SLIPSTREAM

The Slipstream Computer System

Hardware Reference Guide

Version 0.01
21 November, 1988

by Martin Brennan, Ben Cheese and John Mathieson,

Flare Technology Limited,
Unit O,
The Paddocks Trading Estate,
347 Cherry Hinton Road,
Cambridge CB1 4DH.

DISCLAIMER AND WARNINGS

Neither the whole nor any part of the information contained herein, nor any product described in this manual may be adapted or reproduced in any form except with the prior approval of Flare Technology Limited ('Flare').

All information of a technical nature and particulars of the product are given by Flare in good faith. However, it is acknowledged that there may be errors or omissions in this manual.

Flare has a policy of continuous review and improvement of its designs, therefore information in this manual is subject to change without notice. Whilst every effort will be made to remain compatible with existing standards, software authors and add-on hardware designers should note that :-

1 - Use of undocumented features or modes
2 - Use of I/O or Memory locations marked as 'reserved'
3 - Assumptions about values read from undocumented I/O read bits
4 - Setting of unused I/O write bits to other than 0
5 - Any other use of the hardware in a manner not indicated in this or other Flare documentation to be valid

are all actions that are liable to make software or hardware products not compatible with future revisions.

All data marked in pencil are provisional and precise details are subject to change.

At the date of issue of this guide the information contained herein is a company secret of Flare. Persons issued with this guide should treat the information contained here as highly CONFIDENTIAL, and in particular must not reveal the contents of this guide to any third party without the prior approval of Flare. If this information has been revealed under a non-disclosure agreement then the conditions given in that are binding with regard to this document.

TABLE OF CONTENTS

DISCLAIMER AND WARNINGS 2

TABLE OF CONTENTS 3

THE SLIPSTREAM COMPUTER SYSTEM 5

1. INTRODUCTION 5
    1.1. This Document 5

2. ARCHITECTURE 6
    2.1. Memory Map 7
    2.2. IO Map 8
    2.3. Memory Interface Timing 11

THE VIDEO CONTROLLER 14

1. INTRODUCTION 14

2. THE SCREEN MAP AND DISPLAY MODES 15
    2.1. Low resolution 200 line display 15
    2.2. Medium resolution 200 line display 16
    2.3. High resolution 200 line display 16
    2.4. Logical to Physical Colour Translation 16
    2.5. Colour Hold Mode 17
    2.6. Border Colour 17

3. SCREEN TIMING 18
    3.1. Video Line Timing 18
    3.2. Video Field Timing 18

4. THE REGISTERS 19
    4.1. Interrupt registers 19
    4.2. Start registers 19
    4.3. Horizontal count registers 20
    4.4. Vertical count registers 20
    4.5. Scroll registers 20
    4.6. Screen mode register 21
    4.7. Border colour registers 21
    4.9. End registers 22
    4.10. Memory configuration 22
    4.11. General purpose register 22
    4.12. Diagnostic 22
    4.13. Interrupt disable register 23
    4.14. Light-pen registers 23
    4.15. Status register 23

5. OPERATION OF THE CONTROLLER 24
    5.1. Refresh mechanism 24
    5.2. Interrupt mechanism 24
    5.3. A/D Comparators 24
    5.4. Peripheral IO 24
    5.5. Bootstrap 24

DIGITAL SOUND PROCESSOR 25

1. INTRODUCTION 25

2. MEMORY 26

3. DSP INSTRUCTION SET 28
   3.1. Instruction Codes 28

4. HOST MEMORY MAP 31

5. DATA TRANSACTIONS BETWEEN HOST AND DSP 32
   5.1. Under DSP control - DMA transfer 32
   5.2. Under Host Control - INTRUDE 33

THE BLITTER 34

1. INTRODUCTION 34
   1.1. This Section 34
   1.2. The Blitter 34

2. ARCHITECTURE 35
   2.1. Overview 35
   2.2. The Data Path 36
   2.3. The Address Generator 40
   2.4. The Sequencer 42

3. MODES OF OPERATION 50
   3.1. Simple Memory Fill and Copy Operations 50
   3.2. Line Drawing 52
   3.3. Character Painting 55

4. BLITTER COMMAND FORMAT 57
   4.1. Command Register 58
   4.2. Mode Control and Extension Bits Register 59
   4.3. Step Register 60
   4.4. Comparator and Logic Function Unit Control Register . . . 61
   4.5. Source Address Register 62
   4.6. Destination Address Register 63
   4.7. Program Address Register 64
   4.8. Blitter IO Registers 65

The FDC 65a

APPENDICES 66

A. The DSP Arithmetic Logic Unit 66

B. The DSP Assembler Manual 67
   1. How to invoke the assembler 67
   2. Source File format 67
   3. Error Reporting 71

THE SLIPSTREAM COMPUTER SYSTEM

1. INTRODUCTION

The Slipstream computer system was designed to be an ultra-high performance arcade games engine. It achieves this by utilising three processors, one general purpose processor, an Intel 8088, and two highly specialised co-processors, the Blitter and the DSP.

The system was designed to be as simple as possible, while still providing the necessary performance levels (compare the complexity of the Amiga); and to provide as much flexibility as the programmer may wish.

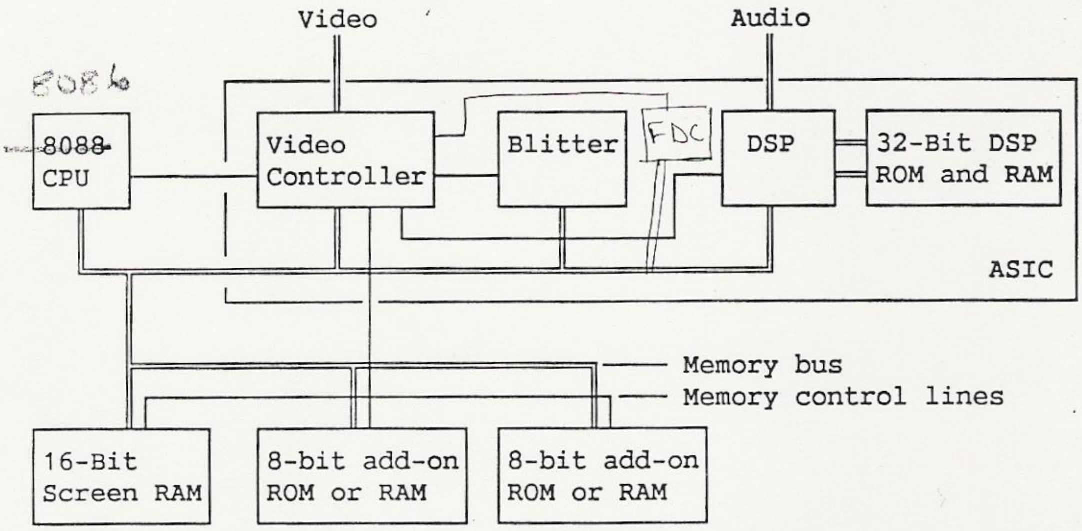
1.1. This Document

This document is intended to be a useful guide to the assembly language programmer who wishes to drive the hardware facilities provided in an effective manner. It is not for novices.

This first section discusses the computer system as a whole, and is followed by individual sections discussing each sub-module. The best approach is to read it through first, then to approach it as a reference guide once the overall context of sections is understood.

2. ARCHITECTURE

The Slipstream computer system is largely contained within one massive custom logic chip, known as an ASIC (for Application Specific Integrated Circuit). The system memory and 8088 CPU lie outside the ASIC. This diagram summarises the architecture: 8086



Sharing the main memory bus are ~~four~~ five bus masters; the Video Controller, FDC the DSP, the Blitter and the ~~8088~~ 8086. Only one of the bus masters may own the bus at any one time, and a priority for the bus exists, giving the Video Controller highest priority, followed by the DSP, the Blitter FDC and the CPU is descending order.

The video controller controls the bus, and provides the memory timing signals for memory devices attached to the bus. It has the highest priority on the bus, and will take the bus over during video lines to fetch video display data, and to refresh dynamic RAM.

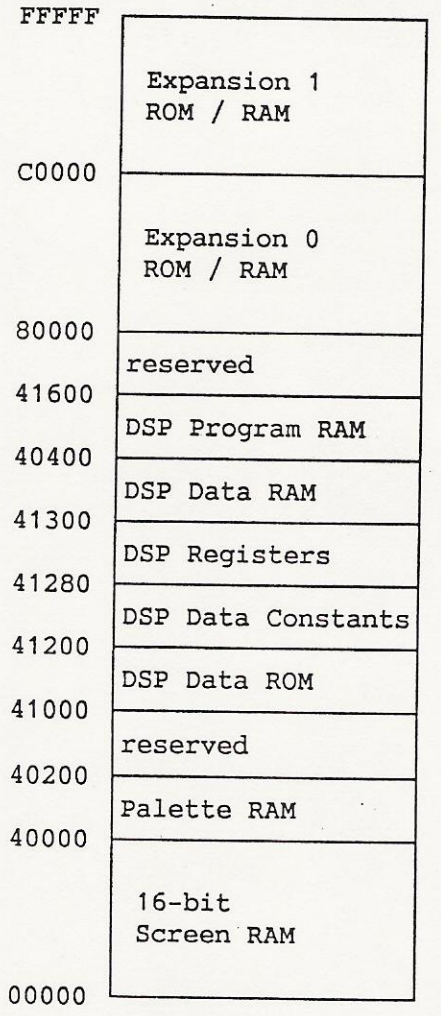
The DSP has access to the main bus via a DMA controller which allows it to read and write bytes or words from main memory. These transfers occur in short bursts, and are under DSP program control. The DSP actually executes programs and stores data in its own private high speed memory.

The Blitter will become bus master throughout a blitter program operation, and may therefore own the bus for considerable periods. However, its priority over the CPU is not total, as it may be requested to give up the bus to the CPU when an interrupt occurs.

The 8088 8086 CPU is the lowest priority bus master at the system level, but has complete control of the other two processors and so the use of the bus is entirely under program control.

The Floppy Disc Controller (FDC) controls the disc drive during read operations. It buffers up to 16 bits of read data, decodes it and then transfers it to system memory. Disc write operations are controlled by the DSP

2.1. Memory Map



The One Megabyte memory space is divided into four 256K areas, with the area based at 00000h being allocated for internal 16-bit screen RAM, the area based at 40000h being used internally by the ASIC, and the areas based at 80000h and C0000h available for expansion memory on the cartridge interface.

2.2. IO Map

Address	Write Register	Read Register
00	Interrupt line low	Horizontal light-pen low
01	Interrupt line high	Horizontal light-pen high
02	Screen start low	Vertical light-pen low
03	Screen start high	Vertical light-pen high
04	Horizontal counter low	Machine status
05	Horizontal counter high	-
06	Vertical counter low	-
07	Vertical counter high	-
08	Scroll register 1	-
09	Scroll register 2	-
0A	Scroll register 3	-
0B	Interrupt acknowledge	-
0C	Screen mode	-
0D	Border colour low	-
0E	Border colour high	-
0F	Colour mask	-
10	Palette index	-
11	Screen end low	-
12	Screen end high	-
13	Memory configuration	-
14	General purpose	-
15	Diagnostic	-
16	Interrupt disable	-
17-1F	-	-
20	-	Blitter destination address 0
21	-	Blitter destination address 1
22	-	Blitter destination address 2
23	-	Blitter source address 0
24	-	Blitter source address 1
25	-	Blitter source address 2
26	-	Blitter status
27	-	Blitter inner counter
28	-	Blitter outer counter
29-2F	-	-
30	Blitter program address 0	-
31	Blitter program address 1	-
32	Blitter program address 2	-
33	Blitter command register	-
34	Blitter control register	-
35-3F	-	-
40	External port 3	External port 1
41-4F	-	-
50	-	External port 2
51-5F	-	-
60-6F	General purpose IO decode 0	
70-7F	General purpose IO decode 1	
80-FF	Free for third party peripheral IO	

The 8088 IO space is internally decoded to eight bits, as shown above. IO locations above 80h are free for third party IO expansion. In addition to this, two general purpose IO decodes are provided on the expansion bus - these may be used to provide an active low chip enables to external devices, note that RD and WR should also be connected to the devices.

All locations marked with a dash "-" are reserved for future expansion or manufacturing test modes. Do not write to any of these locations or use them for external IO.

2.2.1. External Ports

The three external ports in the IO space allow the machine switch controls to be read, and various aspects controlled. These ports are mapped as follows:

Port 1 - read

0	Left foot pedal up
1	Left foot pedal down
	Both the above = light-sensor trigger pressed
2	Joystick 1 fire button A / Slave machine column fire button A
3	Joystick 1 fire button B / Slave machine column fire button B
4	Joystick 1 left / Slave machine right foot pedal down
5	Joystick 1 right / Slave machine right foot pedal up
	Both the above = start button pressed
6	Joystick 1 up / Slave machine right foot pedal down
7	Joystick 1 down / Slave machine right foot pedal up
	Both the above = select button pressed

Port 2 - read

0	Right foot pedal up
1	Right foot pedal down
2	Joystick 2 fire button A / Control column fire button A
3	Joystick 2 fire button B / Control column fire button B
4	Joystick 2 left
5	Joystick 2 right
6	Joystick 2 up
7	Joystick 2 down

Port 3 - write

0	Chair up control
1	Chair down control
2	Chair left control
3	Chair right control
4	Vibrator solenoid enable
5	Joystick port output 1
6	Joystick port output 2
7	Joystick port output 3

On all the inputs to ports 1 and 2, when a 0 is read this indicates that the corresponding switch is closed. Note that three bit pairs on port 1 correspond to mutually exclusive switches, and therefore have an additional function added that corresponds to both switches closed.

