 shades.

8.2 Switches and indicators

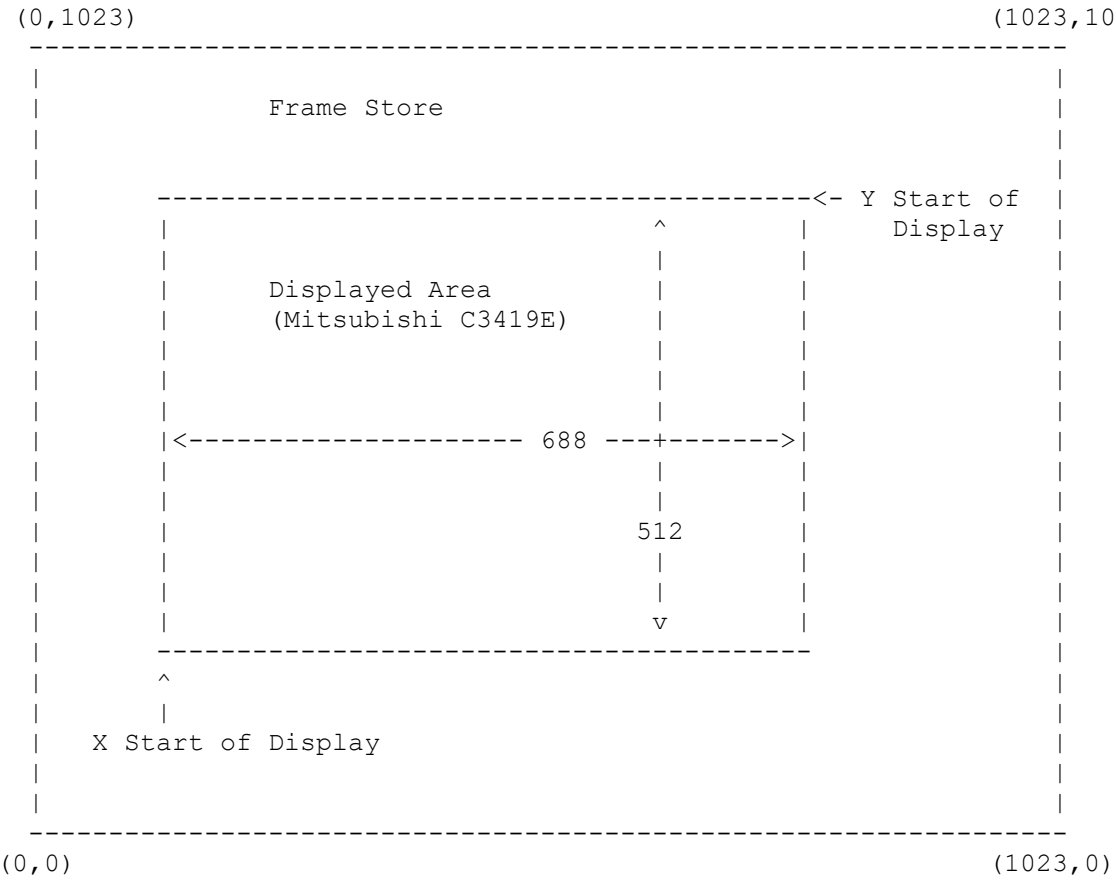
When viewed from the front of the system the layout of the control switches and indicators on the control processor board is as follows:



8.3 Coordinate System

The frame store is an array of 1024*1024 pixels. Pixel (0,0) is at the bottom left of the array and pixel (1023,1023) is at the top right of the array. The attached monitor will display a rectangular sub-region of this array. The size of the displayed area is determined by the Monitor specifications and is configured by PROM's on the graphics board. The display area for the Mitsubishi C3419E monitors is 512(V)*688(H) pixels. The origin of the displayed region is determined by the contents of a write only register, the START OF DISPLAY REGISTER. The Start of Display register

is a 16 bit register. The 10 most significant bits specify the Y index of the TOPMOST displayed pixel. The 6 least significant bits specify the X index of the LEFTMOST displayed pixel, DIVIDED by 16. The X index of the first displayed pixel must be a multiple of 16. The two bytes of the register may be written separately. If the origin is such that an overflow occurs then the displayed region wraps around from high to low index pixels.



8.4 System Bus Interface

The level 1 graphics board provides a number of write only locations that can be accessed as slave devices from the system EUCSD bus.

The locations they occupy within the system bus address space are determined by 4 hex coded switches that are visible from the front of the board. These specify the 14 high order address bits (AD31L-AD18L) used to access the frame store and control registers. The next highest address line AD17L determines whether the frame store (AD17L is 0) or the control registers (AD17L is 1) are accessed.

