FLIGHT
8086
MICROPROCESSOR
TRAINER

TECHNICAL REFERENCE MANUAL

Published by:

Flite Electronics International Limited
Church House Farm
Clewers Hill
Waltham Chase
Hampshire
SO32 2LN

Tel :+44 (1489) 892422
Fax :+44 (1489) 897929
E-mail : Sales@flite.co.uk

Website : www.Flite.co.uk

First published June 1994.
Third Edition published December 1997
Fourth Edition published January 2007

Printed in the United Kingdom

COPYRIGHT

DISCLAIMER

Flite Electronics International Limited, makes no representations or warranties, either express or implied, with respect to the contents hereof and specifically disclaims any warranties or merchantability or fitness for any particular purpose. **Flite Electronics International Limited** software described in this manual is sold or licensed "as is". Should the programs prove defective following their purchase, the buyer (and not **Flite Electronics International Limited**, its distributor, or its dealer) assumes the entire cost of all necessary servicing, repair, and any incidental or consequential damages resulting from any defect in the software. Further, **Flite Electronics International Limited** reserve the right to revise this publication and to make changes from time to time in the content hereof without obligation of **Flite Electronics International Limited** to notify any person of such revision or changes.

FLITE ELECTRONICS INTERNATIONAL LTD
-DECLARATION-

CE

APPROVED

1994 No. 3080
ELECTROMAGNETIC COMPATABILITY

The Electromagnetic Compatibility (Amendment) Regulations 1994

EDUCATION, TRAINING AND EVALUATION EQUIPMENT

Section 8-(1)

This regulation applies only to education and training equipment which would not, except for the provisions of this regulation, conform with the protection requirements under normal conditions of use in its usual electromagnetic environment.

! WARNING !

The use of this apparatus outside the classroom, laboratory, or similar area invalidates conformity with the protection requirements of the Electromagnetic Compatibility Directive (89/336/EEC) and could lead to prosecution.

Electromagnetic Compatibility – Further Notes

From 1st January 1995 it is a legal requirement that we comply with the EEC directive (89/336/EEC). As members of BESA (British Education Suppliers Association) and ETEMA (Engineering Teaching Equipment Manufactures Association) EMC 1944 No 3080 (as amended) Section 8.1 is our lobbied contribution to this statutory instrument enabling us to legally comply.

By using equipment within the confines of an electromagnetically safe environment i.e. a Faraday Cage, or using the PCB in a completely sealed, earthed metal box, with correctly de-coupled and screened leads, EMC radiation will be negligible and well within the wider remit of this directive. However, this makes education training and evaluation equipment either impractical or impossible to use. Therefore Section 8.1 is one of the necessary recent amendments.

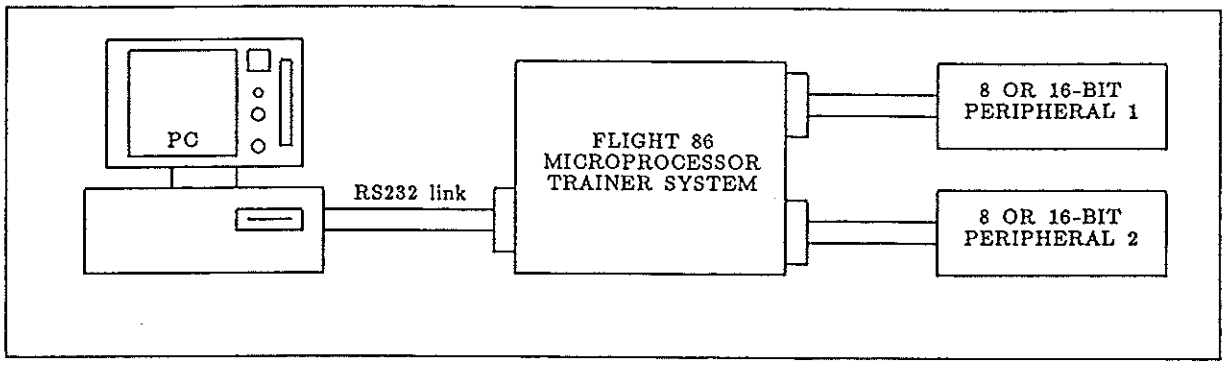
In short, use this product sensibly and not in an electromagnetic sensitive area and only within the confines of a classroom, laboratory, study area or similar place. If in doubt, consult a professional electronics engineer to review the electro-magnetic environment.

Flite Electronics International Limited and its staff are not liable for misuse o fits equipment or breaches of this EEC Directive.

CONTENTS

INTRODUCTION

The FLIGHT 86 Trainer System is designed to simplify the teaching of the 8086 CPU and some of its commonly used peripherals. It can be linked to most PCs with a simple serial line, so that code may be assembled and debugged in a supportive software environment before being downloaded into the RAM on the board. The board itself may then be linked to other peripheral devices. A block diagram of this mode of operation is shown below:



Once downloaded, the code may be executed and examined in a system which is accessible to the user. Data may be manipulated on the board and the effects viewed on the PC. The software which handles this two-way transfer is supplied with the board, in the form of a monitor program resident on the board in EPROM, and a disk containing the "host" software for the PC.

Apart from its use linked to a PC, the board may also be used independently, under the control of a user, either for fault-finding on 8086 systems, or for control projects. For fault-finding exercises, a number of test routines are supplied with the monitor EPROM which enables many faults to be investigated using simple 'scope techniques. The accessibility of components on the board means that the faults may be easily applied.

In control applications, the board is ideal for projects from the simple 'flashing LED' variety, to sophisticated, real time systems such as floppy disc controllers. The control program can be blown into the board's EPROMs either in place of, or in addition to, the monitor program already present. The board then becomes a powerful 'stand-alone' control system. The development and testing of the software is helped enormously by using the system linked to a PC initially, and downloading development code into RAM.

This manual is organised to ensure all the information required for hardware or software interfacing is available in a clear, concise form.

The hardware circuit diagram is split into easily managed sections, each on a fold-out sheet. Each section of the circuit has a description attached. The circuit diagrams are folded to allow two sections to be on view at a time if required. Chip information has been minimised so that only information essential to usage and programming is presented. The manufacturer's data sheets should be obtained if a greater depth of knowledge is required.

The software is covered by a pseudo-code listing and the full monitor source code, liberally commented, to give the user a flavour for 8086 code. The 8086 instruction set is presented in two forms to cover differing requirements. It is expected that most machine code will be generated using an assembler and downloaded to the controller board using the software options supplied.

SPECIFICATION

Microprocessor

CPU	8086	Operating in Minimum Mode
CGD	8284A	Clock Generator Driver. Oscillator source is a 14.7456 MHz quartz crystal. CPU clock, CLK, is 4.9152 MHz. Peripheral clock, PCLK, is 2.4576 MHz (ideal for baud rate generator).

Memory

EPROM	2764 x 2	16k byte (expandable on board to 64k byte).
RAM	6264 x 2	16k byte (expandable on board to 64k byte).

On-board Peripherals

PPI	8255A x 2	Programmable Peripheral Interface providing: 4 off 8-bit parallel ports with handshake lines.
PIT	8253	Programmable Interval Timer providing: 3 off 16-bit counter/timer channels.
USART	8251A	Universal Synchronous/Asynchronous Receiver/Transmitter or Programmable Communication Interface providing: Full duplex asynchronous RS232 link, using standard MC145407 device for level shifting. Baud rate software programmable using the 8253.
PIC	8259A	Programmable Interrupt Controller providing: 8 levels of priority for above devices.

Power Supply Requirements

+ 9V	@ 1 A	A mains adaptor having the required O/P is supplied as part of the FLIGHT 86 Training System UK 240V 50Hz version Part No. 523-027 European 110V 60Hz version Part No. 103-028 The board has a fused crowbar fitted for protection.

Physical and Controls

Dimensions 221mm x 156mm with fixing holes.
Non-maskable interrupt button
Reset button.
Fuse, 20mm, 1 A fast blow.

Interconnections

50 pin IDC male Bus Expansion connector.
2 off 40 pin IDC male Parallel Interface connectors.
1 off 9 way male 'D' type Serial Interface connector.

OVERVIEW

Memory Address Decoding

The on-board memory devices are fully address decoded, making off-board memory expansion a realistic option.

The size of ROM and RAM fitted on board is link selectable making for flexible device usage. They are mapped to make full use of the 16-bit bus. See the Memory Map on page 2 for full information.

I/O Address Decoding

The peripheral chips on the board are mapped in the I/O (Input/Output) area, so are accessed using the IN and OUT instructions. See page 4 for more detail.

Four spare isolated I/O decoded signals are available on the expansion bus ensuring peripheral device expansion is as simple as possible.

Parallel Input/Output

Each of the two 8255A PPI (Programmable Peripheral Interface) chips is connected to a 40 way IDC header connector. See page 5 for full wiring details.

Each connector has two 8-bit ports with associated handshake control lines. These handshake lines may be used to provide interrupt-driven I/O.

The PPIs are mapped to odd and even addresses giving full 16-bit I/O capability using one instruction, ie OUT 00, AX will write the contents of the high byte of AX to Port A of one PPI, and the low byte of AX to Port A of the other.

Serial Input/Output

Asynchronous serial I/O is provided by the 8251A USART (Universal Synchronous/Asynchronous Receiver/Transmitter), with a MC145407 level shifter providing full ± 12V RS232 signalling, from the TTL levels, at the 9 way male 'D' type connector. The MC145407 also reverses the process by shifting the ± 12V levels back to TTL for the 8251A.

Hardware handshaking protocols may be implemented using RTS, CTS, DTR, and DSR lines. The 8251A Rx and Tx READY lines are connected to the 8259A PIC to give interrupt capability. The baud rate is software programmable using the 8253 PIT. The Host to Controller software normally uses 9600 baud without handshaking, although this may be changed from the Host if desired. See page 7 for wiring details.

Timer

The 8253 PIT (Programmable Interval Timer) chip provides three 16-bit counter/timer channels. The PIT input clock is PCLK, derived from the 8284A clock generator driver.

A link allows the clock for channel 1 to be selected between PCLK and the output of channel 0. Thus time delays in excess of 25 minutes may be achieved.

The output of one channel is available on the Parallel I/O connectors. Channels 0 and 1 are available to the 8259A PIC chip to give interrupt capability.

Clock output 2 is used by the controller software as a baud rate generator for the 8251A serial communication device.

Interrupts

The 8259A PIC (Programmable Interrupt Controller) may be used to accept prioritised interrupts from the on-board peripheral chips. They are hard-wired to give a pre-determined priority. See the interrupt table on page 4 for more detail.

A non-maskable interrupt, NMI, is provided by an on-board push button switch or via P4, the bus expansion connector.

The monitor software has two NMI routines that are used by our Fault Finding books. These routines are inhibited once serial communication with the host is established.

Host Software

The host computer must have a RS232 port fitted. The 8086 code may be assembled on a standard MSDOS PC and the .OBJ files LINKED to extended Intel Hex format for downloading to the controller board. See page 26 for details of Intel Hex format.

A CP/M PC may be used with a suitable 8086 cross assembler. (A special version of the host software will be required depending on the serial interface devices used.)

The host software communicates with the controller software to give many of the facilities available to a development system, allowing code development using the controller board. See the command description on page 28.

Controller Board Software

A simple Monitor program resides in the system ROM primarily to allow code to be downloaded from an Intel Hex file into the controller RAM. The monitor code also enables development activity such as register and memory management and program execution to take place, The monitor program may be enhanced to suit particular needs. A full source listing is provided in this manual, starting at page 45.

The monitor software may be integrated into user code to allow fast, flexible code development and an introduction into the techniques of interfacing software.

The monitor ROM will always reside between F000:E800 and FFFF, with a user boot address allowed for at E800. See page 39 for details of user access to the monitor.

Experiments

I/O devices, driven from the 8255A PPIs, may be connected to the parallel ports. Your existing units may well fit, with a suitable lead. See page 5 for wiring details. Alternatively, Flight Electronics product 611-001 will enable connection via a 4mm plug and socket.

The expansion bus allows more sophisticated peripherals to be added, and your existing 8085 devices may even work as the multiplexed bus of the 8086 is very similar to the 8085.

An experiment book is available. This is supplied with a modified monitor ROM that contains the experiment model answers. These may be executed for demonstration purposes if required.

HARDWARE

The basic components are identified on page (vii), the SPECIFICATIONS, and the BOARD COMPONENT LAYOUT is shown on page 8.

The following notes and tables are intended to give you detailed hardware information about the board. This will be of particular interest to those wishing to expand the existing facilities available on the board.

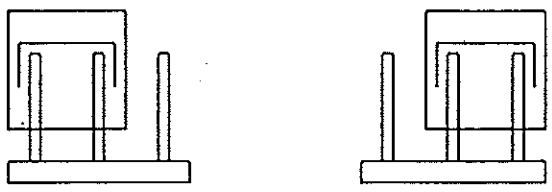
The circuit diagrams are on fold-out sheets to enable detailed work to be carried out with test equipment and allow two different sections of the manual to be available at any time. Each circuit diagram has a brief description attached.