Provision is made for attaching a second un-powered machine as a slave controller, by attaching the two machines via the joystick port using a suitable lead. When the machine is wired into this mode, some of the joystick input pins have an alternate function, as described above.

2.2.2. Analogue Inputs

In addition to the switch controls described above, the machine has three potentiometer based analogue controls, and a light-pen type input.

The potentiometer output voltages are measured by comparing them to a voltage ramp which is reset every vertical video synchronization pulse. When the potentiometer voltage equals the ramp voltage an interrupt is generated, one interrupt being generated in every video frame for each of the analogue inputs. When one of these interrupts occurs, the CPU may read the video vertical counter, and from this estimate the analogue voltage being measured. The interrupt mechanism is described in greater detail in the Video Controller section below.

When a second machine is connected in slave mode, its potentiometer values may also be read, as an analogue multiplexer selects between the internal potentiometers, and a second set connected to the joystick port. In this mode, the two sets of potentiometers would be read on alternate video fields. It is also possible that other analogue controls might be attached to the joystick port, and these would be read in a similar manner. The analogue multiplexer control line is in the General Purpose register.

A light-pen, or other beam detecting device such as a gun, may also be connected. This function is part of the Video Controller described below.

2.3. Memory Interface Timing

All memory timing is based on a single master clock rate, which is either four times the European PAL TV standard chroma carrier frequency, or five times the American NTSC TV standard chroma carrier frequency. These give master clock rates of:

NTSC	17.897725 MHz
PAL	17.734475 MHz

This clock rate is divided by 1.5 to produce the co-processor and memory interface clock, nominally 12 MHz. This is further divided by 2 to give the processor clock, nominally 6 MHz.

In the descriptions below timings either apply to both systems, or two numbers are given, the second one in brackets being the NTSC system figure.

Programmers should be aware that CPU performance will be worse in an NTSC system due to the higher display field rate, and should ensure that their code will run satisfactorily on both standards if they wish a world-wide market for their products.

8086

2.3.1. 8088 CPU cycles

All CPU memory cycles take place in four processor clock ticks, with no wait states inserted, in all the types of memory available. (This gives a memory cycle time of 670 ns or about 1.5 MHz.) However, during video data fetch operations and memory refresh operations the video controller asserts the CPU HOLD line throughout.

In a video or refresh line, HOLD is asserted for 44 out of 64 (63.5) micro-seconds. The number of video lines in a display field is under software control, but memory refresh lines occur for two lines in every sixty-four (for four lines in lo-res). A video line and a memory refresh line can occur at the same time. This is discussed further in the video controller section below.

To calculate the available bus time the proportion of lost time in a video line should be multiplied by the proportion of video lines in a field, which is 312 (262). Therefore for a PAL system with 200 lines used for video or refresh, the proportion is:

$$\frac{44}{64} \times \frac{200}{312} = 44\% \text{ lost or } 56\% \text{ available}$$

For an NTSC system the figures are:

$$\frac{44}{63.5} \times \frac{200}{262} = 53\% \text{ lost or } 47\% \text{ available}$$

This section is being redesigned with an 8086 and bus seperation logic that allows operation during video data fetch. This will increase the speed of CPU operation by 3 to 4 times

2.3.2. Co-Processor Cycles

Both the Blitter and the DSP will perform memory cycles in two 12 MHz clock cycles when possible. Co-processor memory cycles into Static RAM and Pseudo-Static RAM (fast RAM) will take two 12 MHz cycles, co-processor memory cycles into ROM and Dynamic RAM will take three 12 MHz cycles (slow RAM or ROM).

The 256K area at 00000h, the screen RAM, will always contain 16-bit fast RAM. The ASIC internal RAM area based at 40000h looks like 8-bit fast RAM (but see notes below about its availability). The area at 80000h will normally contain 8-bit slow RAM (if fitted); and the area at C0000h will normally contain ROM.

Like the CPU, co-processors are affected by video lines; however, unlike the CPU, co-processor memory cycles may continue during video fetch, albeit at a slower rate. During video fetch, four 12 MHz clock cycles out of every eight are occupied by video data fetch operations. The remaining four clock cycles offer a "slot" into which two fast memory cycles or one slow memory cycle may drop. In lo-res, four clock cycles out of every sixteen are occupied, leaving a twelve clock cycle slot. The co-processor will have WAIT states injected into its memory cycles until a slot is available.

Therefore 4/8 or 50% of the bus bandwidth is available to the co-processors in fast RAM during video or refresh lines, and 3/8 or 37.5% is available in slow RAM (more are available in lo-res). Taking the PAL example above, this gives:

$$\frac{4}{8} \times \frac{44}{64} \times \frac{200}{312} = 22\% \text{ lost in fast RAM}$$

This gives the blitter a bus rate of 4.6 million memory cycles per second, i.e. a rate of 9.2 Megabytes per second into 16-bit RAM, giving it the ability to block fill an entire 64K screen in 7.1 milliseconds.

2.3.3. Bus Latency

When a co-processor requests the bus from the CPU, there is a finite period before the bus is granted by the processor HLDA signal, and this period is the bus latency. This latency is particularly relevant to DSP programmers. Unfortunately, the Intel 8088 documentation does not specify a maximum latency, but it is believed to be safe to assume a maximum of six processor clock cycles, which is twelve co-processor clock cycles.

If the video controller owns the bus, then hold acknowledge will be issued immediately. The DSP will have to wait until the blitter finishes its current memory cycle if the blitter is running. However its memory cycles may then be subject to wait states until a slot appears.

For the benefit of DSP programmers, here is a summary of the bus latencies:

Video	Blitter	Latency	Notes
off	off	12	Given by the 8088 HLDA response time
on	off	0	But will be held in WAIT until a slot
off	on	3	Time for blitter to complete cycle
on	on	3+	As above, but subject to WAIT

It can be seen that the first case is dominant. Note that DSP programmers will have to add the memory cycle times into their total data latency calculations, and they should consider the worst case to be when they request the bus at the same time as the video, and are therefore subject to WAIT states on top of the 8088 latency.

THE VIDEO CONTROLLER

1. INTRODUCTION

This document describes the SLIPSTREAM video block. The video block has the following functions :-

1) To provide memory timing on behalf of the processors
2) To generate video and refresh cycles
3) To generate video timing and support gen-locking
4) To convert video memory contents to pixel data.
5) To generate an interrupt at a given display line.
6) To latch the presence and position of a light-pen signal
7) To control data and address bus buffers
8) To vector interrupts
9) To provide IO to slipstream peripherals

2. THE SCREEN MAP AND DISPLAY MODES

The screen may begin on any 256 byte boundary, or 128 byte boundary in lo-res, in 16 bit memory, (0-256k). The number of screen lines is programable between 1 and 256 lines. The address of the top left pixel is given by the vertical and horizontal scroll registers. The horizontal scroll register gives the least significant eight bits. The vertical scroll register gives the most significant ten bits.

Two television standards are supported: non-interlaced 312 lines at 50Hz (PAL) and non interlaced 262 lines at 60Hz (NTSC). The standard is determined by the state of the JOYL0/PAL pin during reset.

For PAL a crystal frequency of 17.734475 MHz is required. For NTSC a frequency of 17.897725 MHz is needed. The crystal clock is divided by 1.5 to give a frequency referred to as 12 MHz for convenience.

Three display resolutions are supported - low, medium and high.

In low resolution the pixel rate is 6Mhz and pixels are represented by nibbles. In medium resolution the pixel rate is 6MHz and pixels are represented by bytes. In high resolution the pixel rate is 12Mhz and pixels are represented by nibbles.

The mapping between pixels and memory addresses is as follows.
For convenience the horizontal scroll register contains 34 and the vertical scroll register contains 12. The top left pixel comes from address 01234.

> 1234.0 refers to the less significant nibble at address 1234.
> 1234.1 refers to the more significant nibble.

2.1. Low resolution 200 line display

<table>
<tr><td>1234.0</td><td>1234.1</td><td>1235.0</td><td>1235.1</td><td>- -</td><td>12b2.1</td><td>12b3.0</td><td>12b3.1</td></tr>
<tr><td>12b4.0</td><td></td><td></td><td></td><td>- -</td><td></td><td></td><td></td></tr>
<tr><td>1334.0</td><td></td><td></td><td></td><td>- -</td><td></td><td></td><td></td></tr>
<tr><td>|</td><td>|</td><td>|</td><td>|</td><td>- -</td><td>|</td><td>|</td><td>|</td></tr>
<tr><td>7534.0</td><td></td><td></td><td></td><td>- -</td><td></td><td></td><td>75b3.1</td></tr>
<tr><td>75b4.0</td><td>75b5.1</td><td></td><td></td><td>- -</td><td></td><td></td><td>7533.1</td></tr>
</table>

2.2. Medium resolution 200 line display

1234	1235	1236	1237	- -	1231	1232	1233
1334				- -			
1434				- -			
|	|	|	|	- -	|	|	|
d834				- -			d833
d934	d935			- -			d933

2.3. High resolution 200 line display

1234.0	1234.1	1235.0	1235.1	- -	1232.1	1233.0	1233.1
1334.0				- -			
1434.0				- -			
|	|	|	|	- -	|	|	|
d834.0				- -			d833.1
d934.0	d935.1			- -			d933.1

The number of screen lines is determined by the SIZE register.

2.4. Logical to Physical Colour Translation

The physical colour is made from four bits of red, green and blue. The logical colour is determined by either a byte or a nibble in memory. The mapping between logical and physical colour is performed by a colour look up table or palette. Any bits in the logical colour may be sacrificed and replaced by bits from the palette index register. The unused (or masked) bits may be used elsewhere, e.g. by the blitter for depth information. The most significant bit of each pixel may also be used by the video block. For every bit set in the pixel mask register the corresponding bit in the logical colour is replaced by a bit from the palette index register.

Where the logical colour is derived from a nibble in memory the top four bits are deemed to be zeroes. The mask register would normally mask out these bits in this case; allowing the palette index register to select one of 16 palettes.

The most significant bit of the logical colour can be made available on the encrustation output to embed the locally generated picture onto an external video source. (Bit set selects the local picture).

It can be used in variable-mode to determine whether the current byte is displayed as one med-res pixel or two hi-res pixels. (Bit 7 set two hi-res pixels are generated).

The encrustation output may also be used to strobe frame grabbing AD convertors.

The palette memory appears in the memory map of the host processor from 40000 to 401FF. Each word has the following meaning.

MSB	--	--	--	--	r3	r2	r1	r0	g3	g2	g1	g0	b3	b2	b1	b0	LSB

Where r3 is the most significant red bit of the physical colour. The word at address c0000 corresponds to logical colour 0, The word at address c0002 corresponds to logical colour 1 etc.

The palette should only be read or written while the display is inactive.

2.5. Colour Hold Mode

In colour hold mode logical colour zero is elected to be 'transparent'. This colour is replaced by the previously displayed non-transparent colour. This display scheme may be used to fill large areas with colour by simply drawing their outline.

Note that the left side should be drawn in the fill colour, and the right side should be drawn in the required background colour. If the first pixel displayed on a line is transparent, then the colour held is undefined.

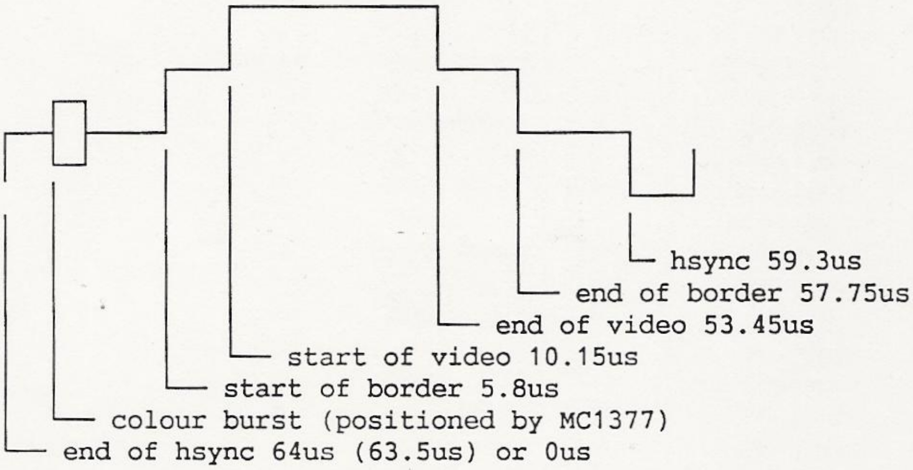
2.6. Border Colour

A border register determines the physical colour of picture outside the computer screen.