If the control registers are accessed AD16L is used to determine whether the general control registers (AD16L is 0) or the colour map (AD16L is 1) is accessed. Note that accessing the colour map is effective only when the CSD155 board is available.

8.5 Frame Store Addressing

The Frame Store appears as a linear sequence of 128K bytes in the system bus address space. This may be considered as a 1024*1024 array of bits, organized as groups of 8 bits in a 1024*128 array of bytes. Increasing byte addresses correspond to increasing X coordinates first and increasing Y coordinates second. Within a byte the most significant bit corresponds to the pixel with lowest X coordinate. Thus, if we consider a particular sequence of pixels as a binary number they appear on the display in the 'natural' left to right order of decreasing significance.

Up to 4 aligned bytes may be written in a single frame store access. The user should note that internally 16 bit operations are performed. This implies that the store cycle time is doubled for 32 bit accesses.

8.6 Pixels - CSD154 board only

Each pixel is associated with 4 bits of data. Each bit is stored in a separate 1024*1024 bit storage array or PLANE. The contents of planes 0, 1 and 2 determine whether the red, green and blue guns of the monitor are on or off. Plane 3 , the 'cursor' plane allows the sense of data from planes 0, 1, and 2 to be inverted.

Thus, for any given pixel:

```
if plane 3 = 0
      red gun is on if plane 0 = 1
    green gun is on if plane 1 = 1
     blue gun is on if plane 2 = 1

if plane 3 = 1
      red gun is off if plane 0 = 1
    green gun is off if plane 1 = 1
     blue gun is off if plane 2 = 1
```

Note that a pixel may be any one of the following colours :

black	all off	0000 or 1111
red	red only	0001 or 1110
green	green only	0010 or 1101
yellow	red+green	0011 or 1100
blue	blue only	0100 or 1011
magenta	red+blue	0101 or 1010
cyan	green+blue	0110 or 1001
white	red+green+blue	0111 or 1000

Notice that the colour displayed when plane 3 is 0 is the complement of that displayed when plane 3 is 1, provided that planes 0, 1 and 2 remain the same. Thus plane 3 may be used as a cursor plane on which any symbol written will probably be noticeable, regardless of the contents of the other planes.

8.7 Pixels - CSD154 and CSD155 two board system

With the CSD155 option installed each pixel is associated with 8 bits of data. Each bit is again stored in a separate 1024*1024 colour plane. The method of updating the frame store is unchanged. However, the manner in which each pixel is displayed is altered.

When a pixel is displayed the 8 bits of data are used as an index to a writable COLOUR MAP. For each of the 256 possible values a pixel may take a 16 bit value is generated by the colour map. These 16 bit values are written into the colour map before the desired picture can be displayed. The 16 bits are used as three 5 bit fields and a single 1 bit field. The 5 bit fields are used as an unsigned integer to determine the analogue intensity of the red, green and blue guns of the CRT display. The 1 bit field is used as a control bit that determines whether the pixel flashes at one 16th of the display field rate; about 3 times a second.

The colour map may be updated by normal writes to the graphics system. The low order address lines (AD9L-AD2L) are used to index the colour map and the data written provides the 16 bit values generated by the colour map. The colour map appears to the bus as 32 bit entries, the 16 high order bits being ignored. The individual bytes of the colour map entries may be written separately.

The operation of writing a single new entry to the colour map will cause the display to be blanked for 3 pixels. This is to minimise the effect of generating incorrect data when a change is made. It may be possible to use the vertical sync output from the controller to synchronise colour map changes to the display vertical blanking period. This would allow colour map updates to go completely unnoticed on the display.

8.8 Updating the Frame Store

Only write accesses to the frame store are allowed. When the frame store is written an UPDATE OPERATION takes place. The data when written does not go directly to the planes, but is used as a mask. Each bit in the mask corresponds to a pixel as discussed in the section on frame store addressing. If a data bit in the mask is 0 then no change is made to the corresponding pixel. If a bit is 1 then the pixel is updated. The change made to the bits that make up the pixel is determined by the contents of two control registers.

The PLANE ENABLE REGISTER is a single byte and contains a bit corresponding to each of the colour planes. If the bit for a plane is zero, then that plane is not changed by frame store update operations. If the bit is 1 then the plane may be modified by the update operation. If the CSD154 board is used alone then only the 4 least significant bits are used.

The COLOUR REGISTER is a single byte and contains a bit corresponding to each of the colour planes. When a frame store update operation takes place the contents of the colour register is written to the corresponding plane of each pixel that has 1 in the data mask. This is of course conditional on

the enable bit for that plane being 1. If the CSD154 board is used alone then only the 4 least significant bits are used.