Board Links

The board has 9 links fitted, labelled L1 to L9, clustered around the 8086 CPU chip. These links have the functions described in the table below:

Link	Function
L1 L2 L3	Select RAM size (see page 3)
L4 L5	Select EPROM size (see page 3)
L6	PIT channel 1 clock source A PCLK B PIT channel 0 output
L7	HOLD source remove for external control (requires pull-up resistor)
L8	Serial I/O RTS source remove for external control
L9	8086 TEST Mode remove for test mode only

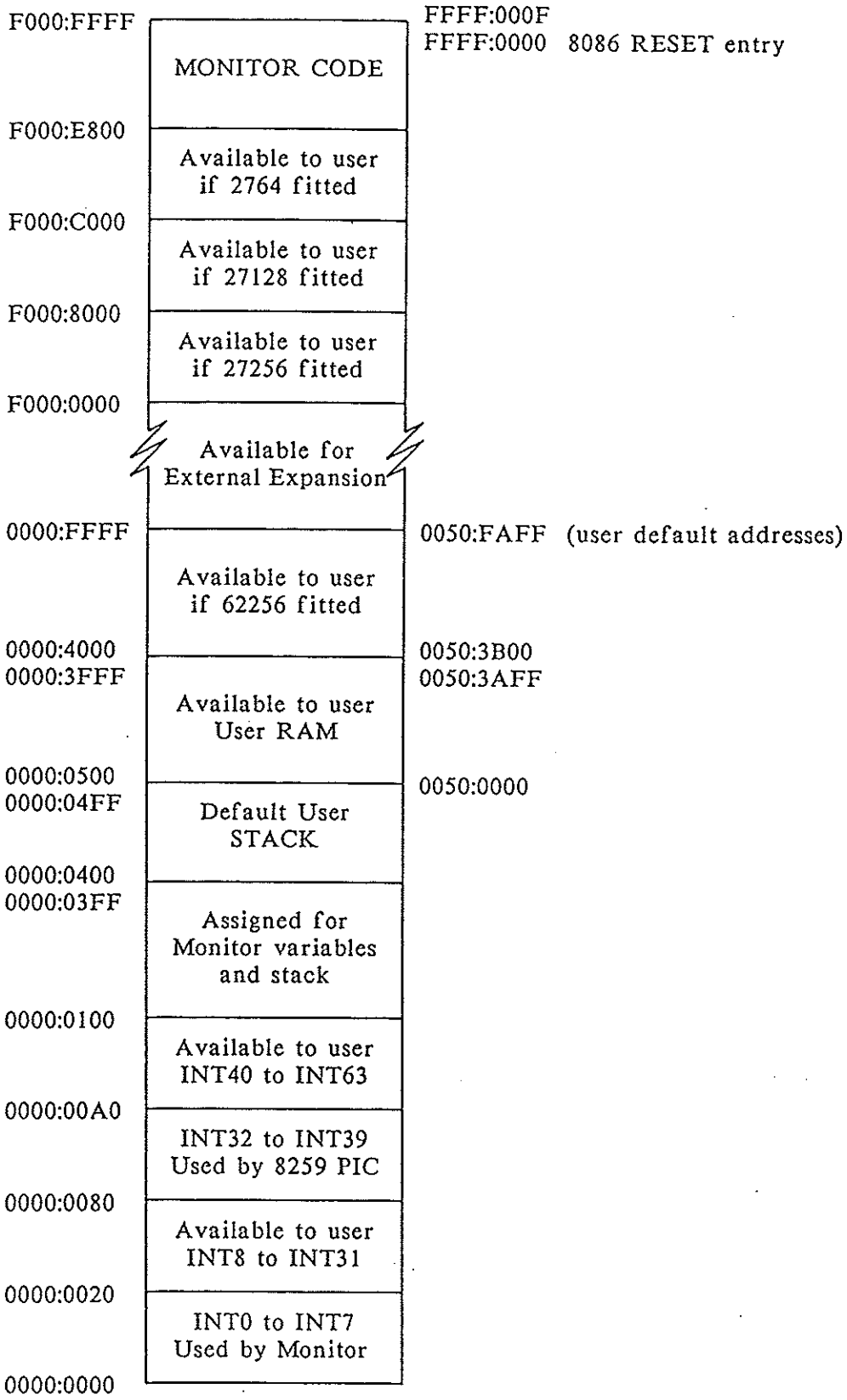
The links are a shorting bar that slides on and off the protruding set of pins. The links L1 to L6 go on two ways. The diagram below shows the two positions in section:



A B

The board is supplied with links L1 to L6 fitted to position A, and links L7, L8 and L9 are fitted. Unless you intend to make changes to the system this will be the default condition.

Full Memory Map



On-board Memory

The memory for this board is fully decoded, to allow for user expansion via the 40-way IDC expansion bus connector, P4.

The memory is split into ODD and EVEN banks of 8-bit memory devices.

On-board RAM expansion

The board is supplied with 2 x 6264 8k RAM chips giving 16k byte of RAM. This may be expanded, by plugging in 2 x 62256 chips and changing the board links as shown below:

Link Position L1	L2	L3	RAM Device	Total Storage	Address Range
A(2-3)	A(2-3)	A(2-3)	6264	16k byte	00000 to 03FFF
B(2-1)	B(2-1)	B(2-1)	62256	64k byte	00000 to 0FFFF

On-board EPROM expansion

The board is supplied with 2 x 2764 8k EPROM chips giving 16k byte of ROM. This may be expanded, by plugging in larger chips and changing the board links as shown below:

Link Position L4	L5	EPROM Device	Total Storage	Address Range
A(2-3)	A(2-3)	2764	16k byte	FC000 to FFFFF
A(2-3)	B(2-1)	27128	32k byte	F8000 to FFFFF
B(2-1)	B(2-1)	27256	64k byte	F0000 to FFFFF

Link pins are annotated 1, 2, and 3 on the board, and identify as shown below:

Link Position		Link Pins
1	A	2-3
1	B	2-1
2	A	2-3
2	B	2-1
3	A	2-1
3	B	2-3
4	A	2-3
4	B	2-1
5	A	2-3
5	B	2-1
6	A	2-1
6	B	2-3

I/O Map

All the on board peripheral chips are 8-bit devices, with their I/O addresses mapped EVEN. The exceptions are the two 8255A's; one is mapped EVEN and the other ODD. So true 16-bit I/O may be achieved.

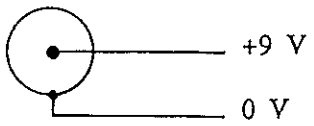
Device	Register	Decoder Output	Address
8255A U10 PPI	Port A	$\overline{Y0}$	00000000 (00)
	Port B		00000010 (02)
	Port C		00000100 (04)
	Control		00000110 (06)
U9 PPI	Port A	$\overline{Y0}$	00000001 (01)
	Port B		00000011 (03)
	Port C		00000101 (05)
	Control		00000111 (07)
8253 U8 PIT	Count 0	$\overline{Y1}$	00001000 (08)
	Count 1		00001010 (0A)
	Count 2		00001100 (0C)
	Mode Word		00001110 (0E)
8259A U11 PIC	ICW1, OCW2-3	$\overline{Y2}$	00010000 (10)
	ICW2-4, OCW1		00010010 (12)
8251A U7 USART	Data	$\overline{Y3}$	00011000 (18)
	Status/Control		00011010 (1A)
Available on P4 (Pin 38)		$\overline{Y4}$	00100xxx
Available on P4 (Pin 37)		$\overline{Y5}$	00101xxx
Available on P4 (Pin 36)		$\overline{Y6}$	00110xxx
Available on P4 (Pin 35)		$\overline{Y7}$	00111xxx

Maskable Interrupt Wiring Table

The following on board peripheral device signals are hard wired to the 8259A PIC with priority shown.

Device	Signal	Priority
8251A	U7 RxRdy	0
	U7 TxRdy	1
8255A	U9 PC3	2
	U9 PC0	3
8255A	U10 PC3	4
	U10 PC0	5
8253	U8 Clock 0	6
	U8 Clock 1	7

Power Supply Connector P6



Power Supply Requirements

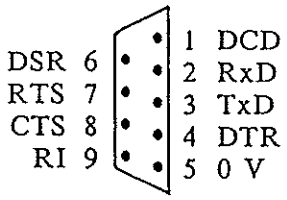
+9 V @ 1 A The board has a fused crowbar fitted for protection.

Parallel Port socket, P1 and P2, connections

Connection to the 8255A PPI device is via 40 way IDC header connectors. The wiring of P1 and P2 is identical, except P1 connects to U10 and P2 connects to U9.

	40	20		
	39	19		
PIT-INT	38	18		
PC7	37	17		
PC6	36	16		
	35	15	PA0	
	34	14	PA1	
	33	13	PA2	
	32	12	PA3	
(Port B strobe) PC2	31	11	0 V	
0 V	30	10	PA4	
	29	9	PA5	
PB7	28	8	PA6	
PB6	27	7	PA7	
PB5	26	6	PC5	(Port A ready)
PB4	25	5	PC4	(Port A strobe)
PB3	24	4	PC3	
PB2	23	3	PC1	(Port B ready)
PB1	22	2	PC0	
PB0	21	1	+5 V	

Serial Socket, P3, connections



Expansion socket, P4, connections

P4 is a 50 way IDC male connector. Pin 3 is not used so may be cropped to ensure proper polarisation.

The pins are as viewed from the component side of the board, with the nearest board edge on the right.

P4

+5 V	1	2	+5 V	
not used	3	4	NMI	
HOLD	5	6	HOLDA	
AD15	7	8	PCLK	
A16	9	10	AD14	
A17	11	12	AD13	
A18	13	14	AD12	
A19	15	16	AD11	
$\overline{BHE}$	17	18	AD10	
AD8	19	20	AD9	
AD7	21	22	$\overline{RD}$	
AD5	23	24	AD6	
AD4	25	26	$\overline{WR}$	
AD3	27	28	M/$\overline{IO}$	
AD1	29	30	AD2	
AD0	31	32	ALE	
RESET	33	34	$\overline{INTA}$	
($\overline{CSD}$) I/O $\overline{Y7}$	35	36	I/O $\overline{Y6}$ ($\overline{CSC}$)	
($\overline{CSB}$) I/O $\overline{Y5}$	37	38	I/O $\overline{Y4}$ ($\overline{CSA}$)	
Ground	39	40	Ground	
not used	41	42	not used	
not used	43	44	not used	
not used	45	46	not used	
DT/$\overline{R}$	47	48	$\overline{TEST}$	
$\overline{DEN}$	49	50	CLK	

Host to FLIGHT 86 Connection

The board is connected to the host PC via an RS232 link terminating in P3, on the controller board, and the standard RS232 socket on the PC.

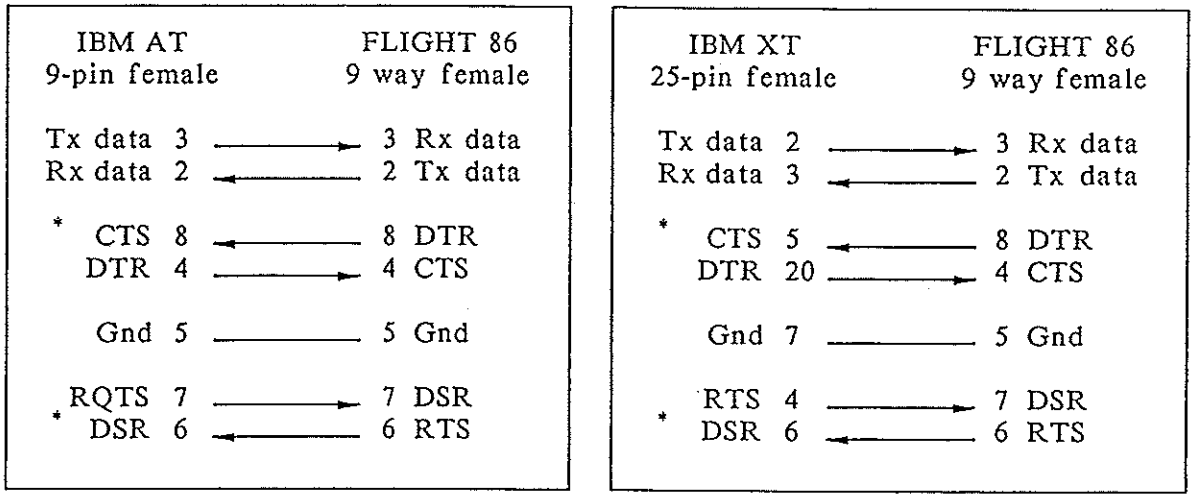
A 9 way female to 25 way female cable (part no. 998-009) is supplied with the Flight 86 system. Alternatively, a 9 way female to 9 way female cable can be supplied on request (part no. 998-014). P3 is a 9 way male 'D' type connector which uses the same pinout as the serial port used by the IBM AT and clones, detailed wiring opposite. The pins used on the controller board are shown below (others are not connected):

PC PIN OUT NAMES

1 not used (Data Carrier Detect)
2 Receiver data input
3 Transmitter data output
4 Data Terminal Ready
5 Ground
6 Data Set Ready
7 Request to Send
8 Clear to Send
9 not used (Ring Indicator)

The communication software does not use hardware handshaking, so the minimum handshake wiring required is that needed to allow the PC to send and accept data. A standard printer lead with a gender changer at the controller board end will usually suffice.