3. SCREEN TIMING

3.1. Video Line Timing

The video line timing is based on the '12Mhz' clock and is 64 us in PAL mode and 63.5 us in NTSC mode. The timing is shown below:-



The sync pulse is generated last so that gen-locking can be achieved simply by resetting the horizontal counter with the sync from an external source.

3.2. Video Field Timing

A display is built up by a sequence of video lines into a field. This contains 312 lines in PAL mode and 262 lines in NTSC mode. These are displayed at 50 Hz in PAL mode and at 60 Hz in NTSC mode. (This gives a non-interlaced display, unlike broadcast TV.)

The number of video (pixel) display lines in a field is under software control, and on lines that are not part of the video display, the border colour is displayed.

Not all the actual display lines in the field are visible. Some are occupied by "flyback", a period in which the beam is moved from the bottom of the screen to the top, and some are lost at the top and bottom due to "overscan", which occurs as the picture fills the whole screen and overflows it slightly (in broadcast TV some of these are used for teletext).

In PAL mode, the maximum of 256 lines will all be visible. However, software that takes advantage of this will not be compatible with the NTSC standard as some of these lines are lost to overscan on an NTSC set. Therefore the maximum number of lines used should be rather less.

A sensible maximum is about 200 lines visible; for example the MSX standard allows 212, and this should probably be treated as a maximum.

4. THE REGISTERS

There are 32 I/O locations used by the video block

Address	Write Register	Read Register
00h	Interrupt low INTL	H. Light-pen low HLPL
01h	Interrupt high INTH	H. Light-pen high HLPH
02h	Start low STARTL	V. Light-pen low VLPL
03h	Start high STARTH	V. Light-pen high VLPH
04h	Hcount low HCNTL	STATUS
05h	Hcount high HCNTH	- reserved -
06h	Vcount low VCNTL	- reserved -
07h	Vcount high VCNTH	- reserved -
08h	Scroll register 1 SCROLL1	- reserved -
09h	Scroll register 2 SCROLL2	- reserved -
0Ah	Scroll register 3 SCROLL3	- reserved -
0Bh	Interrupt ACK	- reserved -
0Ch	Screen MODE	- reserved -
0Dh	Border low BORDL	- reserved -
0Eh	Border high BORDH	- reserved -
0Fh	Colour MASK	- reserved -
10h	Palette INDEX	- reserved -
11h	End low ENDL	- reserved -
12h	End high ENDH	- reserved -
13h	Memory configuration MEM	- reserved -
14h	General purpose GPR	- reserved -
15h	Diagnostics DIAG	- reserved -
16h	Interrupt disable DIS	- reserved -
17h-1fh	- reserved -	

4.1. Interrupt registers

INTL	00h
INTH	01h
ACK	0Bh

INTL and INTH determine the screen line at which an interrupt is generated. The interrupt is generated just after the active video area on that line. The low eight bits of the line number are given by INTL, the top bit is bit 0 in INTH. Zeroes should be written into the top seven bits of INTH. This interrupt is associated with interrupt vector 21h. The interrupt is cleared by writing anything to the ACK port.

4.2. Start registers

STARTL	02h
STARTH	03h

These two registers determine the screen line at which the video display begins. The low eight bits of the line number are given by STARTL, the top bit is bit 0 in STARTH. Zeroes should be written into the top seven bits of STARTH. The display begins on the line **AFTER** the specified line.

4.3. Horizontal count registers

HCNTL	04h
HCNTH	05h

When a byte is written into HCNTL the horizontal time-base counter is loaded with this value, and the value in bits 0 & 1 of HCNTH. This is mainly useful for test purposes.

In PAL mode the counter counts up from 0 (just after hsync) to 756 (the end of hsync). The active display area is from 120 to 631 inclusive. The counter counts in units of one hi-res pixel (11.82MHz).

4.4. Vertical count registers

VCNTL	06h
VCNTH	07h

When a byte is written into this VCNTL the vertical time-base counter is loaded with this value, and the value in bit 0 of VCNTH. Again this is mainly useful for test purposes.

In PAL mode the counter counts up from 0 (just after vsync) to 311 (the end of vsync). The active display area is from 33 to 288 inclusive. The counter counts in display lines (15.625Khz).

4.5. Scroll registers

SCROLL1	08h
SCROLL2	09h
SCROLL3	0Ah

These three registers determine the address, within the first 256k of memory, of the top left pixel. SCROLL1 gives the bottom eight bits, SCROLL2 gives the next eight bits and bits 0 & 1 of SCROLL3 give the top two bits. Zeroes should be written to the top six bits of SCROLL3.

The screen must begin on a word boundary, so the bottom bit of SCROLL1 is ignored. Note also that the data for one line will lie on a 128 or 256 byte boundary depending on the screen mode, and therefore the line will wrap in this area if the start is not on the boundary.

4.6. Screen mode register

MODE 0Ch

Bits 0,1 Determines the screen mode, 0 is low resolution, 1 is medium resolution, 2 is high resolution, 3 is unused.

Bit 2 Enables gen_locking. When enabled hsync and vsync become inputs and the video time-base generators are reset by these inputs.

Bit 3 Enables encrustation. If zero the local display is enabled. If one then the local pixels are enabled if the most significant bit of each pixel is set.

Bit 4 Enables encrustation of the border. If encrustation is enabled this bit enables the local border colour.

Bit 5 Enables colour hold mode.

Bit 6 Enables variable resolution mode.

Bit 7 Enables the screen to cross 64k boundaries. This may be used if more than 64k of screen ram is installed.

4.7. Border colour registers

BORDL	0Dh
BORDH	0Eh

These registers give the physical border colour. The registers define red, green and blue intensities like the palette RAM.

	BORDH								BORDL								
MSB	--	--	--	--	r3	r2	r1	r0	g3	g2	g1	g0	b3	b2	b1	b0	LSB

Zeroes should be written to the top four bits of BORDH.

4.8. Colour mask and palette index

MASK	0Fh
INDEX	10h

This register allows bits in the logical colour of a pixel to be sacrificed for other purposes. For each bit set in the mask register the corresponding bit in the logical colour is replaced by a bit from the palette index register. For instance say the most significant bit is used by the blitter for depth information. The mask register would be set to 80h giving two palettes of 128 colours selected by the most significant bit of the palette index register.

4.9. End registers

ENDL	11h
ENDH	12h

These registers specify the last active display line, in a similar manner to the start registers.

4.10. Memory configuration

MEM	13h

This register determines the type of memory between 512k and 1024k.

Bits 0,1 These determine the type of memory between 512k and 768k as follows:-
0 is ROM (default)
1 is DRAM
2 is SRAM
3 is PSRAM.

Bits 2,3 These determine the type of memory between 768k and 1024k as above.

Bits 4-7 These should be programmed with zeroes.

4.11. General purpose register

GPR	14h

This register provides general purpose output.

Bit 0 Analogue input select. When set this selects the second set of analogue inputs.

Bit 1 This enables the DSP output.

Bits 2-7 These should be programmed with zeroes.

4.12. Diagnostic

DIAG	15h

Bit 0 When set this bit lets the light-pen registers reflect the current position of the electron beam. This bit is cleared by reset.

Bit 1 When set NTSC mode is used. This is set or cleared during reset and may be read in the status port.

Bit 2 When set the vertical counter advances every clock cycle rather than at the end of every line.

Bit 3 When set the internal clock is the same as the XTAL pin (rather than divided by 1.5).

Bits 4 When set the screen mode is updated immediately when written and the palette is made transparent.

Bits 5-7 These should be programmed with zeroes.

4.13. Interrupt disable register

DIS	16h

This register selectively disables the interrupt sources.

Bit 0 Disables the video interrupt.
Bits 1-3 Disable analogue inputs 0 to 2.
Bits 4-7 Reserved, write zero.

4.14. Light-pen registers

HLPL	00h
HLPH	01h
VLPL	02h
VLPH	03h

These registers determine the current position of the light-pen or electron beam depending on the state of bit 0 in DIAG.

HLPL gives the low eight bits of the horizontal count. The top two bits are given by bits 0 & 1 of HLPH. Bits 2-7 of HLPH are zero.

VLPL gives the low eight bits of the vertical count. The top bit is given by bit 0 of VLPH. Bits 1-7 of VLPH are zero.

Light-pens produce well defined rising edges but slow decays and respond to several display lines. The value latched is the value at the leading edge of the first pulse each frame. A latch is cleared at the start of each frame (at the end of vsync) and is set by the arrival of the light-pen pulse. The state of this latch can be read in the status register.

4.15. Status register

STAT	05h

Bit 0 This bit is set if the machine is configured for NTSC.
Bit 1 This bit is set if a light pen pulse has been received.
Bit 2 This bit is set if the processor is working at 9MHz.
Bits 3-7 Reserved

5. OPERATION OF THE CONTROLLER

5.1. Refresh mechanism

Dynamic and Pseudo static memory are refreshed by the video fetch mechanism. DRAM is refreshed by one CAS before RAS cycle for each four bytes fetched. PSRAM is refreshed by two OE cycles. Because the screen size may be adjusted by the program the video fetch mechanism is always active on certain lines. In low res. it is active for four lines every 64 lines; ie 0-3,64-67,128-131 etc. In medium and high res. it is active for two lines every 64 lines.

5.2. Interrupt mechanism

There are four interrupt sources within the slipstream chip. One video interrupt and three interrupts from the comparator inputs. Each has its own vector. The video interrupt uses vector 21h, the comparators use vectors 22h to 24h. If an external interrupt occurs vector 20h is supplied. Interrupts with lower vectors have higher priority. Interrupts are cleared by writing to the ACK port. If several interrupts occur together each will generate an interrupt and each must be acknowledged.

5.3. A/D Comparators

In order to measure analogue voltages economically three comparator inputs are provided. The external comparators should compare the voltage to be measured against a ramp triggered by vertical sync. Negative transitions cause a vectored interrupt allowing the 8088 to read the line number and estimate the voltage.

5.4. Peripheral IO

Three IO locations relate to predefined ports outside the Slipstream chip.

Port 1	Input	40h
Port 2	Input	50h
Port 3	Output	40h

Two IO locations are decoded for general purpose use:-

GPIO 0	I/O	60h
GPIO 1	I/O	70h

All undefined IO locations between 0 and 7Fh are reserved for future expansion. IO locations above 80h are available for third party peripherals.

5.5. Bootstrap

After the two microcomputers in the computer and in the ROM cartridge have established contact and released reset execution begins at address FFFF0h. ROM cartridges should be enabled by csL[1] and should contain suitable code at this location.

DIGITAL SOUND PROCESSOR

1. INTRODUCTION

The DSP system in Slipstream is a general purpose arithmetic processor with sufficient power to implement a high performance music synthesizer. Pulse width modulated outputs are provided for generation of stereo audio signals with 14 bit precision and hence approaching the sort of sound quality normally associated with Compact Disk technology.

The DSP is micro-programmable from the host CPU and the instruction set is sufficiently flexible to enable the user to program the device to fulfill many different functions that are considerably at variance with that of 'music synthesizer'. Such applications might include algorithmic speech generation and audio analysis using fast fourier transform techniques.

The DSP uses Harvard Architecture for maximum data throughput (separate program and data buses).

The ALU features a hardware 16 x 16 hardware multiply/accumulate as well as addition, subtraction and logical functions. The carry bit from the adder/subtractor is stored in a separate latch and can be either used either to propagate carry for multiple precision arithmetic or can be used for conditional instructions. All instructions may be made to be dependant on this bit being set.

Data transfers within the device (with the exception of internal transactions within the multiplier/accumulator) are all 16 bit.

2. MEMORY

Program RAM available is 256 x 16 bit words.

Data space is organised as follows:

000 - 0FF	256 x 16 full wave SINE table in ROM
100 - 13F	a few common constants in ROM/Hardwired logic:

100	0000h
101	0001h
102	0002h
103	0004h
104	0008h
105	0010h
106	0020h
107	0040h
108	0080h
109	FFFFh
10A	FFFEh
10B	FFFCh
10C	8000h
10D - 13F	Not implemented

140-17F	internal registers:

140	Intrude data register
141	Index register
142	DMA address 0. Writing to the location initiates the DMA transfer, as specified by the control bits.
143	DMA address 1 and control bits:

D11	D10	D9	D8					D3	D2	D1	D0
HOLD	RW	BW	LOHI	-	-	-	-	A19	A18	A17	A16

HOLD if set to ONE, the DSP will assert HOLD to the host, until the HOLD bit is set to ZERO again.

RW Read/Write. If RW is set to ONE the following DMA transaction(s) will be READ.

BW Byte/Word. If BW is set to ONE the following DMA transaction will be a byte transfer. If a word transfer then the least significant bit of DMA0 will be ignored.

LOHI if set to ONE and the following DMA transaction is a byte transfer, the transfer will take the high byte of the DMA data register.

144	DMA data
145	Multiplier result 0
146	Multiplier result 1
147	Multiplier overflow bits and ALU carry flag:

D4	D3	D2	D1	D0
CARRY	Z35	Z34	Z33	Z32

CARRY is the ALU carry latch
Z[35..32] are the most significant four bits of the multiplier/accumulator result register. During multiply operations carry is set according to bit 36 of the accumulator result.