This arrangement allows all planes to be updated simultaneously. However, it does not have any overall speed up when drawing finely detailed pictures. For example, to clear the frame store the following must be performed. First all 1's are written to the plane enable register. Then all 0's are written to the colour register. Finally all 1's are written to all the frame store locations.

To draw an arbitrarily coloured picture we must proceed as follows. For each colour the colour register must be written before writing 1's to each pixel that will be that colour in the final picture. This must be repeated until the detail for all colours is drawn.

For 'layered' applications such as VLSI plots it is sufficient to proceed as follows. First write all 1's to the colour register. Then for each layer write a single 1 to the plane enable register, before updating the frame store with the detail for that layer. Again this must be repeated until the detail for all of the layers is drawn.

8.9 Register Address Map

To summarise, the registers appear in the system address space as follows:

```
SSST00000 - SSST0FFFF  Frame store.

SSST20000(4)SSST2FFFC  Plane enable register.
SSST20001(4)SSST2FFFD  Colour register.
SSST20002(4)SSST2FFFE  Start of Display register.

SSST30002(4)SSST3FFFE  Colour map registers.

SSST is the setting of the board address select switches
The system is deselected unless T is a multiple of 4.
```

8.10 Register Bit Encodings

Collectively the general control registers may be considered as a single 32 bit word. Each of the bytes may be accessed separately. The meaning of each of the bits in order of decreasing significance is as follows:

Byte 0 (MSByte) - Plane Enable register

```
bit 7 (MSB) - Plane 7 enable bit -
bit 6       - Plane 6 enable bit  | Significant only
bit 5       - Plane 5 enable bit  | with CSD155 extension.
bit 4       - Plane 4 enable bit -
bit 3       - Plane 3 enable bit
bit 2       - Plane 2 enable bit
bit 1       - Plane 1 enable bit
bit 0 (LSB) - Plane 0 enable bit
```

If a bit is 0, updates to corresponding plane are disabled
If a bit is 1, updates to corresponding plane are enabled

Byte 1 - Colour Register

```
bit 7 (MSB) - Plane 7 data bit -
bit 6       - Plane 6 data bit  | Significant only
bit 5       - Plane 5 data bit  | with CSD155 extension.
bit 4       - Plane 4 data bit -
bit 3       - Plane 3 data bit
bit 2       - Plane 2 data bit
bit 1       - Plane 1 data bit
bit 0 (LSB) - Plane 0 data bit
```

The content of each bit is written to the correspondingn plane during a suitably enabled and masked update operation.

Bytes 2 and 3 (LSByte) - Start of Display Register.

```
byte 2 bits 7-0
byte 3 bits 6&7  -  Y Finishing Index
                    Y index of the last pixel displayed.
                    byte 2 bit 7 is the most significant bit.
                    byte 3 bit 6 is the least significant bit.

byte 3 bits 5-0  -  X Starting Index
                    X Index/16 of first pixel displayed.
                    The X index of the first pixel displayed
                    is always a multiple of 16.
                    bit 5 is the most significant bit.
```

The colour map entries are 16 bits wide, but are 32 bit aligned. Only the Least significant 16 bits are used. Either of the two bytes may be written separately.

They have the following form :

```
byte 2 bit 7    -  Flash bit
                   If 0 pixel is steady
                   If 1 pixel alternates between colour specified by rest
                   of entry and black every 16 fields (about 3 times a
                   second)
```

byte 2 bits 6-2 - Blue intensity
byte 2 bit 6 is the most significant bit.

byte 2 bits 1 & 0,
byte 3 bits 7-5 - Green intensity
byte 2 bit 1 is the most significant bit.

byte 3 bits 4-0 - red intensity
byte 3 bit 4 is the most significant bit.

8.11 Software Support Routines

A number of standard routines are available for using the Level 1 Graphics System. The following routines are written in assembler and their close interaction with the hardware exploits the full potential of the graphics system. their peformance is such that they should be used whenever possible. In assembler, parameters are passed in registers d0, d1, ... d5 in the order specified by the prototypes given below.

plot(x, y)
Sets the pixel at coordinate (x, y) to the current contents of the colour register. The planes affected are those specified by the plane enable register.

fill(x0, y0, x1, y1)
Fills a rectangular area bounded by the coordinates (x0, y0) and (x1, y1). The colour used and planes affected are taken to be the current contents of the colour and plane enable registers. If x0>x1 or y0>y1 the the area filled is wrapped around the frame store.

line(x0, y0, x1, y1)
Draws a line between the coordinates (x0, y0) and (x1, y1). The colour used and planes affected are taken to be the current contents of the colour and plane enable registers.

triangle(x0, y0, x1, y1, x2, y2)
Draws a filled triangle with vertices (x0, y0), (x1, y1) and (x2, y2). The colour used and planes affected are taken to be the current contents of the colour and plane enable registers.

trapeze(x0l, x0r, y0, x1l, x1r, y1)
Draws a filled trapezium with parallel sides parallel to the X axis. The coordinates of the vertices are (x0l, y0), (x0r, y0), (x1l, y1) and (x1r, y1). The colour used and plane affected are taken to be the current contents of the colour and plane enable registers. Trapeze is primarily intended for polygon filling.