For IBM PC's and clones a suggested lead wiring is shown below for both a 9 and a 25 way connector:

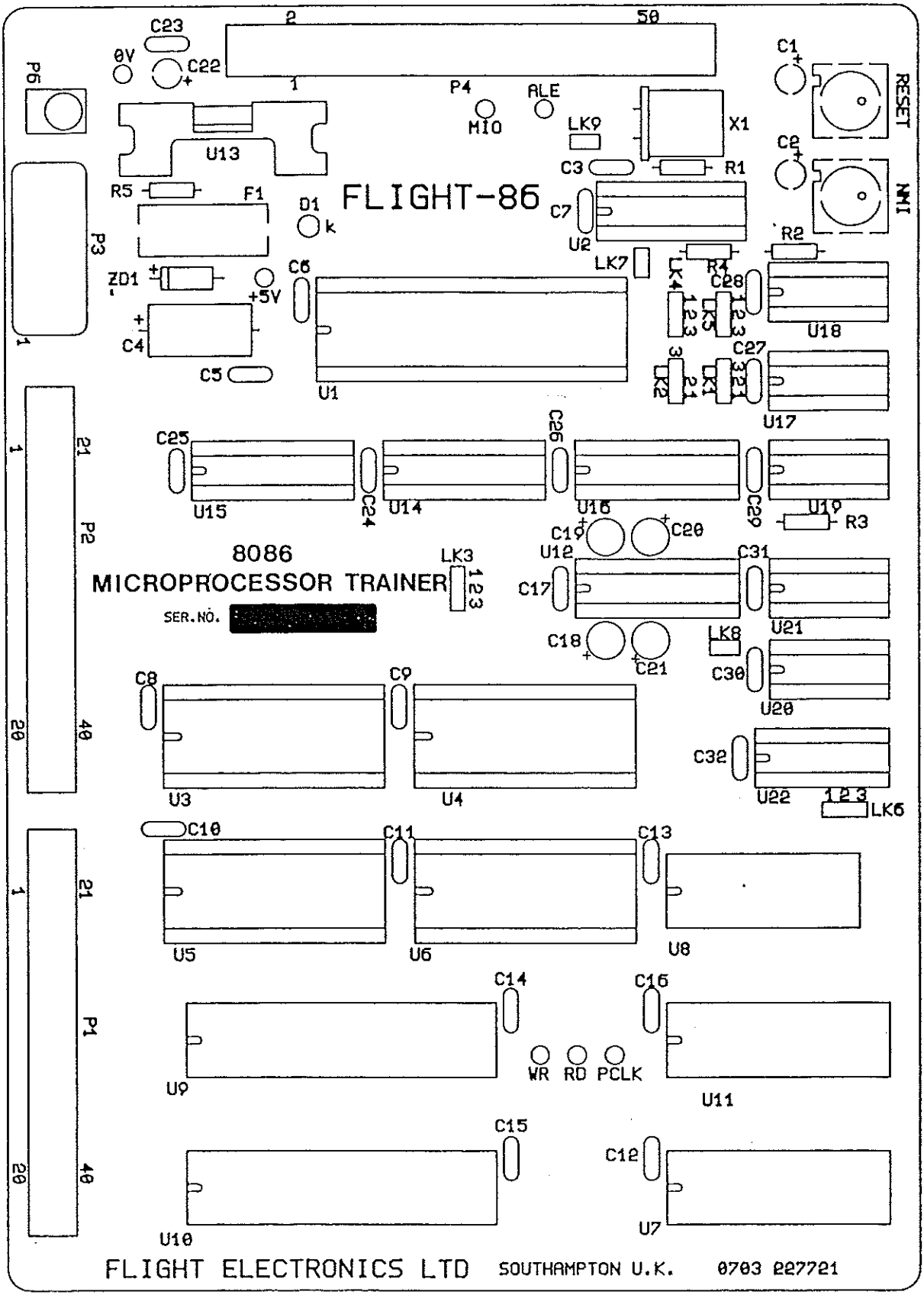


* DSR and CTS may be linked at the computer end to provide a null-modem connection. As hardware handshaking is not required by the software, this is usually perfectly satisfactory.

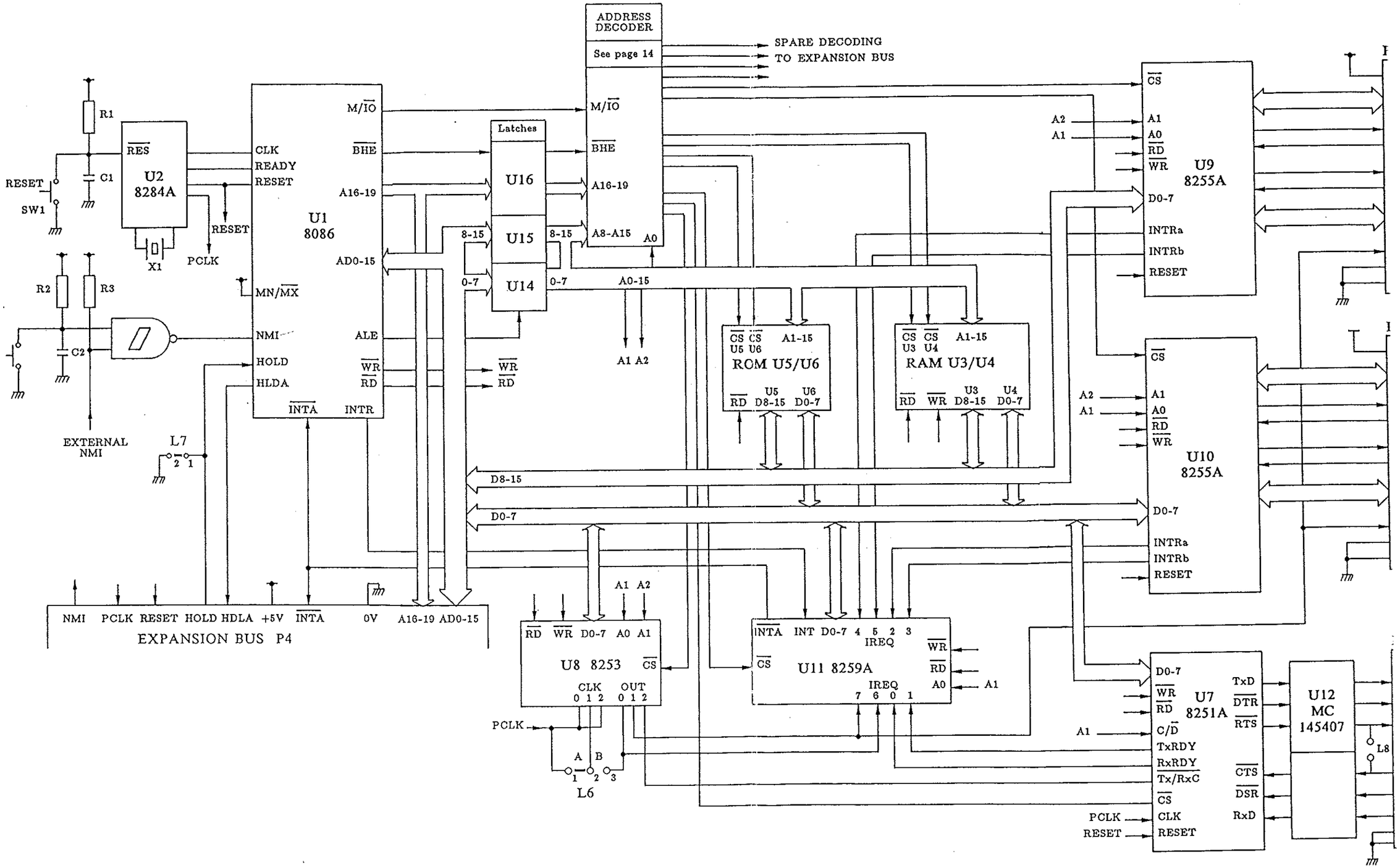
The arrangements given above should work for all IBM AT and XT clones. For non-IBM compatible machines the link should be similar. If in doubt please contact your supplier for advice.

Communication defaults to 9600 BAUD, but may be readily changed using the host TS command once communications have been initiated.

BOARD COMPONENT LAYOUT



FLIGHT 86 TRAINER - FUNCTIONAL BLOCK DIAGRAM



CPU & LATCH CIRCUIT

U1, the 8086, is operating in minimum mode with its timing derived from U2, 8284A, CGD (Clock Generator Driver).

Clock Generator Driver

The 8284A CGD provides the RESET, CLOCKS and READY signals for the system.

The system reset is derived from the R1, C1 combination, either by applying dc power or pressing the RESET button.

The basic system timing is derived from the 14.7456 MHz quartz crystal. This frequency is divided by three to give the CLK for the 8086, with a 1:2 mark-space ratio. This is further divided by 2 to provide the symmetrical PCLK for on-board devices such as the 8251A and the 8253.

The 8284A is permanently enabled by taking RDY2, F/$\overline{C}$, etc. to ground. So, the READY input to the 8086 is simply synchronised to the crystal and the RESET inputs.

Address/Data Bus and Latch Circuitry

The 8086 multiplexed 16-bit data bus AD0 to AD15 is available to the rest of the board and to the expansion connector, P4.

The 8086 ALE signal is used to latch the AD0 to AD15, A16 to A19 and $\overline{BHE}$ signals into the 74LS373 D-type latches to provide demultiplexed address lines for the board. The ALE line is available on P4, the external expansion socket, to allow external latching to take place.

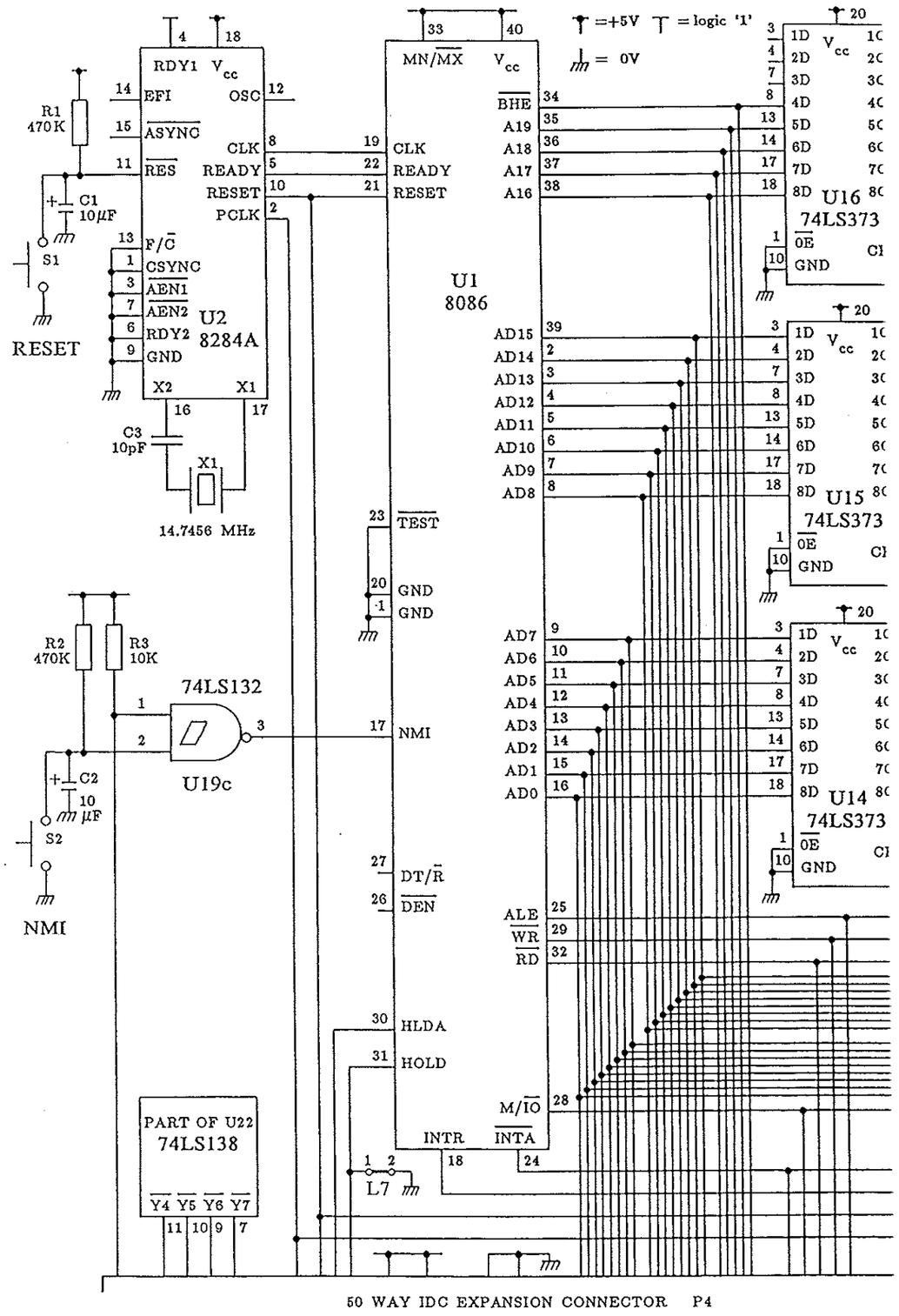
Interrupts and Hold

Either the on-board NMI button or an external signal, from P4, may be used to apply non-maskable interrupt control to the 8086. These signals are passed through a schmitt NAND gate (U19c) to ensure spurious noise does not cause repetitive interrupts.

INTR is hard wired to the 8259A, Programmable Interrupt Controller, providing maskable interrupt capability for the on-board peripheral devices. $\overline{INTA}$ returns to the 8259A and is provided at the edge connector to allow external devices to be aware of any on-board interrupts.

For an external device to take control of the board it must raise the HOLD line. To allow this to happen link L7 must be removed.
NOTE: A suitable pull up resistor must be provided on the external board.

See page 6 for details of the P4 connections.



50 WAY IDC EXPANSION CONNECTOR P4

ADDRESS DECODER CIRCUIT

The address decoder provides 'unique' decoding for the ROM and RAM devices and partial decoding for PORT devices. The decoded signals are active low and are unique to allow for full expansion within the 8086 1 Mbyte addressing range. See the simplified memory map on page 15.

RAM decoder

The RAM decoder has three links provided to allow for larger devices than the 6264 RAMs normally supplied. Using the 6264 the address lines A14 to A19 are 'nanded' using U17 and U19 with the M/$\overline{IO}$ control line. Thus the RAM cannot be selected if the M/$\overline{IO}$ line is low.

If the 62256 RAM chip is used then the address lines that are now used by the chip are disabled in the address decoder. A13 is a special case in that it is normally held high for the CS2 of the 6264. With the larger device A14 is allowed through to the chip address inputs. The Link table shows the linkage required for the two RAM chips capable of being used with the board.