148 PWM Dac LEFT (write only)
149 PWM Dac RIGHT (write only)
14A Program counter (note that this can only be written to by a DSP program. When the DSP is in STOP mode, the program counter is loaded with the value on the host address bus).
14B ALU Mode register:

D6	D5	D4	D3	D2	D1	D0
TCX	TCYN	M	S3	S2	S1	S0

TCX If TCX is set to ONE then the X register will be treated as two's complement number for all multiply operations.
TCYN If TCYN is set to ZERO then the value (nn) will be treated as a two's complement for MULT and MAC instructions. Note that it has the inverse sense to TCX.
M & S3 to S0 are as described in the TI data book for the 74181, and in an appendix at the rear of this manual.

14C ALU X register
14D ALU Z register (result)
14E Intrude address register.
14F DSP IO pin. If this location is read then bit 0 will return the logical sense of the DSP input. If the output is enabled (this can be done by setting bit 1 on the general purpose port GPR at 14h) then the output can be written to by the DSP.
150-17F Not implemented - returns zero

---

180 - 1FF 128 x 16 words of RAM

---

3. DSP INSTRUCTION SET

Each instruction has a 7 bit opcode and a 9 bit address vector.

All micro-coded instructions are completed in one 85ns cycle. All instructions are memory to register transfers or register to register transfers. Immediate values are not allowed. If a constant is needed in programming and it is not available in the constant table, then a data RAM location must be set aside for the value.

3.1. Instruction Codes

3.1.1. Condition Bit (if CARRY)

If bit 10 in the instruction code is SET, then the command is ONLY implemented IF the CARRY bit in the ALU is also set.

3.1.2. Indexed Addressing

If bit 9 in the instruction code is SET, then the 9 bit address vector in the instruction code is added to the 9 bit value in the INDEX register IX.

3.1.3. Opcodes

Code	Mnemonic	Description
00	MOV (nn), MZ0	Move data from the least significant word of the multiplier result register into data location nn.
01	MOV (nn), MZ1	Move data from the most significant word of the multiplier result register into data location nn.
02	MOV MZ0, (nn)	Move data word from data location (nn) to least significant word of multiplier result register.
03	MOV MZ1, (nn)	Move data word from data location nn to most significant word of multiplier result register.
04	CCF	Complement the carry flag.
05	MOV DMA0, (nn)	Move data from (nn) to the least significant word of the DMA address register. Executing this instruction will start the DMA transaction in the manner described by the DMA1 register. The programmer must allow the minimum latency time to expire between two consecutive instructions of this type.
06	MOV DMA1, (nn)	Move data from data location (nn) to the most significant bits of the DMA address register.

		Control bits are also passed with this instruction. See note on DMA transfer.
07	MOV DMD, (nn)	Move data from nn to DMA data register.
08	MOV (nn), DMD	Move data from the DMA data register to nn.
09	MAC (nn)	Multiply and Accumulate. Multiply the contents of nn by the contents of the X register and add the result to the number held in the multiplier result register.
0A	MOV MODE, (nn)	Move data from location nn to the ALU mode register. This register defines: a) the operation to be performed with the General Arithmetic Instruction. b) Selects signed or unsigned multiply operations.
0B	MOV IX, (nn)	Move data from (nn) to the index register.
0C	MOV (nn), PC	Move data from Program counter to nn (useful for relative jumps, computed GOTOs etc).
0D	MOV X, (nn)	Move data from nn to ALU x register.
0E	MOV (nn), X	Read data from ALU x register to data location (nn).
0F	MULT (nn)	Multiply the x register by the data in (nn).
10	ADD (nn)	Add data in location (nn) to data in ALU x register and place result in the ALU z (result) register. Ignore carry flag but set it afterwards according to result.
11	SUB (nn)	Subtract data in nn to data in ALU x register. Ignore carry flag but reset it if the result overflows.
12	AND (nn)	Perform logical AND between data location (nn) and ALU x register.
13	OR (nn)	Logical OR between the data in nn and the ALU x register.
14	ADC (nn)	Add the data in nn, the ALU x register and the carry bit together. Carry flag will be updated.
15	SBC (nn)	Subtract data in nn and the carry flag from the ALU x register. Carry flag is updated.
16	MOV (nn), AZ	Move the data from the ALU result register to nn.

17	MOV AZ,(nn)	Load the result register directly from location nn
18	MOV (nn), Z2	Load nn with the MAC overflow bits and the ALU carry bit.
19	MOV DAC1, (nn)	Data from nn is passed to the LEFT D to A converter.
1A	MOV DAC2, (nn)	Data from nn is passed to the RIGHT D to A converter.
1B	MOV DAC12, (nn)	Move data from nn to BOTH D to A converters.
1C	GAI (nn)	General Purpose Arithmetic Instruction. Implements the operation defined by the code loaded into the MODE register.
1D	MOV PC, (nn)	Jump instruction. Note that because instructions are pipelined the instruction immediately following the jump instruction in program RAM will be executed before the one at the location specified by the contents of nn.
1E	NOP	Null operation.
1F	INTRUDE	At least one INTRUDE instruction must be put into the dsp program if any data transfer is to take place between host and dsp under host control (see note).

4. HOST MEMORY MAP

All of the internal registers are available to the host. All of the internal data memory is available to the host via the INTRUDE function, although the program memory is only available to the host during the DSP STOP mode.

41000 - 411FF DATA ROM - This contains a SINE table for sound generation and TRIG functions.

41200 - 4127F DATA Constants - a few useful hardwired constants for use in DSP programs.

41280 - 412FF INTERNAL REGISTERS - for maximum visibility and debugging purposes, all the internal registers may be inspected by the host.

41280	Intrude data register
41282	Index register (9 bits are valid)
41284	DMA address 0
41286	DMA address 1 (8 bits are valid including mode bits - HOLD, R/W,B/W, LO/HI)
41288	DMA data
4128A	Z0 MAC result register
4128C	Z1 MAC result register
4128E	Z2 MAC overflow & carry (5 bits valid)
41290	DAC Left (14 bits valid - D2 to D15, write only)
41292	DAC Right (14 bits valid - D2 to D15, write only)
41294	Program counter (8 bits are valid)
41296	Mode register (6 bits are valid)
41298	ALU X register
4129A	ALU Z register
4129C	Intrude address register (8 bits are valid). Note that reading this register from the host is somewhat meaningless as it can only return its own address.
4129E	DSP IO pin (D0 is valid)

Un-implemented data bits within the above registers are set to zero during DSP read operations.

41300 - 413FF DATA RAM

41400 - 415FF PROGRAM RAM

41600 Run/Status register.

Bit 4 in the write register controls RUN/STOP of the DSP. Writing a ONE into this bit sets the DSP to RUN. Writing any other value to this port causes the DSP to single step.

Bits 0-3 of the READ register are the bits of the state machine controlling the INTRUDE mechanism. All three bits need to be zero in order for the host to initiate any INTRUDE transaction. If bit 0 is set then the host has only completed one byte of a two-byte transfer.

Bit 4 reads the RUN/STOP condition.

5. DATA TRANSACTIONS BETWEEN HOST AND DSP

5.1. Under DSP control - DMA transfer

In order to transfer data between the host memory and the DSP a DMA transfer needs to be set up under DSP program control. A typical transaction might be as follows:

```
     MOV  DMA1,(topbits) ; This instruction is used to set the top four
                           bits of the host address (A19 - A16), the
                           direction of the transfer (RW), the number of
                           bits to be transferred (BW), and to put the
                           host processor into HOLD mode so that the DSP
                           can have access to the host bus.
```

; After a number of instruction cycles (see discussion earlier), the host will have released the bus and the DSP can have direct access to the locations in host memory that it needs.

```
     MOV  DMA0,(addr1)   ; Here the rest of the address is defined and
                           the state machine which implements the data
                           transaction  is set off.
```

; After a further number of instruction cycles (depending on the type of memory, see discussion earlier) the data will be transferred and the DSP is free to collect the new data from the DMA data register or to place a new word in it.

```
     MOV  (data),DMD
```

; If more than one word is needed then the DSP already has the bus.

```
     MOV  DMA0,(addr2)
```

; After an appropriate number of cycles to allow the state machine to collect the data....

```
     MOV  (moredata),DMD
```

; Finally the host bus must be released as soon as possible.

```
     MOV  DMA1,(release) ; where the HOLD bit in "release" is zero.
```

5.2. Under Host Control - INTRUDE

All the internal memory is mapped into the HOST address space. When the DSP is in STOP mode, the host may read these locations just as if it were normal HOST memory.

When the DSP is running, however, the Program memory is not available to the host and the data memory is only available by the INTRUDE mechanism.

In order that DSP operations are not disturbed in any way, data transactions can only take place when the DSP is executing INTRUDE instructions.

5.2.1. Host to DSP transfer

1. Ensure that the INTRUDE state machine is ready to accept new data. This is indicated by bits 0 - 2 in the status register (host location 600h) being at zero. If the DSP program length or the maximum time between INTRUDE cycles is known, then it is legitimate for the host to wait this amount of time before being sure that the DSP is ready.

2. Write data to the memory location as normal

5.2.2. DSP to Host transfer

1. Ensure that the state machine is ready (as above).

2. Read the memory location. The data read will be the data from the last INTRUDE location and not the desired data. The action of this read is to set up the address for the INTRUDE state machine to fetch the new data from.

3. Wait for the state machine to become ready again.

4. Read the data (If it is known what location the data will be required from next, then the operation can be reduced to 2 steps).

THE BLITTER

1. INTRODUCTION

1.1. This Section

This section describes the graphics co-processor, or blitter, of Slipstream. It explains its modes of operation and how these are controlled by the programmer.

The intention of this description is to allow a competent programmer to drive the blitter hardware directly.

The blitter is a complex device, and to drive it properly requires a pretty clear understanding of how the hardware works. This is all explained here, but a knowledge of some aspects of digital hardware design is assumed.

1.2. The Blitter

The blitter is a co-processor of the Slipstream system, whose purpose is to perform graphics creation and animation as fast as possible (limited by the memory bandwidth). It can perform arbitrarily long sequences of graphics operations by reading new command sets from memory.

While it is performing any graphics operations, the blitter becomes a bus master, and denies the CPU any bus activity whatsoever. This is reasonable behaviour because the blitter is being used to perform operations that the CPU would otherwise have performed, and is therefore speeding up program operation. This also removes the need for any asynchronous control programming for blitting operations, and the need for any interrupt generation hardware in the blitter.

However, to allow real-time programming of either of the other two processors, the blitter will grant the bus to the DSP DMA channel, and to the CPU if an interrupt occurs. During either of these the current operation is suspended but will restart when the interrupt goes away or when the DSP DMA access completes.

The blitter may be operated in a variety of modes to perform graphics and block move operations. These are described in the sections that follow.

2. ARCHITECTURE

This section discusses the architecture of the blitter, so that its modes of operation can be understood.

2.1. Overview

The internal architecture of the blitter is best considered as three, largely separate, blocks. These are the Data Path, the Address Generator, and the Sequencer.

The sequencer acts in software terms as the program that the blitter runs, with two **for** loops and a couple of procedures. This **program** is fixed, although various parts of its operation are conditional on flags in the blitter command, and the loop counts are also part of the command.

The address generator contains three address registers. These are the program address used to fetch blitter commands, and the source and destination address registers. It also contains an ALU with an associated step register to update addresses, and a multiplexer to generate the output address.

The data path contains three data registers - the source data, destination data and pattern data registers. It also contains a versatile comparator to allow intelligent blitting operations, and a logic function unit to generate the output data.

These sections are described below.

2.2. The Data Path

The Data Path contains three data registers, and two data manipulation blocks. These are the Logic Function Unit, which can combine the contents of the data registers in a number of ways to produce the output data; and the comparator, which can perform certain comparisons on the data to inhibit write operations, and optionally stop blitter operation.

The data path can handle data in three quanta, 16, 8 and 4 bits. Words (16 bits) are used when performing fast block moves and fills, and pixels (8 or 4 bits) may be manipulated using all the blitter modes, such as line-drawing, multiple plane operations, character painting and so on.