In other languages further routines may be provided. This will be in addition to suitably interfaced calls to the basic routines given above.

The Souris Mouse is a graphics pointing device consisting of a hemi-spherical plastic box which is grasped by the hand lying on the domed top with the thumb and small finger gripping the flattened sides. Three switches are mounted on the front edge, corresponding to the middle three fingers.

A steel ball is held, free to rotate, in a chamber above the base of the mouse with a hole in the base so that the ball can run on any reasonably flat and horizontal surface. Spring loaded rollers press the ball against two steel shafts whose axes lie at right angles to each other in the horizontal plane. These shafts carry encoders which send signals to the interface to control x and y counters. The states of the switches are also transmitted to the interface so that they can be used as command inputs.

Although the mechanism is accurately made it is not suitable for use as a digitiser. It should be used only as an interactive device where the position of some object, eg a cursor on a screen, is related to the position of the mouse on whatever surface it is running on. The advantages of the mouse are that it does not require any special surface to run on and it is less expensive than a digitiser.

8.12.1 Low level Software Interface

The state of the mouse is held in three 16 bit registers: the x axis counter, the y axis counter and the buttons register. These are available at addresses 16_7FFF0, 16_7FFF2 and 16_7FFF6 respectively.

The buttons register is read only. The left button corresponds to bit 8, the middle button corresponds to bit 9 and the right button corresponds to bit 10, where bit 0 is the LSB of the 16 bit register.

The x and y axis counters are read/clear only. Their resolution is 15 counts per mm. Clearing is accomplished by writing 0 to them. Writing any other value has the same effect.

No interupt mechanism is provided by this interface since it is assumed that a system timer will be available to provide a general timed interupt mechanism which was thought to be the most appropriate way to handle this device. Since the device keeps track of the X and Y positions it is not time critical below human reaction times.

8.12.2 Hardware Interface

The Souris Mouse is interfaced to the APM Control Processor Local bus but acquires its Power and Ground from the EUCSD Bus Connector. The interface occupies part of the bottom half of a double height extended length Eurocard. The first batch are built using the department’s Zap technique since it was thought possible to incorporate a keyboard interface on the same board at a later date.

The Souris Mouse provides quadrature signals for each of the X and Y axes of movement and three signals indicating the state of the three buttons. These signals along with power and ground are carried through a 9-way cable to a 9-way D-type plug which mates with a 9-way IDC ribbon cable socket. The ribbon cable terminates in a 10-way IDC socket which mates with the on board 10-way 3M type dual in line IDC plug.

The X and Y axis quadrature pairs are taken to a 74LS374 latch (N50). The corresponding outputs are fed back to the remaining 4 inputs so that following each clock edge the latch outputs give the current and previous states of the quadrature pairs. These signals are fed into the least significant 8 address lines of a 2716 Eprom (M50). The least significant 4 data outputs of the Eprom provide count enable and Up/Down signals to each of the X and Y counters (A,B,C,D50 and A,B,C,D62).

The X and Y counters are 16 bit synchronous counters with asynchronous clear and tri-state outputs. Each change of state of the X or Y quadrature signals results in an up or down count of the X or Y counters so that the resolution of the counters is one 15th. of a mm.

Use is made of the 800KHz E clock of the Control Processor Local Bus. This is divided by 2 using a 74LS74 (E74) to give CLOCK which is used to clock the X and Y counters and the Latch/Eprom state machine mentioned above.

The interface to the Control Processor Local Bus provides read access to the X axis counter, the Y axis counter and a Buttons register (E62). The counters can be cleared by writing to them but the write data is ignored. The register addresses are as follows

X axis Counter	16_7FFF0
Y axis Counter	16_7FFF2
Buttons Register	16_7FFF6

Address 16_7FFF4 is taken but not used. Otherwise, the addresses are fully decoded.

The bus signals UDS' and LDS' are ORed together by a 74LS08 (D74) and inverted by the permanently enabled half of a 74LS240 inverting bus buffer (E62) to give the signal UDS. The signals R/W', A3 and AS' are similarly inverted to give the signals R'/W, A3' and AS.