The $\overline{BHE}$ and A0 lines are used, via U20, to select the odd or even RAM chip select lines. Under normal circumstances both RAMs are selected as a 16-bit word.

ROM decoder

The ROM decoder has two links provided to allow for larger devices than the 2764 EPROMs normally supplied. Using the 2764, the address lines A14 to A19 are 'nanded' using U18 with the M/$\overline{IO}$ control line. Thus the ROM cannot be selected if the M/$\overline{IO}$ line is low.

If larger EPROMs are to be used then the address lines that are now used by the chips are held high in the address decoder. The Link table shows the linkage required for the different types of EPROM capable of being used with the board.

As with the RAM the $\overline{BHE}$ and A0 lines are used, via U20, to select the odd or even ROM chip select lines. Under normal circumstances both ROMs are selected as a 16-bit word.

PORT decoder

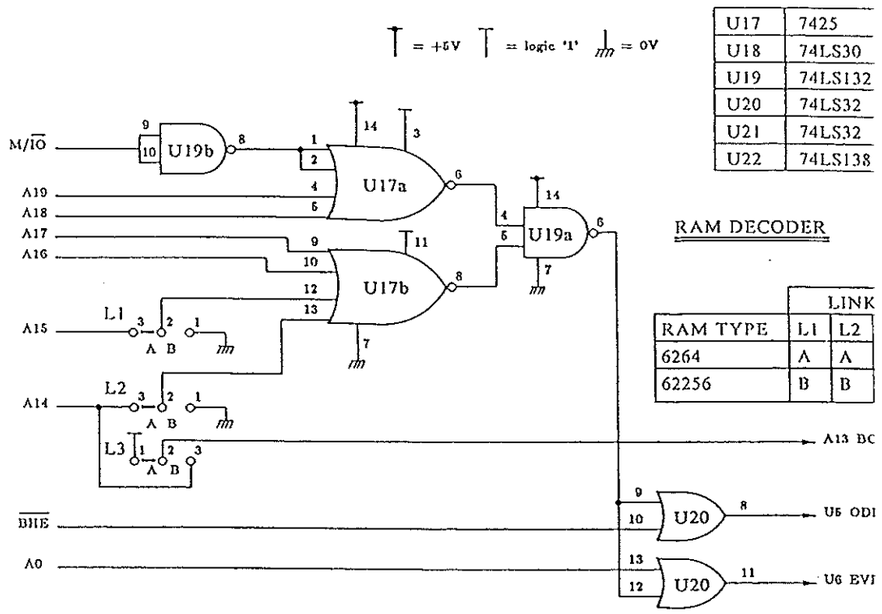
The port decoder uses U22, 74LS138, a 3 to 8 decoder, to provide the chip select signals required for the 8255A, 8253, 8251A and 8259A I/O devices from the address lines A3 to A5.

The M/$\overline{IO}$ line must be low to activate the decoder circuit. $\overline{BHE}$ and A0 are used to select between the two 8255A devices. U10 is connected to the low data bus and U9 is connected to the high data bus. All other peripheral devices are connected to the low data bus, so are decoded to even addresses.

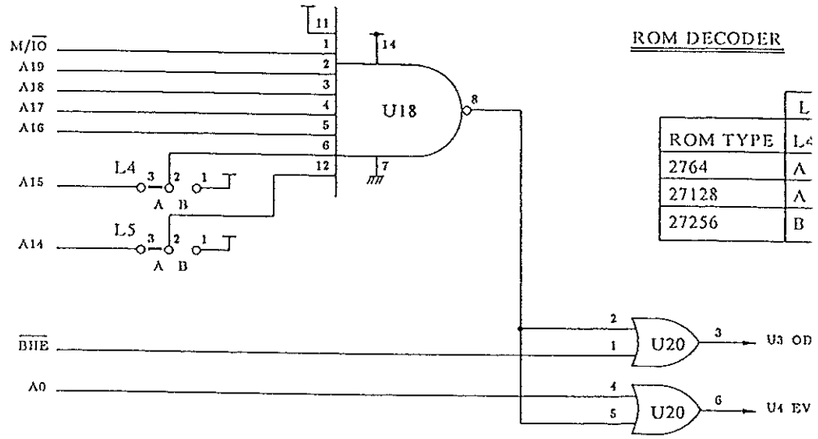
Spare decoded outputs are available on the expansion connector, P4.

They are $\overline{Y4}$ = 00100xxx, $\overline{Y5}$ = 00101xxx, $\overline{Y6}$ = 00110xxx, $\overline{Y7}$ = 00111xxx.

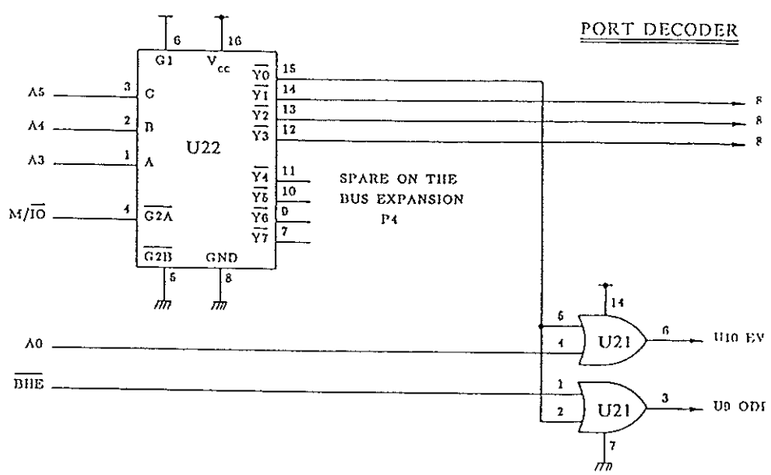
U17	7425
U18	74LS30
U19	74LS132
U20	74LS32
U21	74LS32
U22	74LS138



	LINK	
RAM TYPE	L1	L2
6264	A	A
62256	B	B



	L
ROM TYPE	L4
2764	A
27128	A
27256	B



MEMORY CIRCUIT

Two 8-bit EPROM and two 8-bit RAM chips are used across the 16-bit data bus.

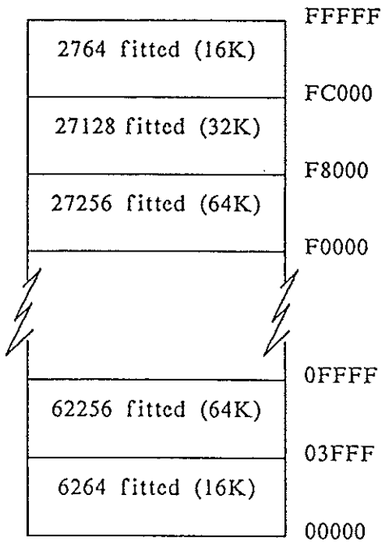
The 8086 demultiplexed address bus, A1 to A15, is connected to A0 to A14 on the chip sockets. The smaller devices do not use all of these address lines but they are provided to allow larger chips to slot in.

The inset pin diagrams show the difference in action for the various memory devices. The wiring is not changed, only how the pins are used, provided suitable links are changed for the address decoding. See table on page 1.

The address decoder outputs are shown going to each of the appropriate memory sockets. These signals are active low and are unique to allow for full expansion within the 8086 1 Mbyte addressing range.

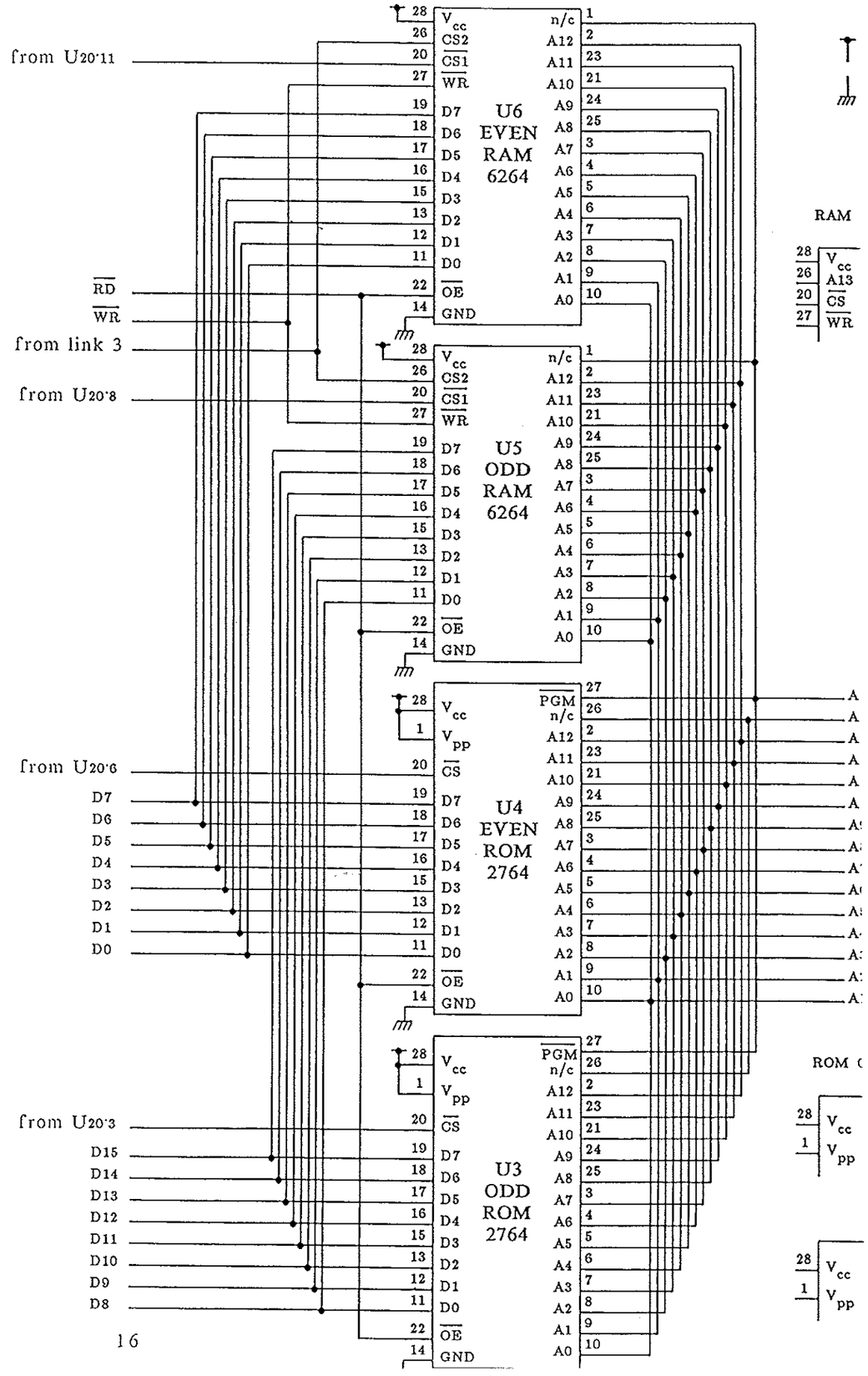
The RAM decoding starts at address 00000 and the ROM decoding starts at F0000. The links L1 to L5 provide tighter mapping within these boundaries.

A simplified memory map:



The EVEN memory devices, U4 and U6, are connected to the LOW data bus, D0 to D7, and the ODD memory devices, U3 and U5, are connected to the HIGH data bus, D8 to D15.

Link 3 is wired to logic '1' for the 6264 RAM chip, and to A14 when the 62256 is used.



PARALLEL I/O, TIMER AND PIC CIRCUIT

Programmable Peripheral Interface (PPI)

Parallel Input/Output Interfacing may be established through either U9 or U10, 8255A's, the PPI devices, and sockets P2 and P1.

Connection to the 8086 bus is via the low 8 data bits, D0 to D7, for U10 and the high 8 data bits, D8 to D15 for U9. A0 and A1 are used to select the programmable registers. The 8086 $\overline{RD}$ and $\overline{WR}$ lines are used to handle data transfer to and from the bus. The $\overline{CS}$ line for U10 is decoded to even port addresses, U9 to odd, 00000xxx.

Connectors P1 and P2, female 25 way 'D' type connectors, are wired direct to the 8255A ports A, B and the strobe lines of port C. To assist in the design of parallel peripheral hardware, +5V, 0V and timer 1 output from the 8253 PIT are also provided.

The 8255A's may be used in the strobed mode so PC0 and PC3 of each 8255A are made available to the 8259A Programmable Interrupt Controller. See page 4 for priorities.

Programmable Interval Timer (PIT)

Connection of U8, 8253, to the 8086 bus is via the low 8 data bits, D0 to D7. A0 and A1 are used to select the programmable registers. The 8086 $\overline{RD}$ and $\overline{WR}$ lines are used to handle data transfer to and from the bus. The $\overline{CS}$ line is decoded to even port addresses 0001xxx.

Clock inputs 0 and 2 of the 8253 PIT are hard wired to the system PCLK, from the 8284A.

Link 6 allows either PCLK (position A), or the output from timer 0 (position B), to be used for clock input 1, giving the capability of generating very long delays if required. Timer 1 output is available at the sockets P1 and P2.

Timer 2 output is dedicated to the 8251A USART as a programmable baud rate generator. Timer outputs 0 and 1 are available to the 8259A PIC as programmable interrupts. See page 4 for priorities.