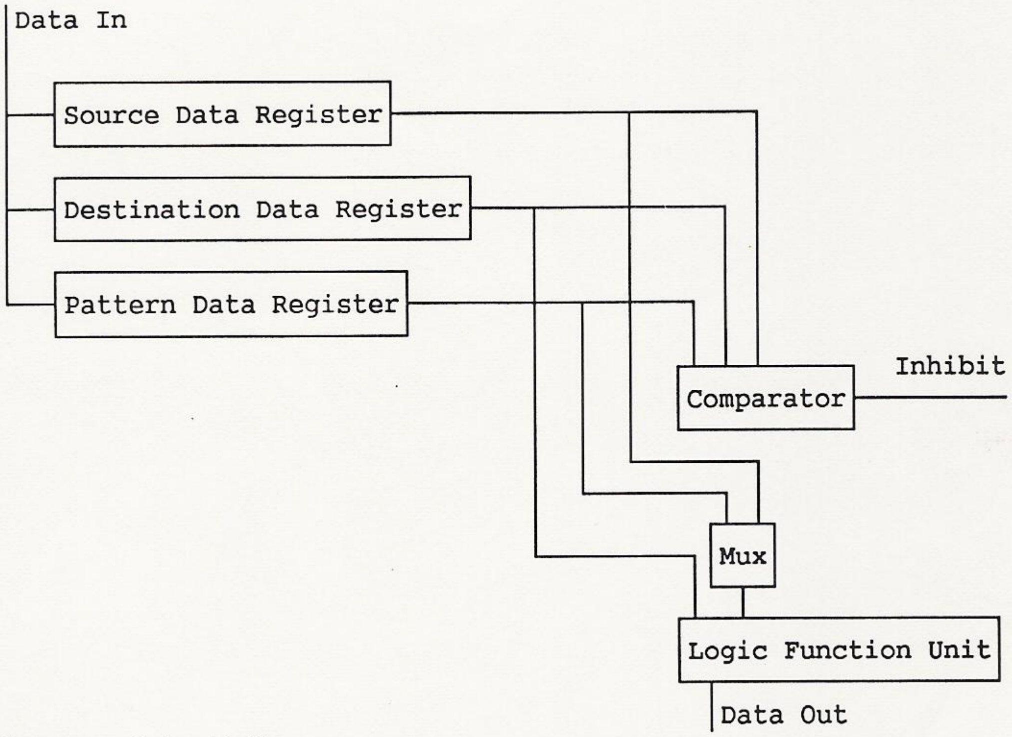


Figure - Data Path

2.2.1. Data Path Width

The majority of the data path is eight bits wide as this is the maximum pixel size. However the source data register is sixteen bits wide, and the top 8 bits of this are used to produce the top 8 bits of the data written in sixteen bit mode.

2.2.2. Registers

There are two eight bit wide and one sixteen bit wide data registers. These are the Pattern, Destination and Source Data registers respectively.

The Source and Destination data registers are loaded from the Source and Destination addresses in memory when the corresponding read cycles are enabled in the Inner Loop (see below). However, all three data registers are loaded at the start of blitter operation with the pattern data, and this may be used as an additional source of data either in producing the output data or in the comparator. For example it may be a mask, a pattern for writing, a reference value, and so on. (The pattern data is loaded into both bytes of the source data register)

2.2.3. Logic Function Unit

The Logic Function Unit (LFU) generates the output data, which is written to the destination in memory. It can perform any logical combination of the source and destination pixels (in this case, 'source' data may be selected from either of the source or pattern data registers).

It does this by selecting any of the four boolean minterms[1] of the two sets of input data from the data registers, and generating the logical OR of the selected terms. This allows any logical combination of the of input data, there are therefore sixteen possibilities.

In sixteen bit mode, the LFU will normally be set to produce source data, as it is only eight bits wide, the top eight bits being source data directly.

2.2.4. Comparator

The comparator can perform a variety of comparisons on the data in the source, destination and pattern data registers. !!!!!! If its comparison conditions are met then it generates an inhibit signal. This can be used to inhibit a write operation, and optionally to stop blitting operation.

The comparator may be used to provide a pixel plane effect, to give transparent colours, for collision detection and memory search operations, and as an aid to character painting.

[1] The minterms of a set of logical inputs are the set of AND terms given by each of the inputs combined together in all combinations of complimented and normal. Therefore the minterms of two inputs A and B are:

A AND B, A AND (NOT B), (NOT A) AND B and (NOT A) AND (NOT B)

Any boolean combination of A and B can be given by choosing a suitable set of minterms to OR together.

2.2.4.1. Comparator Pixel Plane Operation

Multiple plane operation is supported by assigning a plane number to every pixel. In eight-bit pixel mode two of the eight bits are used giving two or four planes, in four-bit pixel mode one of the four bits is used giving two planes.

The bits are allocated within the byte thus:

Bit	Plane Function	
	8 bit pixel	4 bit pixel
0		
1		
2		
3		Plane bit
4		
5		
6	Plane bit 0	
7	Plane bit 1	Plane bit

The comparator can produce an inhibit output if the plane number of the destination data is equal to, not equal to, or greater than the plane number of the source data, or any combination of these. This means that data being written onto the screen can be masked by data already present in a different plane.

2.2.4.2. Comparator Pixel Value Comparison

The comparator can produce an inhibit output if the entire source pixel is equal to or not equal to the destination byte. This may be used for searching memory for a particular value; and, more importantly, for designating a colour to be transparent, and holding the transparent colour value in a data register.

2.2.4.3. Comparator Bit to Byte Expansion

This comparator operation allows bit to pixel expansion of data, used for example for character painting. In this mode the comparator selects a bit of the source byte based on the value of the inner counter (refer to the Sequencer Description), and inhibits the write operation if this bit is zero.

2.2.5. Handling of Data in Four and Eight Bit Pixel Modes

The blitter makes provision for handling two screen pixel resolution modes. These are eight bit mode, where each byte corresponds to one pixel, and for bit mode, where each byte corresponds to two pixels.

In eight bit pixel mode the data path is handling one pixel at a time, and this makes its operation straightforward. In four pixel bit mode, however, only half of the byte that is read from or written to is the current pixel so certain additional requirements are placed on the data path.

In a four bit mode write operation unchanged destination data is written to the half of the data byte that does not correspond to the current pixel. This means that write-only blitter operation will not work properly in this mode, i.e. control bit DSTEN must be set.

It is also possible that the source four bit pixel address and the destination four bit pixel address point at different halves of a byte. If this is the case, a shifter swaps the two halves of the source data.

This is all handled transparently to the user. However certain caveats should be noted:

1. In four bit mode, destination reads must always be performed. If this is not done then the pixel in the other half of the destination byte will be corrupted.

2. In four bit mode, the two nibbles of the pattern byte should normally be set to the same value.

Note, however, that the blitter may be set to eight bit mode when the display is in four bit mode, to perform operations in two pixel multiples, and this will speed up certain operations (but not as much as word mode where this is applicable).

2.3. The Address Generator

The address generator contains three address registers, an increment or step register, an address adder, and an address output multiplexer.

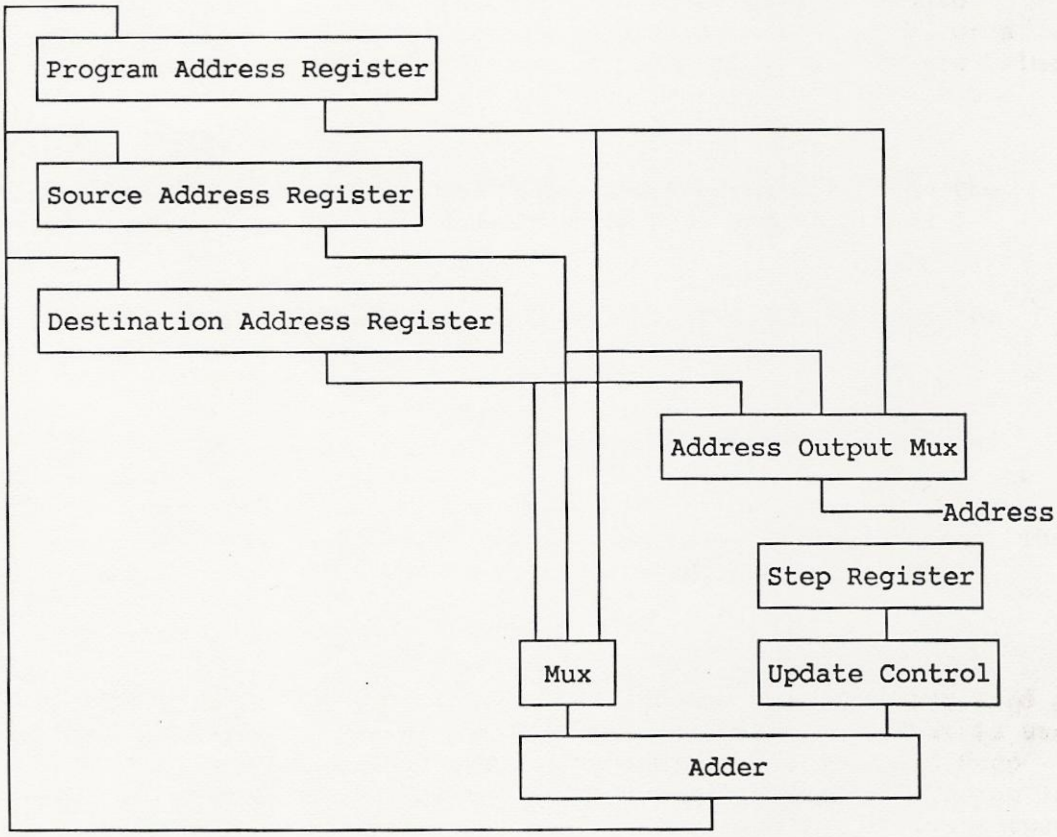


Figure - Address Generator

2.3.1. The Address Registers

There are three address registers holding the source address, the destination address, and the program address.

Each of these registers contains twenty address bits, referred to as bits 0 to 19, allowing the blitter to address up to One Megabyte. In addition to this the source and destination address registers contain a nibble bit used in high resolution mode, designated bit -1.

The Program address register holds the address that the program is fetched from, and is incremented by one each time a memory cycle is performed using it.

The Source and Destination Address registers are updated after each cycle, and at other times, using an adder that allows them considerable flexibility in the objects to which they refer.

All source and destination address updating may be performed optionally on just the bottom sixteen bits of the address register. This means that the blitter will then effectively operate in 64K pages. In this mode if an address overflows in a page, it will wrap, and the overflow (or underflow) will be lost.

2.3.2. The Address Adder

The address adder is a twenty-one bit wide adder used to update addresses. It allows either a constant value of 0.5, 1 or 2, or a variable stored in the step register, to be added to an address value. It can also subtract the same values. The twenty-first bit is the nibble part of the address.

Note that an increment of one pixel has a different effect on the address depending on the screen resolution set, and that this is automatically catered for.

All address registers are updated automatically at the end of the appropriate memory cycles.

2.3.3. The Address Output Multiplexer

The address output multiplexer provides the external address to the system memory. It provides three types of address, these are the Source Address, the Destination Address and the Program Address. These are derived directly from the corresponding address registers.

2.3.4. Line Drawing Address

When a line draw is being performed, the address registers are used in a different way from normal. The Destination Address register is used as the line draw address, and the Source Address register and Step register are used as Delta One and Delta Two respectively. During line drawing Delta Two is subtracted from Delta One, and the borrow output produced is used to determine what is added to the Destination Address register. See the section on line-draw below.

2.4. The Sequencer

The sequencer and controller section controls the operation of the other sections of the blitter, and therefore governs the overall operation of the device.

The flow of control is best considered at two main levels. There is an outer loop governing the overall flow of control, and within that three subsections, the inner loop, the parameter read procedure and the command read procedure.