A3' and A(4:18) are taken through two 74LS30 nand gates (A74 and B74) to give signals EN1' and EN2' which, together with AS and UDS, enable the 25LS2548 address decoder (C74). A1,A2 and R'/W provide the address inputs to the decoder which produces the valid signals RD0',RD2',RD4',WD0' and WD2'. In addition the Bus signal DTACK' is produced by the 25LS2548 when it is enabled and LUDS' is active.

LUDS' is produced by latching UDS with the system clock CLK and is simply a delayed and inverted UDS. It is almost certainly redundant and was incorporated only because it cost nothing and was "purer".

RD0' enables the x counter outputs to the bus. RD2' enables the y counter outputs and RD4' enables the buttons register (half of E62). RDO' and RD2' are ORed by the 74LS08 (D74) to give HCLK' which holds the signal CLOCK during read access to the counters.

The bus signal RST' is similarly ORed with WD0' to give CLRX' which clears the x counter and with WD2' to give CLRY' which clears the y counter. In addition it initialises LUDS' to be inactive.

It was found that the mouse interfaces were failing intermittently and not so intermittently depending on the particular board and which APM they were plugged into. This was traced to noise on the RST' signal and as a result a small AC filter in the form of a 100 ohm resistor in series with a 100 pF capacitor was connected between RST' and GND to remove this problem. This fix was found to be of more general benefit to the APM system and it is recommended that the APMs have such filters fitted to all bus control lines.

9. C ON THE APM AND VAX

As of 12/04/83, a version of the MIT portable C compiler (sans floating point support) which generates M68000 machine code has been moved to the APM. It compiles C as described in Kernighan & Ritchie's book "The C Programming Language". Enough of a run-time handler exists to support the compiler, linker, etc. The standard i/o header is in C:STDIO.H which, for the present, should be denoted as “C:STDIO.H” rather than as the more conventional <STDIO.H>.

On the APM, C can be accessed via the 'command' C:CC. The format is:
C:CC <filename> [<pre-processor flags>]

The output relocatable machine code is left in <basename>.REL, where <basename> is formed by stripping any extension (rightmost ...) from <filename>. Pre-processor flags are described below.

.REL files can be linked together using the C: CLINK command:

C:CLINK <f1>....<fn> [-i<ct1>] -?<output>

<f1> etc. are a space-separated list of .REL files; C:CLIB.REL is the name of the standard run- time library (see below). -i<ct1> is an optional control file containing a list of files to be linked and possibly; linker -i and the filename. ? is o, b or p depending on whether the output is to be another .REL file (which can be linked further), a binary image (.MOB file, conventionally) which is runnable, or a patched (optimised) binary image (.MOB, again) in which as many JSR xxxxxxxx as possible are converted to PC-relative form to minimise image size and loading time. NOTE: THIS CONVERSION MAY NOT ALWAYS BE SAFE. DO NOT COMPLAIN UNLESS A -b IMAGE FAILS TO RUN. Once again, there must be NO space between -? and the filename (which must be given in full, with the extension explicitly supplied by the user).

9.1. Pre-processor Flags

The pre-processor allows a sequence of symbols to be £identified{d} from the command line. These are given as -dthingl -dthing2 etc. Similarly, symbols may be initially £undef{ined} using -uthing...

9.2. The Run-time Library

The run-time library implements functions such 33 printf,scanf, exit, etc. At the present time the library 1-5 sufficiently extensive to support the C compiler. Omissions should be notified to LDS. Deviations from the Unix conventions for library support should, likewise, be notified to LDS. The standard i/o header is in C:STDIO.H. The C source of the run-time library is in C:CLIB.C.

The major omissions at the moment are:

1. No support for floating-point numbers (there is no machine support for them yet anyway).

2. lseek has not been implemented; fseek supports fseek(fp,0,0) ONLY (seek to start of file). Limitations in the filestore interface currently preclude a full implementation of either of these functions.

3. On the APM, there is currently no way to include only part of the run-time library (the bit you need): you either get it all or you get none of it. Fixing this requires further discussion of library standards and linker/loader standards for the APM.

9.3. The C Linker

The C linker assembles a number of .REL files into either (a) another .REL file which can be linked further or (b) an executable, self-relocating image, which can be run according to the APM conventions (1st byte of image is X'FE', load the image anywhere and JSR to the last longword of it). If the image so produced is left in a file THING. MOB then it can be run on the APM by issuing the •command' THING. When the linker is building an executable image it automatically loads the C machine support (C:CRTS) and the bootstrap relocator (C:CLDR). The C run-time library C:GLIB.REL is NOT loaded automatically.

The format of the CLINK command is described above.