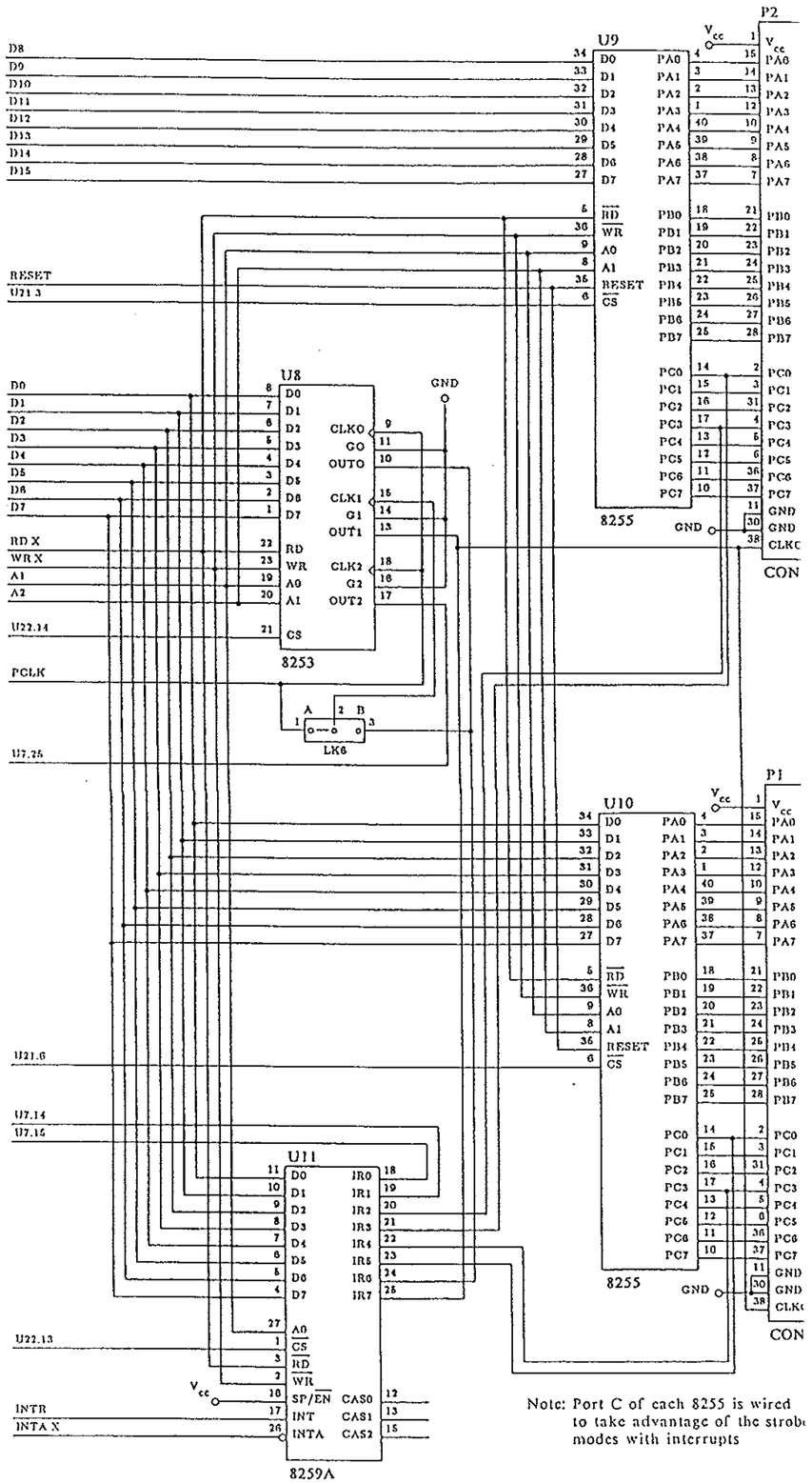
All 3 timer circuits are permanently enabled by taking the gate inputs high.

Programmable Interrupt Controller (PIC)

Connection of U11, 8259A, to the 8086 bus is via the low 8 data bits D0 to D7. A0 is used to select the programmable registers. The 8086 $\overline{RD}$ and $\overline{WR}$ lines are used to handle data transfer to and from the bus. The $\overline{CS}$ line is decoded to even port addresses 00010xxx.

Eight levels of interrupt are hard wired to the 8259A, with IR0 the highest and IR7 the lowest. See table on page 4.
Interrupts are signalled to the 8086 via the INTR line. Two successive INTA pulses from the 8086 force the vector address of the service routines to be placed on the bus by the 8259A.

The cascade lines $\overline{CAS0}$ to CAS2 are not used and the 8259A is wired in the 'master' mode by taking $\overline{SP/EN}$ high.



SERIAL CIRCUIT

Serial Communications may be established through U7 (8251A), the USART (Universal Synchronous/Asynchronous Receiver/Transmitter) and socket P3.

Connection to the 8086 bus is via the low 8 data bits, D0 to D7. A1 is used to select either the Control/Status register or the Data register. The 8086 $\overline{RD}$ and $\overline{WR}$ lines are used to handle data transfer to and from the bus. The $\overline{CS}$ line is decoded to even addresses 00011xx0. Whenever the system is RESET, the 8251A is reset via pin 21, so the software must re-initialise the chip.

RxC and TxC are the clock signals from the 8253 PIT, clock 1, used as the baud rate generator. The monitor software conditions this clock to 153.6 kHz for 9600 baud signalling. RxRdy and TxRdy are interrupt signals to the 8259 PIC. These may be used for interrupt driven transmit or receive routines.

Connection to P3, the RS232 line, is via level shifters. U12, MC145407 generates internally +12 V and -12 V to provide signalling between these levels for the line. U12 also provides shifting from these levels to TTL for incoming signals.

P3 is a male 9 way 'D' type connector providing emulation of an IBM AT or clone serial port.

L8 is normally linked providing loopback to the 8251A for those not requiring the use of control lines RTS and CTS. To use these control lines L8 must be removed.

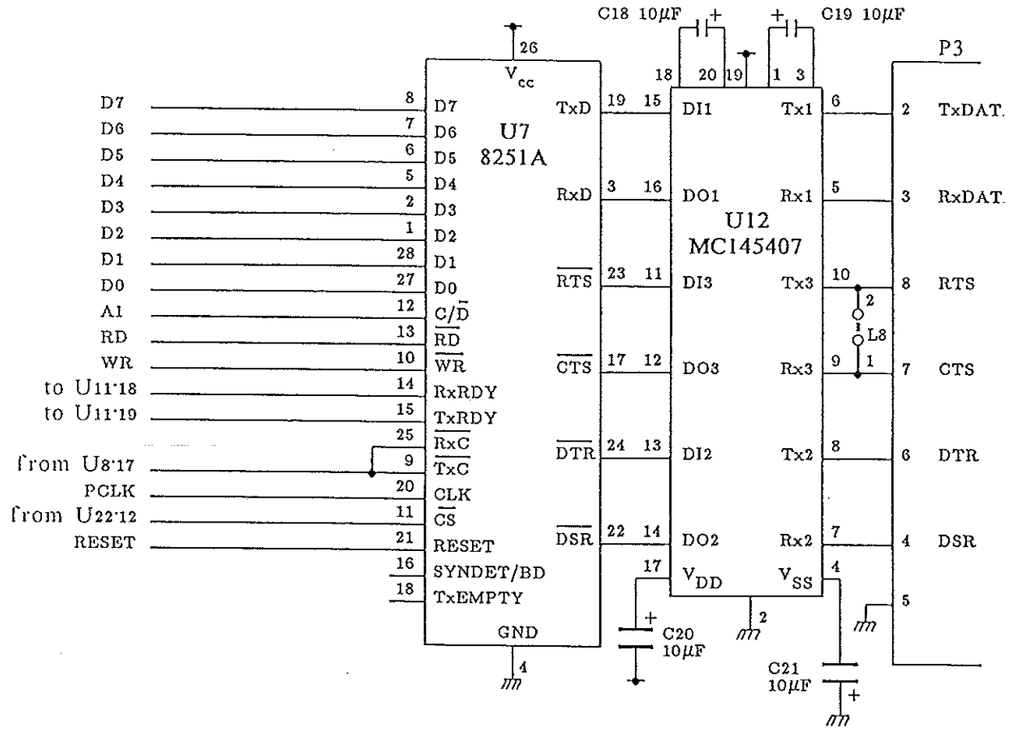
POWER SUPPLY

The controller board power supply current requirements is 9 V supplied via P6. This is regulated to +5 V by Voltage Regulator U13 for use on the board.

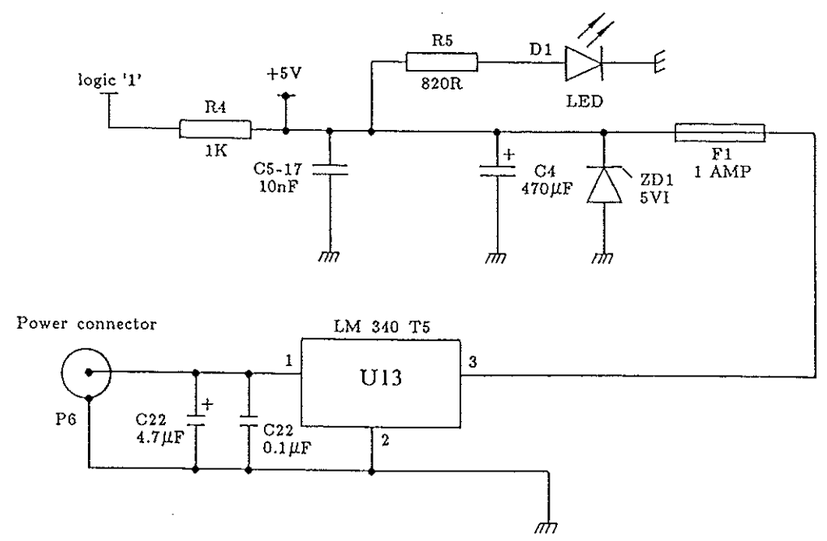
The 5V supply is, partially, over-voltage protected by the 'crowbar' zener diode ZD1 and F1 the 1 amp fuse. C4 provides major supply decoupling, with C5 to C17 liberally scattered around the board for individual chip decoupling.

The power is provided to the board via the connector P6 on the circuit diagrams, where it is regulated to +5 V. A second rail is provided via R4 to give a logic '1' for various logic conditions around the rest of the circuit.

FLIGHT 86 TRAINER - SERIAL CIRCUIT



FLIGHT 86 TRAINER - POWER SUPPLY



SOFTWARE

Getting started

The first thing to do is check you have the correct software for your machine. The disk supplied is normally a 5¼" 360k MSDOS format. Alternative software will also be supplied on request, on a 3½" disc. If you specified another disk format please refer to the additional information supplied with the disk for your machine.

The following notes assume you are using an IBM AT, XT or clone.

Boot-up your computer system and take a directory of the floppy disk supplied with the board. You should have the following files:

FLIGHT86.COM	the executable program file
FLIGHT86.MSG	the ASCII error and system message file (details page 73)
F86GO.BAT	a sample BATCH file (details page 25)

If you don't have all three files, you should contact your supplier.

Optionally there may be a fourth file on the disk:

README	this contains any additional information not supplied in the technical reference manual concerning upgrades.

The disk supplied is NOT copy protected, and you are urged to take a BACKUP copy, and keep the master in a safe place. Accidents do happen, and usually when you haven't got a copy!

Host Software Installation

Having established you have the correct software, and made your working copy, you now need to check a few points before you attempt to use the system.

1. The Serial Link.

   The link with the board is via an RS232 serial port, and this must be available. If your computer has the COM1: serial port available and you are using the LPT1: parallel port for the printer, you may go to note 4.

   Otherwise there are three common scenarios:

   a) You are using a parallel printer but your computer doesn't have a serial port fitted.

      You will need to purchase and install a serial port. This is usually a simple matter and your supplier will be pleased to advise if necessary. Once this is done you may go to note 4.

   b) You are using a serial printer and your computer has a second serial port available, you should now read note 2.

   c) Your computer has only one serial port but it is used for something else, such as a printer.

      In this case you need to purchase and install a second one. This is usually a simple matter and your supplier will be pleased to advise if necessary. You should now read note 2.

2. The Serial Ports

   An alternative serial port to the COM1: may be specified by changing the contents of the FLIGHT86.MSG file. See page 73 for details.

3. The Printer

   The software uses the LST device, this is normally the LPT1: parallel printer port. If you are using a parallel printer connected to the LPT1: port this does not affect you, go to note 4.

   If you have printer connected to another port, serial or parallel you will need to redirect the printer output to the appropriate port. This is readily achieved using a BATCH file, see page 24 for details on redirecting the printer.

4. Default disk drive

   The software supplied operates from a floppy disk, but if your machine has a hard disk, you may prefer to operate from there. This is easily done by setting up a sub-directory and using another BATCH file. See page 24 for details of using a hard disk.

   The only constraint for the software operation is that the 2 files FLIGHT86.COM and FLIGHT86.MSG must be on the same disk, and either in the root directory or in the same sub-directory.

5. Error and System Messages

   To allow for changes to users own wording, and possible language translation of messages, the software reads its command information, error and system messages from an ASCII file, FLIGHT86.MSG, when it first starts up.

   You may edit this file, but take extreme care. See page 73 for details.

Host To Controller Board Communication

Starting Communications

To establish Host to Controller communications follow this set-up procedure:

1. 'Boot-up' your Host PC in your normal way.

   Ensure you have the DOS prompt A> or C> depending on your system.

2. If you are using a floppy disk PC, insert the FLIGHT 86 disk into drive A and close the door. If you are operating from a hard disk you need do nothing more at this stage.

3. Turn OFF your Flight 86 board power supply.

4. Connect the serial lead between the serial socket to be used on the PC and socket P3 on the Flight 86 board. See page 7 for the lead details.

5. Turn on the Flight 86 board DC power supply.

   NOTE: the board indicates that power is actually applied by illuminating the green LED, D1.

6. If you are using a floppy disk on a standard machine you may type FLIGHT86.

   If you have prepared a BATCH file type F86GO.