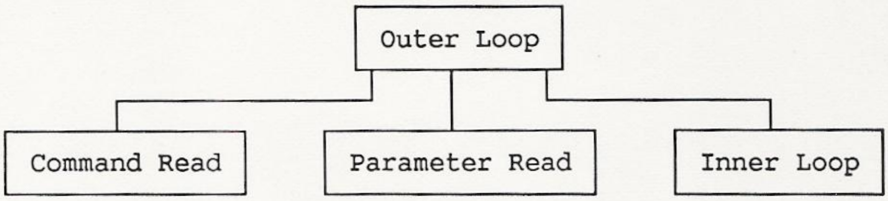
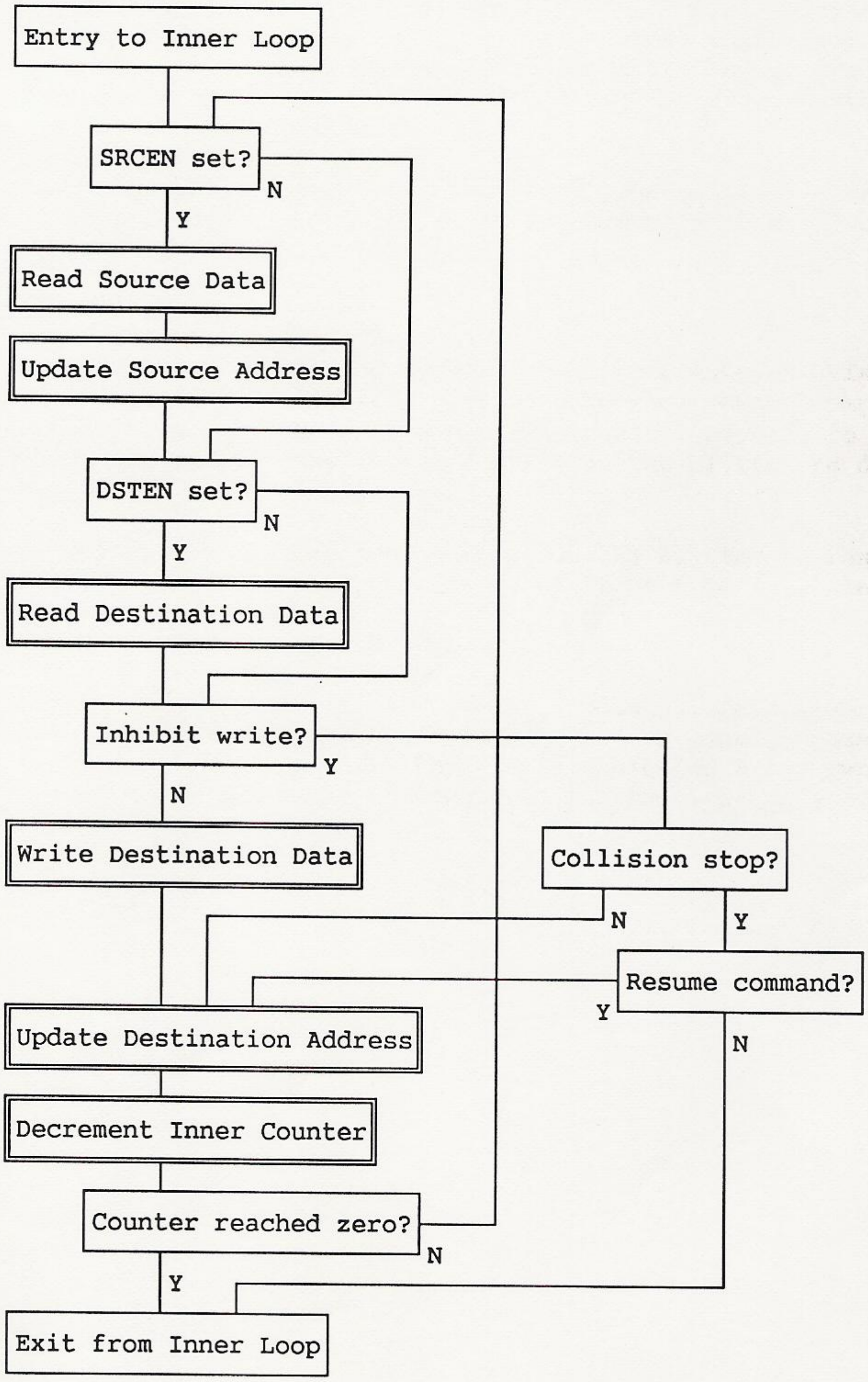


Figure - Control hierarchy

These blocks represent state machines within the blitter, and various control bits within the blitter command affect their operation. To set these control bits correctly requires some understanding of the flow of control within the blitter, and this is discussed below. These control blocks are analogous to a program being run to control blitting.

2.4.1. The Inner Loop

The inner loop performs the actual blitting or line-drawing operation. Its basic form is:



Blitter Inner Loop - Flow Diagram

An Inner Loop cycle can contain up to three memory cycles. These are a read from the source address, a read from the destination address, and a write to the destination address. All three cycles are optional.

If the loop type includes a source read, or a source read and a destination read, then the comparator inhibit mechanism is tested before the destination write occurs. This allows the write cycles to be bypassed when the comparator inhibit conditions are met. Under these circumstances it is possible to have the current operation cease and control returned to the CPU. The program may then examine the address registers to determine where the inhibit has occurred, so that collision detection may be performed; and may determine whether to resume operation or abort it.

The Inner Loop is repeated until the inner counter reaches zero. This is an nine bit counter, and so can be any number of times from 1 to 512.

2.4.1.1. Collision Detection

As can be seen in the diagram above, the blitter makes provision for collision detection by allowing operation to stop when a comparator write inhibit occurs. When this happens, control returns to the CPU, which may then examine the internal state of the blitter to determine what has caused the collision.

At this point, the CPU may choose to allow the blitter to resume the operation it was performing, or may reset it back to its idle state.

2.4.1.2. Inner Loop Execution Times

Normally the inner loop will execute at a rate determined only by the cycle time of the memory being accessed, with no gaps between memory cycles. The only exception to this is an inhibited write cycle, in which case two 12 MHz ticks are required for destination address updating.

2.4.2. The Parameter Read Procedure

The parameter read procedure is a very straightforward sequence that loads up a new set of parameters to the inner loop.

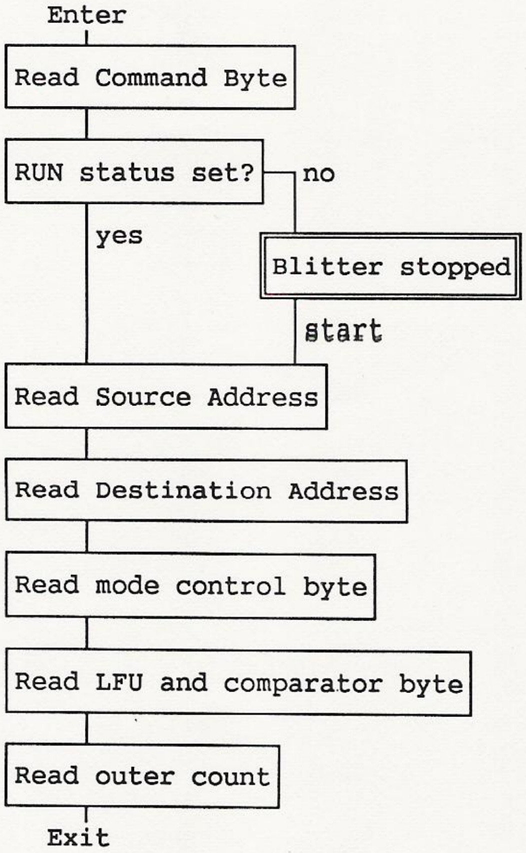
It reads, in sequence, the inner loop counter value, the step register value, and the pattern value - used to preset the data registers.

The inner count is the number of times the Inner Loop is repeated; the step register is used for address incrementing, and the pattern is used for data manipulation.

This procedure is called as part of the command read operation at the start of a blitter operation; and is also called if required during a blitting operation (the PARRD control bit determines this). These extra parameter reads occur between passes through the Inner Loop, to allow various parameters of the loop to be altered. This allows such operations as irregular shape painting and run-length encoded data expansion.

2.4.3. The Command Read Procedure

The command read procedure is used to start a new blitting operation. The inactive, reset state of the blitter of the blitter is part of this procedure. It looks like this:



Blitter Command Read Loop - Flow Diagram

The Stopped state of the Command read loop represents the normal inactive state of the blitter. From this state a Command register write is performed, usually preceded by a write of the program address register, and blitting operation starts.

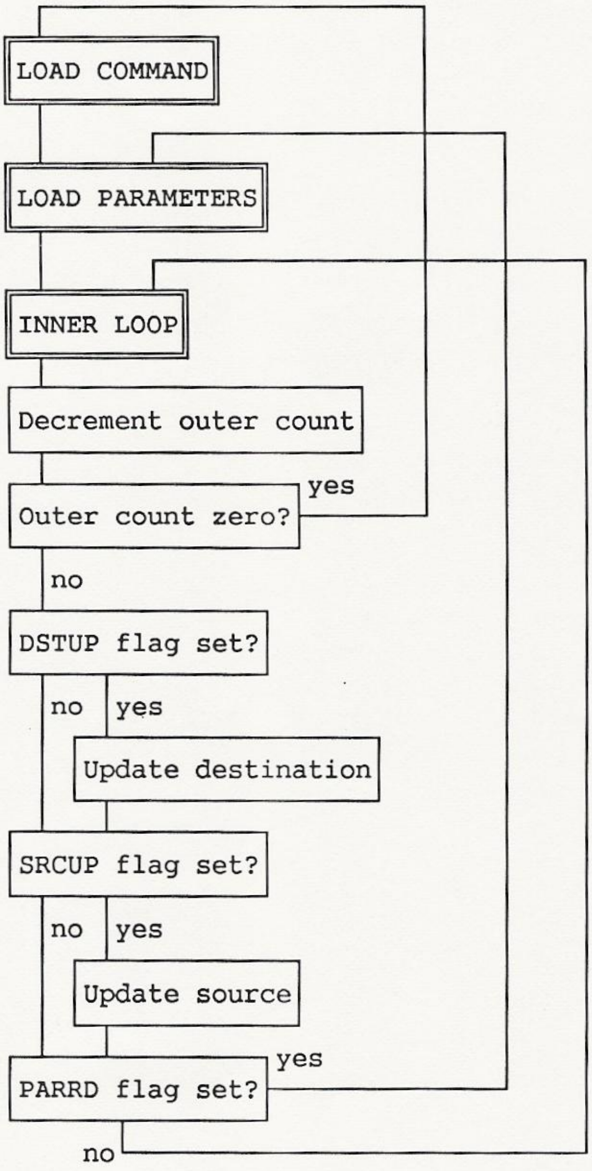
The full set of operational parameters is loaded from the program count address, which is auto-incremented, and control passes out of the command read loop.

When a blitting operation is completed, a new command is read from the program count address, and if this command leaves the blitter in Run mode, then a new set of parameters is loaded and another operation is started. Otherwise the blitter enters its Stopped state and returns the bus to the CPU.

This mechanism allows the blitter to perform arbitrarily long sequences of graphics commands without requiring any processor intervention. This is extremely useful, as processor I/O write cycles are very slow in comparison to blitter memory reads.

2.4.4. The Outer Loop

The Outer Loop controls the overall operation of the blitter, including starting off the three inner loops described above. Its flow of control looks like this:



Outer Loop - Flow Diagram

Normal operation starts on exit from the command read loop. The parameter read loop then entered to read the first set of parameters, and then the inner loop is entered, with the inner counter being loaded ti its initial value before the start of operation.

The Outer counter is then decremented, and if it is zero, then the command read loop is entered; then either or both of the source address and destination address register may be updated with the contents of the step register.

The parameter read loop may then be optionally entered to update various inner loop parameters, before the inner loop is entered again.

The two-level loop nature of the blitter allows it to cope with two-dimensional screen structures, with the source and destination address register updates moving screen address pointers onto the start of the structure on the next line. The parameter read loop adds flexibility by allowing the screen structure parameters to be altered on a line by line basis.

3. MODES OF OPERATION

This section describes some of the modes of operation of the blitter, and how they should be programmed. An understanding of the previous section describing the mechanisms of the blitter is assumed.

3.1. Simple Memory Fill and Copy Operations

The simplest operations are those involving copying one block of memory to another, and filling a block of memory with a pre-defined value. These operations can be performed on linear parts of memory, and on arbitrary screen rectangles.

The destination data register is used as the address of the memory being modified, and the source address register is used as the address of the data being copied, if it is a copy operation.

3.1.1. Handling of Data

The output data being given by the Logic Function Unit will be selected as Pattern data in the case of a memory fill, or as Source data in the case of a memory copy. For simple operations, the comparator control register will be set to zero.

3.1.2. Addressing and Counters

When the operation is to be performed on linear areas of memory, most of the address control bits will be set to zero. The step register is not used, and the only requirement is to determine whether the copy will be made with the address incrementing or decrementing, and setting DSIGN and SSIGN appropriately. Note that the initial value placed in the address register should be the bottom of the area to be operated on if the sign bit is not set, and at the top if it is. In both cases, the first byte read or written will be the first address. The length of the operation will be placed in the inner counter, and the outer counter set to one.

If the block being operated on is very large, then the inner and outer counters may both have to be used, and the number of bytes operated on will be given by the product of the inner and outer counter values.

When either or both of the source and destination data are rectangles rather than linear areas, then the inner counter will contain the rectangle width, and the outer counter the rectangle height. The step register is set to the address increment from the right hand side of the rectangle round to the left hand side on the next line. The SRCUP and DSTUP bits are set according to whether the source or destination are rectangles.

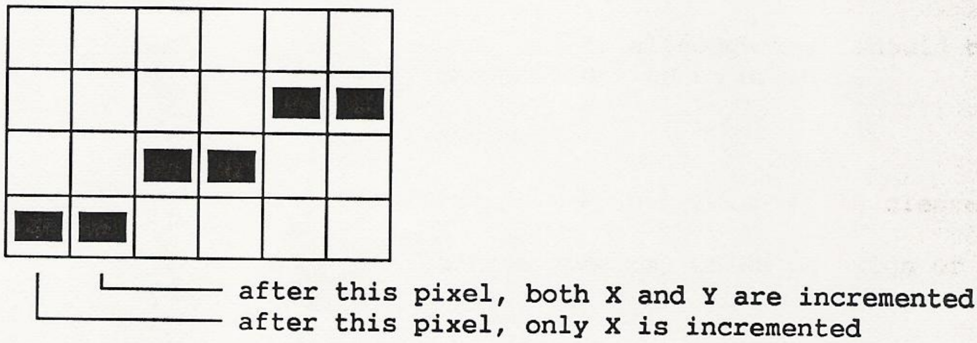
Note that if you are writing a general purpose routine to work with variable sized rectangles, then when the rectangle is 256 bytes wide the inner counter register will be set to zero - this case must be caught and the top bit of the inner counter ILCNT8 should be set (Its a nine-bit counter!).