If an input control file is specified with a -i<ct1> parameter then a list of names of files to be linked may be given (the list maybe empty) FOLLOWED by a list of linker control commands. The names of files to be linked together and the linker control commands must be separated by spaces and/or newlines. The linker control commands are as follows:

```
*equate  <symbol1> to <symbol2>
*renaae  <symbol1> as <symbol2>
*hide    <symbol>
*hideall
*unhide  <symbol>
```

ONLY *equate HAS BEEN TESTED; report bugs to LDS.

*equate <s1> to <s2> causes all references to <s1> (which MUST be UNdefined) to be re-directed to <s2> (which need not be defined).

*rename <sl> as <s2> causes the symbol <s1> to be renamed <s2>; <s2> must NOT EXIST at the point where the renaming takes place.

*hide <s> deletes external symbol <s> from the symbol-table of the output .REL file. <s> MUST be defined. This allows to modules with different definitions of (say) external routine fred (or, rather, the C equivalent thereof) to be linked together quite happily

by hiding one of the freds from the other.

*hideall hides all DEFINED symbols; an UNdefined symbol cannot be hidden (otherwise the reference would NEVER be resolved).

*unhide <s> reverses the effect of a *hide <s> or *hideall command.

Typically, I would expect these commands to be used to assemble an almost complete module with a carefully controlled set of exported symbols. Such a module can be called by users without fear that their choice of external names might clash with that of the module builders (and if it does then the clashes can be resolved using *rename).

9.4. C Cross-System on VAX

The C68000 cross-compiler can be accessed on VAX by using @dro0: [lds.c68000jc.com. Assuming that c68 has been equivalenced to this, the command format is:

c68 <file> [<pre-processor flags>]

An extension .C is appended to <file> unless it already has an extension; the output is left in <basename>.REL, where basename is formed by stripping ant extension from <file>.

The cross-linker is dr0:[lds.a68000]clink.exe. The command format is indentical to that on the APM. However, the -i<ctl> file MUST be in a non-standard (to VAX) simulated filestore format. Such files can be created using @dr0:[lds.fs68000] fssend.com <vaxfile> <simulated filestore file>. (Thus, linker control input must be sent be sent to the simulated filestore before use.) @dr0:[lds.fs68000] fsfetch.com <sim-fs-file> <vaxfile> reverses the process. NOTE: only text files with line lengths < 128 characters can be sent and fetched this way (blame VMS Vsn 3's Pascal, not me).

The run-time library is in dr0:[lds.c68000]clib.rel and the standard header is in dr0:[lds.c68000]stdio.h.

Use the subcommands of the VAX command newfs to transfer .rel and simulated filestore files between the filestore and VAX.

9.5. Running APM C Programs on VAX

As is well known, we currently only have the rather crummy Whitesmith's C on VAX. It is especially crummy for having a (very) non-standard run-time library. However, with some inconvenience, APM C programs may be run on VAX under a simulated filestore environment (input files must first have their format converted from VAX standard to filestore standard using @dr0:[Ids.fs68000lfssend.com and output files must be converted back using @dr0:[lds.fs68000]fsfetch.com). Indeed, the C cross-linker and the whole of the C68000 cross-compiler suite operate under this

environment (however ,for the convenience of users, the C preprocessor reads standard VAX-format text files as input so no special action is needed in order to use the cross- compiler).

The advantage of this is that EXACTLY the same run-time library is maintained VAX and on the APM.

To compile an APM C program for running on VAX use the VAX command CC (input file extension .C assumed, output file extension .OBJ generated).

To link this into an executable image use dr0:[Ids.c680003cl.com. Assuming that CL has been equivalenced to this the command format is:

CL <f1>,<f2>,....,<fn> [<exe>]

If no .EXE file is specified then <f1> is stripped of any extension and .EXE is appended to form the name of the output image. MOTE: any bits of the C run-time library you reference are included automatically.

Unfortunately there are certain differences between the machine support on VAX and that on the APM and between the source language accepted by the Whitesmith's compiler and that accepted by the MIT portable compiler.

9.6. Machine Support Differences

1. VMS insists on translating command lines to upper case, so that C m/c support translates them back to lower case. Ergo, don't expect any upper case letters in the arguments passed to main () from the startup code. On, the APM, command line arguments are NOT translated at all.

2. Programs interrupted by CTRL + Y on the APM MAY leave the m/c in an insanitary or hung state. This problem MAY be fixed if/when the APM operating system stabilises.

3. On VAX, terminal i/o is lazy: newline ('\n') and input operations both flush the output buffer. Input is line buffered. Up to 127 chars are buffered on both input and output. On the APM, terminal input is line buffered but terminal output is character at a time-.