   If using a hard disk type F86HGO.

7. If using a BATCH file you should see the message 'Loading FLIGHT 86 host program, please wait. . . . ' displayed while the program is being loaded from disk. After a few seconds you should see the following messages:

   ```
   FLIGHT86 Controller Board.  Host Program Version 2.0.
   Press ? and Enter for help - Waiting for controller board response. . .
   ROM found at F000:C000 to F000:FFFF FLIGHT Monitor ROM version 2.0
   RAM found at 0000:0000 to 0000:3FFF
   -
   ```

   Note: These messages may be slightly different depending on the software versions and the size of ROM or RAM fitted.

8. The '-' is the host prompt.

9. You have control over the controller board once this prompt is displayed.

   You are ready for action. If you do not see the prompt, you do not have communication with the controller board. (If this occurs during general usage it probably means a command is in progress.)

   If the above message stopped at 'Waiting for controller board response. . . ' turn the Controller board power supply OFF, wait a few seconds, and turn it ON again. You should then see 'Controller Reset' followed by the memory test messages.

Some Common Problems

If you have difficulty setting up communications try the following:

a) Check ALL three power supplies are connected CORRECTLY, including the 0V, to the controller board. Check the fuse on the controller board.

b) Check the link settings on the board are correct for the chips used and your expected usage. The board is shipped with links L1 to L6 in position A and the links L7 and L8 both fitted.

c) Check you have a serial lead (properly wired) between the host and the controller board. Ensure it is firmly connected to the correct port.

d) Check you are using the COM1: port. If not, ensure you have altered the FLIGHT86.MSG file to the required port.

   If possible, try to ensure you have communications under the default conditions before making any 'improvements' to the FLIGHT86.MSG file.

e) If the host messages do not appear, make another copy of the master disk supplied and repeat the installation notes on page 21.

General Communications

During normal usage the '-' prompt indicates when you may enter commands at the Host PC keyboard.

If this prompt does not appear when you expected it to, the host has lost control of communications with the controller.

The usual reasons for this are:

A program was executed without a valid return to the monitor. See page 33 and the G command for suitable options.
A program has over-written the monitor variables in monitor RAM.
The serial communications baud rate has been changed on the controller without informing the host!!

To restore communications, press the Esc key on the host, press the RESET button on the controller board, then press X and the Enter key on the host. You should see the normal FLIGHT 86 'boot message' and the results of the ROM and RAM tests. As these tests are non-destructive, any data in user memory will not be touched.

If you do not press the RESET button after pressing the Esc key you will see the prompt, but will NOT have communications. You MUST press the RESET button.

Using a hard disk

All the notes applying to the floppy disk apply to the hard disk except the normal drive is C instead of A and a sub-directory is recommended.

To store the program files on the hard disk, log onto the hard disk.
Make a fresh sub-directory using the command MD\FLIGHT86.
Log onto this sub-directory using the command CD\FLIGHT86.
Copy the files from the floppy disk, in drive A, using COPY A:FL*.*.

To get best results a BATCH file is suggested. See a sample listing opposite, F86GO.BAT. This file should be created using a non-document mode text editor and put into the ROOT directory of the hard disk.

Now you can boot-up the machine with the hard disk in the normal way, and when the C> prompt appears type F86HGO to execute the software. The batch files should return you to the same point when you Quit the FLIGHT 86 software.

Redirecting the Printer

To redirect the printer output to another port is a simple matter. It may be achieved using the MSDOS MODE command. If you are using a floppy disk system you must copy the MODE.EXE file from your MSDOS system disk to your FLIGHT 86 disk.

To redirect the printer output to the COM1: port use MODE LPT1:=COM1:,,P

To make the process automatic a sample BATCH file, F86GO.BAT, is included on your FLIGHT 86 disk, listed opposite. Edit this file, with a non-document mode text editor, to suit the redirection required. If you have difficulties, please contact your supplier for assistance.

F86GO.BAT

A sample batch file. Useful if you need to redirect the printer output or wish to change default parameters.

Lines starting with REM are treated as comments until the REM statement is removed.

```
echo off
cls
echo Loading FLIGHT86 host program, please wait. . . .
REM ---------------------------------------- Test for program file
if exist eiolp.com goto :ok1
echo Program file FLIGHT86.COM missing - operation abandoned
goto err1
REM ---------------------------------------- Test for message file
:ok1
if exist eiolp.msg goto :ok2
echo Message file FLIGHT86.MSG missing - operation abandoned
:err1
echo Both FLIGHT86.COM and FLIGHT86.MSG files must be on the default disk drive
goto fini
:ok2
REM ---------------------------------- redirect printer if necessary
REM the following is an example line for using COM1: for the printer
REM mode lpt1:=com1:,,P > nul
REM ---------------------------------------- execute host program
eiolp
REM ---------------- when finished restore original port settings if necessary
REM the following are example lines to restore the PC to its original settings
REM echo Restoring original printer settings
REM mode com1:9600,N,8,1,P > nul
REM mode lpt1: > nul
:fini
REM ---------------------------------------- return to root directory
cd\
```

FLT86HGO.BAT

A suitable BATCH file for accessing the software from the ROOT directory of a hard disk system.

```
cls
echo
REM ----------------- change to sub-directory containing FLIGHT86 files
cd\flight86
REM ---------------------------------- execute host software batch file
flt86go
```

Extended Intel Hex format

This is a line orientated industry standard ASCII communications data format. As the software will accommodate a line of up to 128 characters, the 'normal' 16 or 32 data bytes created by most assemblers will be easily handled.

It is the format displayed on the screen by the both the Upload and Download commands.

Each line is terminated in a carriage return code and all the ASCII characters are in the range 0. .9 and A. .F. The fields are defined as:

:nnaaaattdddddd. . . . . .ddddcc

where	:	is the record start
	nn	is the data byte count
	aaaa	is the address of the first data byte in the data field
	tt	is the type of record
		00 indicates a data record
		01 indicates an end record
		aaaa will be 0000, dd is not used
		02 indicates a segment address record
		aaaa will be 0000
		dddd will be the CS address
		03 indicates a start address record
		dddddddd becomes the CS:IP start address
	dd	are the actual data bytes
	cc	modulo 2 checksum of the entire line (except the :)
		This means that when the entire line including the checksum is added up, and the sum ANDed with FF, the result will be zero.

The monitor software handles each line individually, so the end record line is not essential, although most assemblers create one. This means that more than one file, or parts of files, may be down loaded to 'set-up' different areas of memory.

Examples

To show examples of these lines in use it is assumed that the the memory locations from 0123:1240 to 1253 contain the following:

```
-M 123:1240 1253

          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F     Ascii-code
0123:1240 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF  .."3DUfW........
0123:1250 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
```

If this data is now uploaded to a file SAMPLE.HX using the U command you will see the following on the screen:

```
-U SAMPLE 123:1240 1253
:020000020123D8                                  (line 1)
:10124000001122334455667788 99AABBCCDDEEFFA6     (line 2)
:04125000FFFFFFFF9E                              (line 3)
:00000001FF                                      (line 4)
```

Taking these lines in turn (and expanding out the relevant fields):

line 1

```
          : 02 0000 02 01 23 D8
          : nn aaaa tt dd dd cc
```

record start
nn = 02 — 2 data bytes
aaaa = 0000 — because type 02
tt = 02 — segment record
dddd = 0123 — segment address
cc = D8 — checksum

(02+00+00+02+01+23+D8 = 100 AND FF = 00)

line 2

```
          : 10 1240 00 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF A6
          : nn aaaa tt dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd cc
```

record start
nn = 10 — 16 data bytes
aaaa = 1240 — offset address
tt = 00 — data record
dddd = — actual data bytes
cc = A6 — checksum

(10+12+40+00+00+11+22+33+44+55+66+77+88+99+AA+BB+CC+DD+EE+FF+A6 = 900 AND FF = 00)

line 3

```
          : 04 1250 00 FF FF FF FF 9E
          : nn aaaa tt dd dd dd dd cc
```

record start
nn = 04 — 4 data bytes
aaaa = 1250 — offset address
tt = 00 — data record
dddd = — actual data bytes
cc = 9E — checksum

(04+12+50+00+FF+FF+FF+FF+9E = 500 AND FF = 00)

line 4

```
          : 00 0000 01 FF
          : nn aaaa tt cc
```

record start
nn = 00 — no data bytes
aaaa = 0000 — because type 01
tt = 01 — end record
dddd = — none because type 01
cc = FF — mod 2 checksum

(00+00+00+01+FF = 100 AND FF = 00)

HOST COMMANDS

A '-' prompt on the screen means that the host is ready to accept a command. With the exception of the Esc command, all commands are of the form:

letter (A to Z) space or comma or semi-colon parameter tail Enter or Return

The command line is executed immediately after the Enter or Return key is pressed. Before this it may be edited using the Del key.

All data must be specified in hexadecimal, although leading zeros may be omitted. Spaces are ignored for flexible entry, although the first character of a line must be a valid command letter.

The command line syntax uses squared brackets, [ ], to indicate parameter options. The pipe symbol, |, is used to indicate a choice of parameters, one of which must be used.

If an invalid parameter is entered, the full command syntax and help line is displayed to assist. If it is not possible to identify the intended command, you get a terse 'invalid command' message.

The following sample command lines have the same effect:

```
M  W  0001  0233
M, W, 1,233
M     W     1;0233
```

They all display the contents of memory between 0001 and 023F in word format, for the default segment.

Command Summary

MEANING	KEY	PARAMETERS	DESCRIPTIONS ON PAGE
Escape	Esc		29
Reset	X		29
Test	T	M | R | C | S ? | baud	32
Register	R	[register]	31
Memory	M	[W] [segment:] address1 [address2]	30
Fill	F	[W] [segment:] address1 address2 data	31
Port Input	I	[W] address	32
Port Output	O	[W] address data	32
Go	G	[[segment:] address]	33
Breakpoint	B	R | ? | S [segment:] address	33
Single Step	S	[R] [[segment:] address]	33
Upload	U	[drive:\path\] file[seg:] add1 add2	34
Download	:	[drive:\path\] filename	34
Directory	D	[drive:\path\] [file mask]	34
Help	?	[Command letter]	29
Printer	P		29
Quit	Q		29
Assemble	A	[[segment:]address1]	35
Disassemble	Z	[[V] [segment:] address1[address2]]	37
Hex Arithmetic	H	number1 number2	38

Command Descriptions

Where appropriate the commands are described by way of specific examples, with the command line first, followed by a brief explanation.

Help - ? [Command letter]

Limited help is available through this command.

? displays the command menu with command syntax.
? M displays the command syntax and a single line of help for the Memory command.
All the commands have an associated single line of help available.

Escape - Esc

Press the Esc key to stop the current command. This does not work for the GO command. If you wish to return to the monitor from your program use the INT 5 instruction at a suitable point in your code, or set a break point using the B command.

Reset - X

Type X to 'Warm' reset the controller board. This sets up all the Monitor variables, including the serial communications speed (back to 9600 baud), and places the controller board into a known working condition. It does not affect the current state of the user registers so may be carried out at any time. The controller memory is retested.

Note: this command will NOT reset the controller board if communications have been 'lost'. Press the board RESET button first, if this is the case.

Printer - P

Type P to echo all the screen output for subsequent commands to the printer. You will be prompted to check that the printer is ONLINE; if this message persists your printer is probably OFF and not online.

The command assumes a printer is connected, turned on, loaded with paper and set on-line. If this is not the case, the lack of printer may 'lockup' the computer. The host assumes the printer is the LST device, normally the parallel printer, but this may be redirected using the DOS MODE command. See page 24 for details.

Type P again to stop the screen echo, and stop output to the printer.

Quit - Q

Terminates running of host software and returns control to the operating system.

Memory - M [W] [segment:] address1 [address2]

Usage 1:

The first use of the M command allows blocks of memory to be displayed in byte/ASCII or word/ASCII form.

Two addresses must be specified, the start and end address of the block. Address1 MUST be less than or equal to address2, this limits the display to a 64k segment.

Complete lines of data are displayed with the ASCII equivalent on the right hand side. Complete lines are displayed even if just one byte is requested. If the screen becomes full, a prompt message will appear asking you to press a key to continue. This will continue until the command is completed.

You may return to the monitor prompt by pressing the Esc key while the display is in progress.

M 100 1FF — If the user default segment is 0050 then this will display the contents of address 0050:0100 to 0050:01FF. The display shows the byte stored at each address and the ASCII equivalent.

M W 100 1FF — displays the 16-bit word (byte reversed) at each address and its ASCII equivalent. This is ideal for examining look up tables etc.

The default segment may be overridden by specifying the segment required.

M W 100:20 3FF — displays the word data from 0100:0020 to 0100:03FF
M F000:E803 E905 — displays the byte data from F000:E800 to F000:E90F.

Usage 2:

The second use of the M command allows access to the memory to change the data byte or word stored there. Only one address is specified.

Once the command has started press "Enter" to go to the next address or press the space bar and "Enter" to go back to the previous address. You may return to the monitor prompt by pressing the Esc key.

M 104 — displays the byte contents of address 0050:0104 (assuming the segment default of 0050) and waits for the user to enter the new data byte required at this address.
M W 104 — allows the 16-bit word at that address to be altered.

The default segment may be overridden by specifying the segment required.

M 100:20 — starts the byte editing at address 0100:0020
M W 0:0 — starts the word editing at address 0000:0000

Fill - F [W] [segment:] address1 address2 data

The purpose of this command is to fill a block of memory with specified byte or word data. This command will fill a single byte to a 64k block.

When the command is completed, you will be returned to the monitor prompt.

F 100 200 42 — If the user default segment is 0050, this will change the contents of each address in the range 0050:0100 to 0050:0200 to 'B'.