3.1.3. Inner loop mode control

In eight-bit pixel mode, neither SRCEN nor DSTEN will be used for memory fill, bit SRCEN should be set for memory copy. In four-bit pixel mode DSTEN must always be set as well, so that a destination read is performed to avoid corrupting the other pixel. Note that this will be slower.

3.2. Line Drawing

Line drawing is based on a simple D.D.A. (Digital Differential Analyzer) algorithm. The basis of this is that for a given line one of the X-address or Y-address is always incremented for every pixel drawn, while the other one is incremented if a suitable arithmetic condition is met. This produces a line like this:


after this pixel, both X and Y are incremented
after this pixel, only X is incremented

The algorithm used by the blitter computes the arithmetic condition that causes the conditional increment by repeated subtraction of the smaller of δx or δy[2] from a working value, with the larger being added back when underflow occurs - effectively division to give the gradient.

3.2.1. Computation of line-draw arguments

If a line is being drawn from (x1,y1) to (x2,y2), then:

$$\delta x = |(x1-x2)|$$

$$\delta y = |(y1-y2)|$$

From these $\delta 1$ is given by the larger of δx and δy, $\delta 2$ by the smaller. Then, for each pixel drawn, $\delta 2$ is subtracted from a working value, which is initially set to $\delta 1 \div 2$, and the sign of the result of this subtraction (indicating underflow) is the arithmetic condition for the conditional part of the screen address update. When this underflow occurs, the original value of $\delta 1$ is added back to the working value.

It can be seen that the ratio of δx to δy will give the frequency with which this underflow and adding-back occurs. The ratio between them is, of course, also the gradient of the line (Q.E.D!).

[2] The notation δx refers to the distance along the X-axis that the line corresponds to. This is given by $|(x1 - x2)|$ where x1 and x2 are the X-coordinates of the end points - this notation means the magnitude or modulus of their difference.

3.2.2. Creating a line draw blitter command

These values are used to set the blitter command as follows:

> The start point of the line (x1,y1) is the destination address
>
> δ1 is placed in the middle byte of the source address register - this is the stored original value, and δ1÷2 is placed in the low byte - this is the working value.
>
> δ1 also gives the inner counter value, although δ1+1 should be used if both end points of the line are to be drawn.
>
> δ2 is placed in the step register.
>
> If δx > δy then the YFRAC flag is set, otherwise it is cleared.
>
> SSIGN gives the sign of X-address updates, DSIGN the sign of Y-address updates.

In low-resolution mode δ1 and δ2 are eight-bit values. However, in high-resolution mode it is be necessary to extend them to nine-bit values to get the necessary resolution of line-position. In this case, a further, less-significant, bit is used which would otherwise be set to zero. This is the so-called "minus one" bit.

The inner counter value, given by δ1, is also a nine-bit value, but in this case the **most** significant bit is placed in ILCNT8, and the lower eight bits in the inner counter byte.

The least significant bit of δ2 is placed in STEP-1, the least significant bit of the add-back value is placed in Source Address bit 16, and the least significant bit of the working value is placed in Source Address bit -1.

Note that as δy is never more than 256, δ2 is similarly restricted, so that in high-resolution mode its top bit (of nine) is always clear.

3.2.3. Handling of Data

As all the registers in the address section are occupied in computing the line address, the blitter has no ability to move data from somewhere else when drawing lines. Therefore the data written at the line address has to be given either directly by the pattern data or by a combination of the pattern register and the data already there, according to the L.F.U.

The implication of this is that SRCEN should not be set, for it would produce random data!

3.2.4. Addressing and Counters

The addressing has largely been discussed above. The inner counter is set to the length of the line, and the outer counter will be set to one.

3.2.5. Mode control

In eight-bit pixel mode, DSTEN need not be set, unless used for read-modify-write operation. In four-bit pixel mode, DSTEN must always be set, so that a destination read is performed to avoid corrupting the other pixel.

3.3. Character Painting

The blitter has the ability to paint characters on the screen as a single operation. Character painting as far as the blitter is concerned involves painting a rectangular area up to 8 pixels wide and of arbitrary height. The pixels in this area are either written to or left unchanged according to a bit pattern.

This mode of use is not restricted to character painting, but may be used for expanding any graphics stored as a monochrome bit plane.

3.3.1. Character Font

The source register addresses the bit pattern, normally part of the font, where each byte corresponds to one row of the character. Therefore blitter fonts may be up to 8 pixels wide. Wider fonts may of course be used, but these will require more than one blitter operation to paint a character.

The data is arranged with the bit corresponding to the left-most pixel in the least significant bit, and the top of a character at the lowest address. If the data is less than 8 pixels wide, then the least significant bits of the font data are not used. For example, the font entry for the letter R in a 6x10 font would be something like this:

Byte	Pixels set	Binary	Hex
0	xxxx	01111000	78
1	x x	10001000	88
2	x x	10001000	88
3	x x	01001000	48
4	xxx	00111000	38
5	x x	00101000	28
6	x x	01001000	48
7	x x	10001000	88
8	x x	10001000	88
9		00000000	00

3.3.2. Mode Control and Screen Addressing

The destination address register is used to address the area of the screen to which the character is to be painted. Normally this area will have been cleared to the required background colour by a previous blitter operation. The destination address is initialised to the top left hand corner of the character.

The character to be painted is a rectangle, and therefore the destination addressing is programmed correspondingly. The Inner Counter is set to the width of the character and the Outer Counter to its height. The step register is set to be the screen width less the width of the character. The DSTUP bit is set to allow the destination address to be updated between passes through the inner loop.

Inner loop control bits DSTEN and SRCENF are set, character painting being the reason for the existence of SRCENF. This allows the font byte for each row to be read just once.

The comparator is used to control the painting of pixels, and so the CMPBIT control bit is set, to enable its bit to byte expansion mechanism.

3.3.3. Handling of data in character paint

The colour to be painted is set as the pattern, and this will normally be held in the pattern data register. In four-bit pixel mode, DSTEN will be set, and the destination data register will hold the read value so that the other half of the byte may be written back undisturbed. The source data register holds the font pattern, as mentioned above.

4. BLITTER COMMAND FORMAT

A blitter command is given as a table of data in memory. The blitter loads the contents of this table into its internal registers and performs the specified operation. The blitter will read successive sets of commands until a stop instruction is read into the command register.

The blitter program address must be set up before the command byte is issued. The blitter program address is given by three writable registers, which contain an absolute system address.

A full table of blitter command data starts with a command byte. However the first blitter command in a sequence has its command byte written into the command register by a CPU I/O cycle, and so starts reading the command data from the second byte; and similarly the last blitter command need consist of no more than a command byte with the RUN bit clear.

A blitter command takes the form:

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th></tr>
<tr><td>RUN</td><td>COLST</td><td>PARRD</td><td>SRCUP</td><td>DSTUP</td><td>SRCEN</td><td>DSTEN</td><td>SRCENF</td></tr>
<tr><td colspan="8">---- Source address byte 0 ----</td></tr>
<tr><td colspan="8">---- Source address byte 1 ----</td></tr>
<tr><td colspan="4">Source address top nibble</td><td>SRCCMP</td><td>SWRAP</td><td>SSIGN</td><td>SRCA-1</td></tr>
<tr><td colspan="8">---- Destination address byte 0 ----</td></tr>
<tr><td colspan="8">---- Destination address byte 1 ----</td></tr>
<tr><td colspan="4">Destination address top nibble</td><td>DSTCMP</td><td>DWRAP</td><td>DSIGN</td><td>DSTA-1</td></tr>
<tr><td>STEP-1</td><td>ILCNT8</td><td>CMPBIT</td><td>LINDR</td><td>YFRAC</td><td>RES0</td><td>RES1</td><td>PATSEL</td></tr>
<tr><td>CMPEQ</td><td>CMPNE</td><td>CMPGT</td><td>CMPPLN</td><td>LOG0</td><td>LOG1</td><td>LOG2</td><td>LOG3</td></tr>
<tr><td colspan="8">---- Outer loop counter value ----</td></tr>
<tr><td colspan="8">---- Inner loop count - low byte ----</td></tr>
<tr><td colspan="8">---- Step register value ----</td></tr>
<tr><td colspan="8">---- Pattern byte ----</td></tr>
</table>

Figure - Blitter command summary

4.1. Command Register

0	1	2	3	4	5	6	7
RUN	COLST	PARRD	SRCUP	DSTUP	SRCEN	DSTEN	SRCENF

RUN causes the blitter to start operation. It is used when writing to the command register as an IO port to start the blitter reading a command. If the blitter loads a command with RUN set to zero, as part of a command read, then operation ceases.

COLST causes operation to stop if a collision (write inhibit) occurs. From this point the current operation can be resumed or aborted, and various internal registers may be read.

PARRD requires the blitter to read a new parameter set (that is the step register, the inner count and the pattern register) from the program counter address, every time the inner loop is exited and the outer count has not reached zero.

SRCUP requires the contents of the step register to be added to the source address on exit from the inner loop if the outer count has not reached zero.

DSTUP requires the contents of the step register to be added to the destination address on exit from the inner loop if the outer count has not reached zero.

SRCEN enables a source address read in the inner loop. This will also cause the source address register to be incremented according to the pixel size.

DSTEN enables a destination address read in the inner loop. This does not affect the destination address register which is incremented as part of the destination write cycle.

SRCENF is a special case of SRCEN and causes the source address to be read when the inner loop is first entered, but not subsequently. This is relevant to character paint mode. SRCENF has no effect if SRCEN is set.

4.2. Mode Control and Extension Bits Register

0	1	2	3	4	5	6	7
STEP-1	ILCNT8	CMPBIT	LINDR	YFRAC	RES0	RES1	PATSEL

STEP-1 is the least significant bit of the step. It is used a the nibble bit of the step when in high resolution mode.

ILCNT8 is the most significant bit of the inner loop count. It is normally only used when drawing long lines, or in high-resolution mode. However note that setting the inner count byte to zero will produce 512 operations unless this bit is set.

CMPBIT is used to give a bit to byte expansion scheme. It causes the comparator to generate an inhibit by selecting a bit of the source operand using the inner counter, and generating an inhibit if the bit selected is a zero. The selection is given by 8 in the inner counter selecting bit 0, 7 selecting bit 1, 6 bit 2, and so on.

LINDR sets line-drawing mode. This mode uses both the source and destination address registers to generate the line-draw address, which may be used for both reading and writing.

YFRAC indicates which of the X and Y addresses has the fractional increment in line-drawing mode. It is set if the Y address has the fractional increment.

RES0 and RES1 give the screen mode used by blitter operations. The screen modes are as follows:

RES1	RES0	Word size	Screen width
0	0	4 bits	256 pixels
0	1	8 bits	256 pixels
1	0	4 bits	512 pixels
1	1	16 bits	n/a

Note that the screen width given is only relevant to line-drawing mode, and that 16 bit mode is not used in line drawing mode.

PATSEL is used to select the pattern data register to replace the source data register as the source input to the LFU. This is relevant to character painting, where the source data register will contain the font data, and the pattern data register will contain the ink colour.

4.3. Step Register

0	1	2	3	4	5	6	7
STEP0	STEP1	STEP2	STEP3	STEP4	STEP5	STEP6	STEP7

STEP 0-7 are the eight most significant bits of the step. The step is used to increment an address by a number other then 1, so that non-contiguous screen data may be handled in single operations. The step register is also used in line drawing mode to hold δ2.

4.4. Comparator and Logic Function Unit Control Register

0	1	2	3	4	5	6	7
CMPEQ	CMPNE	CMPGT	CMPPLN	LOG0	LOG1	LOG2	LOG3

CMPPLN enables plane mode where the three comparator functions operate on the plane number bits as opposed to the entire pixel.

CMPEQ causes the comparator to inhibit an inner loop write, if in plane mode the priority of the destination pixel is equal to the plane priority of the source pixel, or if the entire pixel is the same if not in plane mode.

CMPNE causes the comparator to inhibit an inner loop write, if in plane mode the priority of the destination pixel is not equal to the plane priority of the source pixel, or if the entire pixel is not the same if not in plane mode.

CMPGT only operates in plane mode, and causes the comparator to inhibit the write if the plane priority of the destination pixel is greater than the plane priority of the source pixel.

LOG 0-3 The logic function unit (L.F.U.) controls the data that is written in a destination write cycle. The L.F.U. allows any logical combination of the source and destination data. This is achieved by each of the L.F.U. bits selecting one of the Minterms, with the output being given by the OR of the selected terms.

The correspondence between bits and Minterms is

LOG0	NOT source AND NOT destination
LOG1	NOT source AND destination
LOG2	source AND NOT destination
LOG3	source AND destination

There are therefore sixteen possibilities.

4.5. Source Address Register

Source Address Low Byte

0	1	2	3	4	5	6	7
SRCA0	SRCA1	SRCA2	SRCA3	SRCA4	SRCA5	SRCA6	SRCA7

Source Address Middle Byte

0	1	2	3	4	5	6	7
SRCA8	SRCA9	SRCA10	SRCA11	SRCA12	SRCA13	SRCA14	SRCA15

Source Address High Byte

0	1	2	3	4	5	6	7
SRCA16	SRCA17	SRCA18	SRCA19	SRCCMP	SWRAP	SSIGN	SRCA-1

The Source Address Register is used as a pointer to a source of data for blitting operations.