4. Error messages MAY be different on the two systems, depending on their point of origin.

9.7. Source Language Differences

Fortunately, there are few significant differences between the source language accepted by the Whitesmith's C compiler (VAX CC command) and the MIT portable C compiler (APM C.CC. VAX @[lds.c6800]c.com). Note the following:

1. The Whitesmith's C compiler does not check types (very well). Use the cross-compiler to get your casts right and to detect where a cast has been omitted. The cross-compiler detects more errors and gives (slightly) better diagnostics.

2. MOST ANNOYINGLY Whitesmiths C INSISTS that ALL static and external objects are EXPLICITLY initialised whereas the language definition FORBIDS the initialisation of unions and GUARANTEES that all static and external objects which are not explicitly initialised have every field set to zero. APM C guarantees that all unintialised objects are initially zero. The problem can be circumvented using the following pre-processor code:

```
£ifdef
whsonvms /* better be lower-case...  */
£define BLNKDATA ={0}  /'fortunately, WHS C isn't too fussy...  */
£else
£define BLNKDATA /* a null definition •/
£endif
.....
union thing fred BLNKDATA;
.....
```

It is most efficient on the APM NOT to explicitly initialise static and external data which are intended to be zero initially. On the VAX you have no choice.

3. Whitesmith's C does not support any kind of aggregate assignment. MIT C appears to support whole aggregate assignment of ALL objects which fit into a byte, a short-word or a long-word.

4. The two compilers evaluate expressions in different orders. If you (by accident or by design) write expressions which depend upon evaluation order (e.g. (getchar() « 8) + getchar()) then you will get what you deserve: random output which, by accident, will be correct on one system but not on the other.

10. APM DEMONSTRATIONS

All demonstration files are in directory APM

10.1. General Programs

1. PALIN: A palindrome Program, magic numbers are: 196(2000:4 Sec)
   98(succeeds after 24)

2. CONC Concordances:

   Space Invaders For Visual 200:
   INV Keys are {4 Move left}
   {5 FIRE}
   {6 Move Right}

10.1.1. Graphics Performance Related

Speed of area fill,line drawing,etc:

1. FILL: fills the frame with colour. Type numbers in range 0:15

2. SPOKES: multiple lines

3. WHEEL: draws lines

4. TRIGTEST: Evaluates software floating point SIN,COS,SQRT

5. OCTAG: Triang fill

6. SCAN: Uses cursor keys to move around

7. RUBIK: Uses pre-set sequence. DIR.RUBIK for user-driven operation

8. GOLLUM Adventure game

10.1.2. Vlsi-Related

CHARLES is a simulator of Charles terminal. Files EDWIN: {OUT,PLA,STACK}.APC are available for demonstration purposes. A null filename re-draws previous file without going back to filestore. Cursor keys move the picture around.

10.2. Animation

1. GRAPHAN: Tower of Hanoi.

2. TETRA: Goes through pre-set sequence. For user-driven operation use FTETRA.

3. DISPLAY: displays text, filename{fontfilename}{<space>paper<space>}
   Default fontfile FMACS.FONT:VISUAL

Default colours are green or black

10.3. Other Programs

There is also a program RDISPLAY for a screen rotated through 90`. The Default font is FMACS.FONT:RBANTAM Other font files are FMACS.FONT: {BANTAM, IGOR,RVISUAL,RIGOR}

The Fred-machine or APM served two purposes – as a testbed for research into advanced concepts in computing and as a service vehicle for teaching, research and administration.

The original APM Working Documents formed a starting point for anyone wanting to use the systems or to develop or extend its capabilities. Thirty-five years later it is still a logical starting point for anyone wanting to understand the project or to get a feel for what life was like around it.

The references here describes as many of these projects as can be identified. Some went on to form integral parts of the system, used by everyone, some served as the basis of theses or papers and some were discussed or tried out but never took off..

I was not part of this development – where I came in was to manage what was eventually 50 or so Fred Machines as a departmental computing service and try and balance the needs of a service (reliability and stability) with the needs of research (tweaking it and trying things out, some of which would fail).

The list that follows is an extension of an earlier list prepared in 1986. I have kept the numbers the same to avoid confusion.

I was credited [ref. 1.13] with a document "Complete Guide to the Fred Machine" (EUCSD Manual 1985). I don't recall ever producing that document then but this is as close as I can get now. Missing the publication deadline by 35 years may be some sort of record.