F W 100 200 1234 — stores the 16-bit word (byte reversed) at each address in the range.

The default segment may be overridden by specifying the segment required.

F W 100:20 1FF 12 fills from 0100:0020 to 0100:01FF with 0012.

Register - R [register]

The register command allows the user registers to be displayed or altered.

Usage 1:

The first use of the R command is to display all user registers.

R displays the contents of all 14 user 8086 registers.

AX	BX	CX	DX	SP	BP	SI	DI	DS	ES	SS	CS	IP	FLAG
0000	0000	0000	0000	0500	0000	0000	0000	0050	0050	0000	0050	0000	0000

Usage 2:

The second use of the R command allows access to individual user registers to change the contents. A valid register must be specified initially.

Once the command has started press "Enter" to go to the next register or press the spacebar and "Enter" to go back to the previous register. The registers will 'wrap around'. You may return to the host prompt by pressing the Escape key.

R CS displays the contents of the CS register and will wait for the new contents to be typed in. Any valid 16-bit word may be specified.

You may change the default segment by changing CS.

Note: The segment registers CS, DS, ES, SS and the SP default to those shown above, when the controller board is reset.

Test - T M | R | C | S ? | [baud]

The Test Command performs a number of housekeeping tasks. It allows tests of the controller board memory to be made, serial communications speed to be changed and the Console mode to be entered.

T M — a non-destructive RAM test on the controller board may be carried out at any time. A report is issued indicating the start and end addresses of the on-board RAM.

T R — will identify the start of the Monitor ROM and perform an 8-bit check sum of the monitor software only. If this is non-zero a warning is issued with the checksum calculated. The checksum should always be zero unless the Monitor code between F000:E802 and F000:FFFF has been altered.

T C — the host enters the Console mode.

HEALTH WARNING: YOU'RE ON YOUR OWN!!

> All key entries are sent direct to the controller board. All output from the controller board is put onto the screen. This allows users to write routines that access the host screen and keyboard.

Press the Esc key or controller board RESET button to restore to host control, provided you've left enough of the MONITOR variables intact.

T S 9600 — will set the communications speed for the host/controller software to 9600 baud. It is sometimes convenient to use slower data rates particularly for user software development. Valid baud rates are 300, 600, 1200, 2400, 4800 and 9600. This option may not be available on some non-IBM compatible systems.

Warning: If the controller board RESET button is pressed it will default to 9600 baud. This may cause a 'lock-up' if the host is set to another speed. Enter the command T S 9600 to clear the condition.

T S ? — Displays the current baud rate.

Port Input - I [W] address

The purpose of this command is to read a specified port and display the received byte or word. The Port address may be anywhere in the range 0000 to FFFF. No checks are made for port validity, so reading from non-existent ports may give odd results.

I 4 — Reads the byte from Port 04
I W 1A — Reads the word from Port 1A and 1B

Port Output - O [W] address data

The purpose of this command is to output a byte or word to a specified port. The Port address may be anywhere in the range 0000 to FFFF. No checks are made for port validity. Writing to non-existent ports should have no effect, but writing to the control registers could force you to RESET the board.

O 2 2A — Writes 2A to Port 02
O W 04 123 — Writes 01 to Port 04 and 23 to Port 05

Go - G [[segment:] address]

This command allows the user to execute code that has been downloaded into RAM. It is important to realise that once the user code has started there is no means of stopping it from the host keyboard. There are four methods of exiting from user code back to the monitor (in prefered order):

- use an INT 5 instruction at a suitable point in the code
- set an official breakpoint at an opcode address
- insert an unofficial breakpoint INT 3 instruction in the code
- press the RESET button on the controller board.

```
G          Starts execution of code from current CS:IP.
G 100      Starts execution from current 0050:0100 if CS = 0050
G 100:10   Starts execution from 0100: 0010
```

If you get no sensible response from the host keyboard after using this command, type X and "enter", then press the RESET button on the controller board.

Single Step - S [R] [[segment:] address]

This command is provided to allow the user to step through code one instruction at a time for debugging purposes. The display will be the next instruction address and opcode byte with, optionally, the registers too. Once the command has started, pressing the Enter key will execute the next instruction (other keys may work; Enter is preferred).

All these examples terminate with the next instruction address and opcode:

```
S          Executes the instruction pointed to by current CS:IP
S 123      Executes the instruction at 0050:0123, assuming CS = 0050
S 100:AB   Executes the instruction at 0100:00AB
```

The following examples terminate with the register contents as well:

```
S R            Executes the instruction pointed to by current CS:IP
S R 200:234    Executes the instruction at 0200:0234
```

As with the G command, if you get no sensible response from the host keyboard after using this command, type X and "enter", then press the RESET button on the controller board.

Breakpoint - B ? | R | S [segment:] address

This command is intended for debugging user code. A breakpoint is an INT 3 instruction inserted at an opcode position. The original opcode at this address is saved. When the code is executed it runs normally, at full speed, until it reaches this location. The original opcode is restored and the registers, address and first opcode byte are displayed. The user may set another break point and continue with a G instruction, or may single step from this point.

Currently only one breakpoint may be specified at a time

```
B ?          Displays the current breakpoint address
B R          Clears all breakpoints, if any, set
B R 100:20   Clears the breakpoint at 0100:0020
B S 123      Sets the breakpoint at 0050:0123, if default CS = 0050
B S 100:20   Sets the breakpoint at 0100:0020
```

Directory - D [drive:\path\] [file mask]

The command displays an ordered list of the specified disk directory. The MSDOS version supports directory paths. If no drive or path is specified then the default drive and directory is used. Pagination occurs if the list is too long for a single screen display. Filename 'wild cards' are allowed.

D	Displays current directory
D C:\USER\	Displays all the files in the subdirectory C:\USER\
D C:\FRED*.HX	Displays all the files with .HX extension in C:\FRED\

Upload - U [drive:\path\] file [seg:] add1 add2

This command allows a block of memory to be saved to a disk file. The data is saved in the Extended Intel Hex format. If the file name extension is not supplied then it defaults to .HX (unless this is changed in EIOLP.MSG), this makes file name entry slightly more convenient. The MSDOS version supports directory paths.

One file is limited to a single segment of 64k, although a file of this size is not to be advised. Any number of files may be saved as they are all 'free standing'.

The command allows program development to 'cease' for the day and then return back to the same state when work continues next day, or next week.

U FRED 0 100
Saves contents of block 0050:0000 to 0050:0100 to file FRED.HX on the default drive.

U C:\USER\ALICE.HEX 070:200 400
Saves contents of block 0070:0200 to 0070:0400 to file ALICE.HEX on the drive C:, subdirectory USER.

As the file is uploaded, each line is displayed on the host screen. These lines are in the Extended Intel Hex format, described on page 26. If the file already exists you will be prompted as to whether or not you wish to overwrite it - take care!

Download - : [drive:\path\] filename

This command loads an Extended Intel Hex file from disk and puts it into the appropriate memory locations. The original file may have been generated using an assembler, compiler or the Upload command. If the file name only is supplied, then the extension .HX is assumed (unless this has been changed in EIOLP.MSG). The MSDOS version supports directory paths.

: FRED	downloads file FRED.HX from the default drive.
: C:\USER\ALICE.HEX	downloads file ALICE.HEX from drive C:, subdirectory USER.

As the file is downloaded, each line is displayed on the host screen. These lines are in the Extended Intel Hex format, described on page 26.

If any attempt is made to place the file in non-RAM locations, you will receive a suitable error message, and the download will cease.

Assemble - A [[segment:] address1]

This command allows you to key in 8086 assembly code commands, and these will be assembled and the bytes stored directly into memory. It is an ideal way of making changes to existing code.

If an invalid opcode is specified, then assembly will not occur and an error message will be issued.

There is a short delay as the bytes of code are sent to the board. This delay will vary, depending on the number of bytes to go (between one and six).

Any text following a semicolon will be ignored. This may be handy if printing key entries to note why something was done.

```
A          ; start assembly entry from the current address
A 100      ; start assembly entry from address 0100
A 60:100   ; start assembly entry from address 0060:0100
```

All numeric values, addresses and immediate data must be supplied as HEX numbers.

When specifying memory locations, you must use absolute addresses; generally the address should be specified with [ ] brackets. These will be converted to relative values, if necessary, by the assembly process.

The assembler will assemble the most suitable instruction for short, near or far jumps or calls. However, these may be explicitly over-ridden by the use of the following prefixes:

SHORT	jump within 128 bytes
NEAR	jump within segment
FAR	jump within segment:offset

For example:

```
0050:0100   JMP 150          ; 2-byte SHORT jump to 0050:0150
0050:0102   JMP NEAR 150     ; 3-byte NEAR jump to 0050:0150
0050:0105   JMP FAR 150      ; 5-byte FAR jump to 0050:0150
0050:010A   JMP 350          ; 3-byte NEAR jump to 0050:0350
0050:010D   JMP 0050:350     ; 5-byte FAR jump tp 0050:0350
```

The DB and DW directives may be used to specify bytes to be stored directly into memory.

Up to 6 bytes may be specified with the DB directive, and up to 3 words with the DW directive. Separate each number with a comma.

For example:

```
0050:0112   DB 1,2,12        ; will place  01h at location 0050:0112
                                           02h at location 0050:0113
                                           12h at location 0050:0114

0050:0115   DW 32,5678       ; will place  32h at location 0050:0115
                                           00h at location 0050:0116
                                           78h at location 0050:0117
                                           56h at location 0050:0118
```

To avoid confusion between an immediate operand and a memory operand, all memory operands should be enclosed in square brackets.

```
0050:0119   MOV AX,50          ; load AX with immediate value 0050h
0050:011C   MOV AX,[50]        ; load AX with the contents of
                               ; memory locations 0050h and 0051h
```

Sometimes, the size of the memory location required cannot be determined by the assembler from the information supplied. Under these circumstances the following prefixes must be used:

```
    BYTE PTR                   ; the word PTR is optional.
    WORD PTR
```

For example:

```
0050:011F   NEG BYTE PTR [300] ; negate the byte stored at 0050:0300
0050:0123   DEC WORD [SI]      ; decrement the word stored at the memory
                               ; locations pointed to by the contents of SI
```

Alternative forms of register indirect command entry may be used.

For example:

```
0050:0125   MOV  AX, [BX+SI]       ; these three are the same
0050:0127   MOV  AX, [BX] [SI]
0050:0129   MOV  AX, SI[BX]

0050:012B   MOV  [BX+SI+23] , AX   ; and so are these four
0050:012E   MOV  [BX+SI]23, AX
0050:0131   MOV  [BX] [SI] 23, AX
0050:0134   MOV  [23] BX SI, AX
```

String mnemonics must specify the string type, ie MOVSW for moving word strings, and MOVSB for byte strings.

Any of the four segment over-rides CS:, DS:, ES: and SS: may be used as a separate entry.

For example:

```
0050:0137   ES:
0050:0138   MOV   WORD PTR [BX+SI+1234] , 4A
```

Alternative valid opcodes for the same instruction may be used. The default is that used by the disassembler. The list is:

DEFAULT	OPTIONS	DEFAULT	OPTIONS
JB	JNAE or JC	JNL	JGE
JNB	JAE or JNC	JNLE	JG
JE	JZ	REPNZ	REPNE
JNE	JNZ	REP	REPE
JBE	JNA	LOOPNZ	LOOPNE
JNBE	JA	LOOPZ	LOOPE
JP	JPE	XLAT	XLATB
JNP	JPO	BYTE PTR	BYTE
JL	JNGE	WORD PTR	WORD
JLE	JNG		

Disassemble - Z [[V] [segment:] address1 [address2]]

This command allows the contents of memory to be reverse assembled to 8086 mnemonic codes. Relative addresses are translated to absolute memory locations.

If the V option is specified, then the ascii codes for the disassembled bytes are displayed alongside each line.

If the disassembler comes across a byte that cannot be properly disassembled, then all the bytes read so far for that instruction are displayed as DB or DW directives.

Bytes stored with DB directives will be disassembled if possible.

Examples:

```
Z              ; displays 10 lines of disassembled code starting at current address
Z 100          ; displays 10 lines of disassembled code starting at address 100
Z 100 120      ; displays the disassembled code between the addresses specified
Z V 100 130    ; displays the disassembled code between the addresses specified
                 with ASCII codes alongside.
```

The examples given with the assembler notes would be disassembled (provided the V option specified) as follows:

```
-Z V 100 138

0050:0100  EB 4E          JMP  short 0150                  ; . N
0050:0102  E9 4B 00       JMP  0150                        ; . K.
0050:0105  EA 50 01 50 00 JMP  far 0050:0150               ; . P.P.
0050:010A  E9 43 02       JMP  0350                        ; . C.
0050:010D  EA 50 03 50 00 JMP  far 0050:0350               ; . P.P.
0050:0112  01 02          ADD  [BP+SI], AX                 ; . .
0050:0114  12 32          ADC  DH, [BP+SI]                 ; . 2
0050:0116  00 78 56       ADD  [BP+SI+56], BH              ; . xV
0050:0119  B8 50 00       MOV  AX, 0050                    ; . P.
0050:011C  A1 50 00       MOV  AX, [0050]                  ; . P.
0050:011F  F6 1E 00 03    NEG  byte ptr [0300]             ; . . . .
0050:0123  FF 0C          DEC  word ptr [SI]               ; . .
0050:0125  8B 00          MOV  AX, [BX+SI]                 ; . .
0050:0127  8B 00          MOV  AX, [BX+SI]                 ; . .
0050:0129  8B 00          MOV  AX, [BX+SI]                 ; . .
0050:012B  89 40 23       MOV  [BX+SI+23], AX              ; . @#
0050:012E  89 40 23       MOV  [BX+SI+23], AX              ; . @#
0050:0131  89 40 23       MOV  [BX+SI+23], AX              ; . @#
0050:0134  89 40 23       MOV  [BX+SI+23], AX              ; . @#
0050:0137  26             ES:                              ;  &
0050:0138  C7 80 34 12 4A 00 MOV  word ptr [BX+SI+1234], 004A ; . .4.J.
```

Notice the bytes entered as DB and DW directives have been disassembled.