Bits SRCA0 to SRCA19 correspond directly to address lines 0 to 19. The SRCA-1 bit is used as a nibble address in high resolution mode.

The source address register is also used as δ1 in line-drawing mode.

SWRAP causes source address updates to wrap on 64K boundaries, as opposed to running linearly through memory.

SSIGN is the sign used when updating the source address. Setting it causes the source address to be decremented rather then rather then incremented. It makes X negative in line-drawing.

SRCCMP selects the source data register as the source input to the comparator. If it is clear, the pattern data register is used.

4.6. Destination Address Register

Destination Address Low Byte

0	1	2	3	4	5	6	7
DSTA0	DSTA1	DSTA2	DSTA3	DSTA4	DSTA5	DSTA6	DSTA7

Destination Address Middle Byte

0	1	2	3	4	5	6	7
DSTA8	DSTA9	DSTA10	DSTA11	DSTA12	DSTA13	DSTA14	DSTA15

Destination Address High Byte

0	1	2	3	4	5	6	7
DSTA16	DSTA17	DSTA18	DSTA19	DSTCMP	DWRAP	DSIGN	DSTA-1

The Destination Address Register is used as a pointer to the destination of the data in blitting operations.

Bits DSTA0 to DSTA19 correspond directly to address lines 0 to 20. The DSTA-1 bit is used as a nibble address in high resolution mode.

DWRAP causes destination address updates to wrap on 64K boundaries, as opposed to running linearly through memory.

DSIGN is the sign when updating the destination address. Setting it causes the destination address to be decremented rather then incremented. It makes Y negative in line-drawing.

DSTCMP selects the destination data register as the source of destination data to the comparator. If it is clear then the pattern data register is used.

4.7. Program Address Register

Program Address Low Byte

0	1	2	3	4	5	6	7
PRCA0	PRCA1	PRCA2	PRCA3	PRCA4	PRCA5	PRCA6	PRCA7

Program Address Middle Byte

0	1	2	3	4	5	6	7
PRCA8	PRCA9	PRCA10	PRCA11	PRCA12	PRCA13	PRCA14	PRCA15

Program Address High Byte

0	1	2	3	4	5	6	7
PRCA16	PRCA17	PRCA18	PRCA19				

The Program Address points to the source of blitting operation commands. Data is read from it sequentially upwards through memory.

Note that these bytes must always be written in the order low byte, middle byte, then high byte.

4.8. Blitter IO Registers

The blitter allows some of the above registers to be visible in the CPU IO space. In addition to these, a blitter status byte and control register are accessible to the CPU.

4.8.1. Blitter Status Byte

0	1	2	3	4	5	6	7
CSTOP	ISTOP						

CSTOP indicates that the blitter has stopped because a write inhibit has occurred and the COLST command bit is set.

ISTOP indicates that the blitter has stopped because the CPU interrupt line is active.

If neither of the above bits is set then the blitter is idle.

4.8.2. Blitter Command Register

0	1	2	3	4	5	6	7
IMASK	RESUME	RESET					

IMASK prevents interrupts from stopping blitter operation.

RESUME requests blitter operation to continue after a collision stop has occurred.

RESET sets the blitter state back to idle after a collision stop has occurred.

The Floppy Disc Controller

The FDC controls the disc drive during read operations. It receives instructions from the 8086 of the form STEP FORWARD (or BACK) ** STEPS. READ INTO ADDRESS XXXX.
It transfers complete tracks of data as one block of 4KBytes (6KBytes may be possible) into system memory without further intervention.
It decodes the MFM code and checks the CRC code.
This enables 4K (6K?) block of program or data to be paged into memory with very little CPU intervention. Provided allowance is made for the delay involved in reading the data (seek time 6ms per track, latency time 100ms average 200ms MAX., read time 200ms) the transfer can be transparent to the program execution. This gives a virtual memory size of 720KBytes (960K if possible) with a double sided disc or 360K (480K?) single sided.

Disc writing is done by the DSP in order to reduce the complexity of the FDC. The only intended use for data written to the disc is for saving high scores and game position data. To reduce the possibility of software piracy the drive hardware can only write on track zero. This limits the amount of data which can be written to 8K (12K?) double sided or 4K (6K?) single sided.
The fact that the DSP is not available for programme execution during disc writes is not a disadvantage since this will usually be suspended during save game or high score sessions.
INSTRUCTION CODES AND REGISTER SET - T.B.A.

A. The DSP Arithmetic Logic Unit

The DSP contains an arithmetic logic unit compatible with the Texas Intruments' 74181 device, and all its functionality is available by setting bits in the ALU mode register and using the GAI instruction. For those of you who don't have access to a TI data book here is a summary of the functions. Note that carry in to the ALU is given by the carry flag, and that carry sometimes means borrow.

S	M=1	M=0	
3210		Cin=0	Cin=1
0000	/A	A	A plus 1
0001	/(A+B)	A+B	(A+B) plus 1
0010	/A.B	A+/B	(A+/B) plus 1
0011	0	minus 1	0
0100	/(A.B)	A plus (A./B)	A plus (A./B) plus 1
0101	/B	(A+B) plus (A./B)	(A+B) plus (A./B) plus 1
0110	A#B	A minus B minus 1	A minus B
0111	A./B	(A./B) minus 1	A./B
1000	/A+B	A plus (A.B)	A plus (A.B) plus 1
1001	/(A#B)	A plus B	A plus B plus 1
1010	B	(A+/B) plus (A.B)	(A+/B) plus (A.B) plus 1
1011	A.B	(A.B) minus 1	A.B
1100	1	A plus A	A plus A plus 1
1101	A+/B	(A+B) plus A	(A+B) plus A plus 1
1110	A+B	(A+/B) plus A	(A+/B) plus A plus 1
1111	B	A minus 1	A

Notation

/	Boolean NOT (highest precedence)
+	Boolean OR
.	Boolean AND
#	Boolean exclusive OR

B. The DSP Assembler Manual

The DSP assembler is a program which runs on an IBM PC or compatible under MSDOS. It is a two-pass assembler which produces an assembler listing file and a hex output file in a form suitable for uploading into the Slipstream system, and from there into the DSP.

It allows most of the normal assembler functions (except linking), and will allow the use of include files and macros.

The source file takes the default extension .S if none is supplied, and the listing and output files will take the extensions .LST and .BIN respectively, using the source file name as a root.

1. How to invoke the assembler

```
DASM <filename>
```

2. Source File format

With the exception of strings, case is ignored.
Square brackets [] denote optional items.

String — a sequence of characters enclosed in single quotes.

Identifier — a sequence of the characters A-Z 0-9 $. _ the first of which must be A-Z.

Reserved Word — An identifier which has meaning to the assembler. They are listed below and should not be used by the user.

Label — Identifier followed by a colon ':'

Decimal Number — a sequence of digits 0-9

Hex Number — '$' followed immediately by a sequence of hex digits 0-9,A-F

Bin Number — '%' followed immediately by a sequence of bin digits 0-1

Expr — a sequence of numbers,identifiers,and operators

Addr_Expr — an arithmetic expression which must yield a value in the range 0 to 511 (this includes some pre-defined constants, see below).

Word_Expr — an arithmetic expression which must yield a value in the range -32768 to 32767. Note that this facility is not very efficient in its use of data space - a new location is used for each expression, and no attempt is made to spot identical expressions or use ROM constants.

Register — one of the reserved words MZ0, MZ1, DMA0, DMA1, DMD, PC, X, AZ, DAC1, DAC2, DAC12, IX, MODE

(a) Expression Operators

Listed in decreasing precedence

unary	- , +	
binary	* , /	
binary	+ , -	
binary	< , >	shift left/right
binary	&	bitwise AND
binary	|	bitwise OR
()		

The assembler expects to find each source statement contained on one line. Within the line there are no constraints on the layout of the statement.

```
label:    mnemonic  op1,op2   ; comment
```

(b) Reserved Words

The following identifiers are reserved to the assembler.

CODE_SEG

CODE_SEG
Informs the assembler that further output is to generated in the code segment address space. This directive MUST be used.

CODE_START

CODE_START addr_expr
Tells the assembler at what address the code segment starts at. If it is not used code generation will default to 0. The addr_expr must be satisfied at the end of pass 1.

COUNT_RESET

COUNT_RESET
Resets the internal T-state counter to zero. This may be used as a quick way of determining the length of linear pieces of code.

DATA_SEG

DATA_SEG
Informs the assembler that further output is to generated in the data segment address space. This directive MUST be used.

DATA_START

DATA_START
Tells the assembler at what address the data segment starts at. If it is not used data generation will default to 180h. The addr_expr must be satisfied at the end of pass 1.

END

END [addr_expr]
Informs the assembler that it has reached the end of the source text. An optional parameter defines the address at which the code should be entered. If this is not specified execution will start at the base of the CODE_SEG. The addr_expr must be satisfied at the end of pass 1.

EQU

identifier EQU word_expr
Assign a value to an identifier that may not be changed during this invocation of the assembler.

INCLUDE

INCLUDE string
Includes source text from a specified file at the point marked by the directive.

LIST

LIST
If the listing file is open this directive results in output being sent to the list file. This directive is not itself listed. The default is for listings to be produced.

NOLIST

NOLIST
Turns off listing output. Any source lines following this directive are not listed. Neither is this directive.

PAGE

PAGE
The assembler continues listing output on a new page.

SET

identifier SET word_expr
Assign a value to an identifier that may be changed during this invocation of the assembler.

STATE_COUNT

STATE_COUNT
The assembler displays the current value of the T-state counter in the listing file.

TITLE

TITLE string
Defines the title to be printed at the top of each page of the listing file. If it is not used the title will be blank.

WORD

[identifier] WORD word_expr
Allows the user to define data areas and initialise data in the data segment. This directive should not be used until after a DATA_SEG directive has been issued.

(c) Processor Instruction Mnemonics

```
MOV       addr_expr,Register
MOV       Register,addr_expr
MOV       Register,#word_expr
NOP
MULT      addr_expr / #word_expr
MAC       addr_expr / #word_expr
ADD       addr_expr / #word_expr
ADC       addr_expr / #word_expr
AND       addr_expr / #word_expr
OR        addr_expr / #word_expr
SUB       addr_expr / #word_expr
SBC       addr_expr / #word_expr
GAI       addr_expr / #word_expr
CCF
INTRUDE
```

(d) Macro Facilities

The macro facilities available are limited to parameter substitution and inline expansion.

A macro is introduced by the word MACRO followed by the name of the macro. The body of the macro then follows and finally to mark the end of the macro an ENDM statement is required. To get at the parameters passed to the macro use @n where n is a number or symbol defining the parameter number. The first parameter is @1. If a reference is made to an unitialised parameter a null string will be substituted in its place. This can be of some value.

NOTE: If a macro name clashes with a reserved word then the reserved will no longer be available to the user.

```
MACRO     ADD2      ; add two numbers
MOV       AX,@1     ; get parameter 1
ADD       @2        ; get parameter 2
ENDM
```

To invoke this macro use its name and the parameters seperated by commas.

```
ADD2      symbol1,symbol2
```

(e) Constants

The ROM constants and data-space registers in the machine have a set of pre-defined names built in to the assembler, to save the user having to type them in every time. These are defined as if the SET directive had been used, allowing them to be re-defined if desired. They are:

Address	Data	Name
000		SINE
100	0000	H0000 or ZERO
101	0001	H0001 or ONE
102	0002	H0002 or TWO
103	0004	H0004 or FOUR
104	0008	H0008 or EIGHT
105	0010	H0010 or SIXTEEN
106	0020	H0020
107	0040	H0040
108	0080	H0080
109	FFFF	HFFFF or MINUS1
10A	FFFE	HFFFE or MINUS2
10B	FFFC	HFFFC or MINUS4
10C	8000	H8000
140		R_INTRA
141		R_INDEX
142		R_DMA0
143		R_DMA1
144		R_DMD
145		R_MZ0
146		R_MZ1
147		R_Z2

148		R_DAC1
149		R_DAC2
14A		R_PC
14B		R_MODE
14C		R_X
14D		R_AZ
14E		R_INTRD
14F		R_DSPIO

3. Error Reporting

Error messages are printed during pass 2. If a listing is being produced the messages will be written into the file as well as displayed on the screen. Only one error message per line is written to the file but a number of error messages per line may be displayed on the display. The messages shown on the display will provide more detailed information about the error.

Messages have a standard format.

```
(1)  (2)    (3)           (4)        (5)        (6)
***  nn; SOURCE.S; macro ADD2; line nn; Syntax error
***  nn; SOURCE.S; Syntax error
```

(1) The error message tag.
(2) The line number of the source file that the error occurred on.
(3) The name of the source file currently being worked on, this will be the name of an include file if appropriate.
(4) If the source line has invoked a macro the name is given.
(5) Ditto, the line number in the macro that contains the error.
(6) The error description.

The messages are presented in this way so that an error may be quickly located. Even if macros invoke macros the name of the macro that contains the error is reported. This is especially useful when the bodies of the macros are not listed through the use of listing control directives.