John Butler, May 2020

GENERAL

1.1 APM working documents Rev. 1.0 (May 1983)
1.1a Addendum to APM working documents - Level 1 Graphics (F. King)
1.2 view:apm (HELP APM)
1.3 PAM: Parameter acquisition module (H. Dewar, Mar 1984)
1.4 view:sfp (HELP SFP)
1.5 Motorola specifications: 6840 PTM, 6850 ACLA, 6821 PIA (Motorola)
1.6 M68000: 16-bit microprocessor User's Manual (Motorola)
1.7 Intel specifications: 8257/8257-5 Programmable DMA controller
1.8 Zilog Z80/Z80A CPU: Technical Manual
1.9 "The Edinburgh Advanced Personal Machine, or LEGO bricks and computing" (F. King, H. Dewar, I. Hansen, R. Thonnes)
1.10 "A Brief Introduction to the APM System" (J.G. Hughes)
1.11 512Kb Memory board controller state machine definition
1.12 QSART: 2651 PCI specification and register addresses
1.13 "The Evolution of the Fred Machine" (G. Brebner, F. King), CS Dept. Internal Report CSR-246-87, Sep 1987
1.14 CS component sheets and schematics

LANGUAGES

2.1 “The IMP-77 Language” (P.S. Robertson)
2.2 view:lib (HELP LIB)
2.3 view:c (HELP C)
2.4 IMP-77 core environment definition (Lattice Logic)
2.5 i:maths.inc
2.6 Summary of the major IMP libraries 1984 (J.G. Hughes)
2.7 view:imp (HELP IMP) IMP compiler for Motorola 68000 (H. Dewar)
2.8 APM-TD: Runtime support for IMP/Pascal (H. Dewar, 15/08/83)
2.9 APM-TD: IMP/Pascal Procedure calling and register usage (H. Dewar, 19/10/83)
2.10d IMP Event Numbering - Proposed Revision (H. Dewar)
2.10 Synchronous Events in IMP 2.0 (H. Dewar)
2.11 APM system documentation: Program Environment (H. Dewar)
2.12 APM Pascal (H. Dewar)
2.13 IMP 3.0 (H. Dewar, CLAN systems)

COMMUNICATIONS

3.1 ether:guide.lay "A programmer’s Guide to the EUCSD Ethernet" (R. Thonnes, Dec 1983)
3.2 ids.inc (IMP include file)
3.3 Standard video terminal interface
3.4 fmacs:clients.inc
3.5 jhb:chario.lay "APM Asynchronous character I/O" (J. Butler)
3.6 i:chario.inc
3.7 VAX Ethernet Link Driver (G.D.M. Ross)
3.8 Ethernet Protocols: Design and Implementation (W.P.S. Enos, MSc. Thesis)
3.9 view:eftp The EFTP program (R. Thonnes)
3.10 Installation criteria for Ethernet Coaxial Cable (C. Young 10/5/84)
3.11 A Hitch-Hiker’s Guide to Ethernets, ECMA and ISO (J. Butler)
3.12 APM-TD: The development of the EUCSD Network (G. Cleland, 01/02/84)
3.13 Edinburgh Local Area Network (W. Enos, I. Hansen, R. Thonnes, undated)
3.14 ISOing the network (R. Thonnes)

FILESTORE AND AUTHORISATION

4.1 “The Filestore” (H. Dewar, V. Eachus, K. Humphry, P. McLellan, Dept. of Computer Science Oct 1977, revised Sep 1981 (draft))
4.2 “Chapter 2 – The Local File System” (part document, no further info)
4.3 The (New) New Filestores (G. Ross, Feb 1985)
4.4 The New Filestores (draft, G. Ross, Aug 1985)
4.3 rev The (New) New Filestores (G. Ross, revised, Oct 1985)
4.5 Thoughts on a File Access Protocol (G. Ross, Oct 1985)
4.6 Network Authorisation Protocol (G. Ross, Nov 1985)
4.7 File Access Protocol (G. Ross) (draft, Dec 1985)
4.8 Level 5: What Should it Do? (G. Ross, Dec 1985)

PRINTING

5.1 “Layout”, P. McLellan, Dept. of Computer Science Internal Report CSR-21-78, Feb 1978
5.2 “Layout 1.5 User’s Guide (provisional)” (Hamish Dewar, Dept. of Computer Science, Feb 1984)

FUTURE DEVELOPMENTS AND OPERATING SYSTEMS

6.1 Proposal to Port a Unix Environment to APM (G. Cleland, Aug 1984)
6.2 Some Ideas About the new FS + The Kernel within the APM (J. Wexler, Sep 1985)
6.3 A New Operating System for the Fred Machine (anon, Jan 1987)
6.4 Sons and Daughters of APM (D. Rogers, Dec 1988)

MISCELLANEOUS

7.1 Emulating the Fred Machine (B. Foley, M.Sc. Thesis, 2003)