Hex Arithmetic - H number1 number2

This command allows you to enter 16-bit hex numbers, and it will work out the 16-bit sum and difference of these numbers. This is useful when calculating offsets and the like.

You must enter two valid hex numbers, otherwise the command syntax is displayed. As with all the commands, the leading zero's may be missed out.

Examples:

```
H 12 34       ; displays 0012 + 0034 = 0046     0012 - 0034 = FFDE
H F123 d03a   ; displays F123 + D03A = C15D     F123 - D03A = 20E9
```

Notice the results are truncated to 16-bits.

USER ACCESS TO MONITOR

The monitor code is written with the user in mind. It will always reside in the same place in ROM enabling users to program their code into the spare space and hook into the operating system.

Auto-start of user routines

When the board is 'powered up', it is 'initialised' and then location F000:E800 is read for the start address. Normally the monitor code is executed, alternatively, users may change this address to access their own code.

Access to Monitor Commands

The user may access all of the monitor commands by calling the monitor command handler. The absolute address of this handler may change but the start address is stored at 0000:0265. The user may access the command handler by extracting the start address from here. The command handler executes a NEAR return so the code segment must be set to F000 before calling the handler.

One word of warning: Any messages or results produced by the command handler are sent to the host computer via the serial line. If this is inoperative the system will 'hang'.

List of useful RAM locations

0000:0008 Holds Non-maskable Interrupt Vector

PIC, 8259A, prioritised vectors

0000:0080	8251A	Receiver
0000:0084	8251A	Transmitter
0000:0088	8255A	Even Port A
0000:008C	8255A	Even Port B
0000:0090	8255A	Odd Port A
0000:0094	8255A	Odd Port B
0000:0098	8253A	Timer 0 output
0000:009C	8253A	Timer 1 output

0000:0148 Last character received from serial port
0000:0149 Character to go to serial port
0000:014B ROM checksum
0000:014E Top address of RAM
0000:014E Baudrate word for serial communications speed

Word stored	Baud rate	Word stored	Baud rate
0010	9600	0080	1200
0020	4800	0100	0600
0040	2400	0200	0300

0000:0165 Start of input line from serial port
0000:01E5 Start of line to be sent to serial port
0000:0265 Command Handler start address

Interrupt Vector routines

There are a number of useful features that users may wish to implement into their own code.

The interrupt vectors 0 to 7 are assigned, as listed in the monitor source code, and 20 to 27 are assigned to the PIC (Programmable Interrupt Controller). The others between 8 to 19 and 28 to 63 are available for users to assign for their own usage.

Those that are normally assigned are available to the user if used carefully.

INT 2 May be assigned by the user to take control of the Non-Maskable Interrupt.

INT 5 May be executed by the user to return to the monitor program.

INT 6 May be executed to access the serial communications routines.
On entry AL is used to contain the function required:

Function 0 - Initialise the 8251A
sets up the 8251A and the 8253 channel 2 as the baud rate generator to the desired speed. See baudrate table below.

Function 1 - 8251A receiver input status
IF character waiting returns 02 in AL
ELSE returns 00 in AL.

Function 2 - 8251A receiver character input
IF character waiting returned (capitalised) in location 0000:0148
ELSE 00 returned in location 0000:0148.

Function 3 - receive a complete line from serial port
will not return until a $ or @ code received.
stores line of data bytes starting at 0000:0165
returns with length of line in CX

Function 4 - 8251A transmitter output status
IF ready to send a character returns 01 in AL
ELSE returns 00 in AL

Function 5 - 8251A transmitter character output
character at 0000:0149 sent to serial port
IF successful 0000:0149 loaded with 00
ELSE 0000:0149 remains intact

Function 6 - send a complete line to serial port
returns when line sent complete
the line to be sent must start at 0000:01E5 and terminate in a $ (this is extended to -$ for the host)

Function 7 - host reset command
sends *$ to the host to indicate the board has been RESET
the host re-tests the board and re-starts the monitor or user code, if the user start address has been used.

Function 8 - sets the serial port communications speed
uses the value stored at 0000:014E to set the baud rate.

INT 7 May be executed provided the Experiment book EPROM is fitted to access the model answers. You may find this useful if demonstrating the experiment solutions.

INT 32 to INT 39
May be assigned by the user to take control of the 8259A Programmable Interrupt Controller. These interrupts are extremely powerful and should be used with care.

You will also need to access the 8259A registers at Ports 10 and 12 to set up the 8259A mode.

MONITOR PSEUDO CODE LISTING

This listing gives an overview of the monitor program operation and flow. Line numbers have been included to allow a link to the actual code to be made.

MAIN ROUTINE

```
1544  COLD ENTRY    (RESET button pressed, power up, INT 0 or INT 4)
 210      initialise monitor and user registers (default segment = 0050)
 239  WARM ENTRY    (X command)
          ensure monitor registers intact
 246      initialise Interrupt Vectors          (NMI suitable for fault book, (480))
          initialise default user storage & PIC (disable interrupts)
          initialise serial communications to 9600 baud
 300      REPEAT
              send * to host to set up communications link
          UNTIL ! received from host
 306      send null line to host to indicate communications successful (893)
          re-initialise NMI to default
 308      check user ROM routine start address at F000:E000
          IF address at F000:E800 is NOT the monitor default, execute user routine
          .REPEAT
 314          execute command handler (921)
          UNTIL forever
```

COMMAND HANDLER

All user commands from host processed here. Command tokens range from @ to L, these are used to access vector table of commands. The suffix is in the range 0 to 9. If the range for the individual command is exceeded the routine aborts and returns 'invalid suffix' error.

```
 921      get command line from host (838)
 940      IF invalid command token return with 'invalid command' error (886)
          ELSE extract command from table (starts at 795)
 945      CASE command letter of
          @: ESCAPE - usually used to establish communications handshaking
 950            return with null line (893)
          A: RESET - re-initialise the system, not user registers
 954            perform warm start (239)
          B: TEST - performs testing, baud rate setting and console tasks
 959        CASE subscript of
 972        0: RAM TEST - non-destructive, identifies RAM size
                   perform actual test (591)
 979               IF Ramtop > 0  return with 0000 and Ramtop address (894)
 977               ELSE           return with 'No RAM fitted' error (887)
 986        1: ROM TEST - checksum of monitor code F000:E802 to F000:FFFF
                   perform actual test (577)
                   return with ROM start address, and checksum (894)
1023        2: PRODUCTION TEST - not installed in most user ROMs
                   return with 'not available' error (883)
1008        3: COMMUNICATIONS SPEED
                   extract speed specified (0=9600. . . 6=300)
                   IF speed valid
1011                   calculate setting for 8253, set up the new rate (666)
                       return with null line (893)
1020               ELSE return with 'not available' error (883)
```

```
        C: REGISTER - allows access to user registers
1030        CASE subscript of
1039        0: UPLOAD ALL REGISTERS
                   return with all 14 user register contents (875)
1045        1: UPLOAD SPECIFIC REGISTER
                   extract register number
1064               IF valid   return with register number and contents (894)
1072               ELSE       return with 'invalid command' error (886)
1045        2: UPDATE SPECIFIC REGISTER
                   extract register number
1059               IF valid   put new data in register, return with null line (893)
1072               ELSE       return with 'invalid command' to host (886)
        D: MEMORY - allows access to memory locations
1077        CASE subscript of
1096        0..1: UPLOAD ONE MEMORY LOCATION
                       IF 0 use default CS ELSE use segment supplied
1106                   return with segment, address and byte (859)
1110        2:    UPDATE ONE MEMORY LOCATION
                       store new data.
1099                   IF stored ok  return with null line (893)
1125                   ELSE          return with 'write fail' error (901)
1128        3:    INCREMENT CURRENT ADDRESS
                       increment address, return with new segment & address (1099)
1128        4:    DECREMENT CURRENT ADDRESS
                       decrement address, return with new segment & address (1099)
1139        5..6: UPLOAD A BLOCK OF DATA
                       IF 0 use default CS ELSE use segment supplied
1142                   force data to be one line minimum
1144                   REPEAT
                         send segment, address and line of data (859)
1151                     increment to next line
1158                     IF finished    return with null line (893)
1161                     ELSE           get next command from host (838)
                       UNTIL next command <> 'f'
        E: FILL - allows a block of memory to be initialised
1178        IF subscript 0 use default CS
            IF subscript 1 use segment supplied
1184        calculate number of bytes to do
            REPEAT
1201            IF byte    store byte and step one address
1193            ELSE       store word store and step two addresses
1212            IF NOT stored ok return with 'write fail' error (901)
            UNTIL all bytes stored
            return null line (894)
        F: PORT INPUT - allows port addresses to be read directly
1230        IF subscript 0 return with byte value read (894)
1235        IF subscript 1 return with word value read (894)
        G: PORT OUTPUT - allow data to be sent to ports directly
1253        IF subscript 0 send byte, return with null line (893)
1257        IF subscript 1 send word, return with null line (893)
        H: GO - allows users to execute their own programs
1283        IF subscript 0 use default CS and IP
1274        IF subscript 1 use supplied CS and default IP
1276        IF subscript 2 use supplied CS and IP
            EXECUTE USER CODE
 444        clear exit flag and check single stepping traps
            restore user registers from storage
 474        execute user routine
```

```
        I:  BREAKPOINT - allows user to set break points in code for debugging
1288            CASE subscript of
                0:      DISPLAY BREAKPOINTS
1301                        IF one set      return with number and address (893)
1318                        ELSE            return with 'not set' error (894)
                1..2:   SET BREAKPOINT
1338                        IF 1 use default CS ELSE use supplied segment
                            Save current opcode, replace with CCh (INT 3)
1352                        IF save ok      set flag and return with null line (893)
1359                        ELSE            return with 'cannot set' error (902)
                3:      REMOVE BREAKPOINT
1327                        make sure breakpoint removed (1364)
        J:  SINGLE STEP - allows user to step through code one instruction at a time
1381            save suffix for return
1401            IF subscript 0 use default CS and IP
1394            IF subscript 1 use supplied CS and default IP
1396            IF subscript 2 use supplied CS and IP
                set single step flag and execute user code (444)
        K:  UPLOAD - allows user to save a block of memory to disk
1419            IF subscript 0 use default CS
1421            IF subscript 1 use segment specified
1422            REPEAT
                    Send segment, address and line of data (859)
                    increment to next line
                    get next command from host (838)
                UNTIL next command <> '!'
1441            return with null line (893)
        L:  DOWNLOAD - allows a disk file to be dumped into memory
1454            CASE record type
1495            0:  DATA RECORD
                    do checksum (1480) IF ok store data bytes in memory
1519            1:  END RECORD
                    do checksum (1480) only
1522            2:  SEGMENT ADDRESS RECORD
                    do checksum (1480) IF ok update user CS
1528            3:  START ADDRESS RECORD
                    do checksum (1480) IF ok update user CS and IP
                IF checksum zero    return null line (893)
                ELSE                return 'checksum' error (894)
```

NON-MASKABLE INTERRUPT ROUTINES

There are three NMI routines available. Two use the NMI button BEFORE serial communication is established. These are used by the fault finding book.

```
 480  DEFAULT NMI (does very little)
          restore DS, set exit flag to 05, and return
 192  FIRST NMI (before serial communications started)
          initialise second NMI
          loop forever doing nothing
 202  SECOND NMI
          initialise default NMI
          loop forever writing to 0000:0400
```

RETURN TO MONITOR FROM USER CODE

There are 4 mechanisms for the user getting to this point.

```
319  IF from INT 1 (Single Step exit) restore DS and set flag to 01
349     recover user registers and store for later reference
380     ensure monitor vectors intact
417     IF registers required   return with address and opcode byte and registers
        ELSE                    return with address and opcode byte only
328  IF from INT 3 (Breakpoint exit) restore DS
349     recover user registers and store for later reference
380     ensure monitor vectors intact
        IF official breakpoint set flag to 02
402        restore opcode to breakpoint position
           return with 'official breakpoint' message (902)
        ELSE unofficial breakpoint set flag to 03
           return with 'unofficial breakpoint' message (902)
344  IF from INT 5 (User break exit) restore DS and set flag to 04
349     recover user registers and store for later reference
380     ensure monitor vectors intact
398     return with 'user exit' message (902)
```

SERIAL COMMUNICATIONS ROUTINES

These routines are available to the user through INT 6, with the operation in AL. This value is used to vector to the routine start address from a table (start at 611). All subroutines subsequently return with an IRET instruction. AX and CX are used to return values where appropriate.

```
     CASE operation
653  0: INITIALISE UART
           Put 8251A into command mode and set speed.
676  1: INPUT PORT STATUS
           IF character waiting     return 02 in AL ELSE return 00
683  2: GET CHARACTER FROM SERIAL PORT
           IF character waiting     return with 0000:0148 & AL = character
           ELSE                     return with 0000:0148 & AL = 00
701  3: GET A LINE FROM SERIAL PORT
           REPEAT
              store character received in line starting at 0000:0165
           UNTIL $ received
729  4: OUTPUT PORT STATUS
           IF 8251A ready to send returns 01 in AL ELSE returns 00
736  5: OUTPUT A CHARACTER TO SERIAL PORT
           IF 8251A ready,    send value at 0000:0149, set [0000:0149] to 00
           ELSE               return with no change
757  6: OUTPUT A LINE TO SERIAL PORT
           REPEAT
              send characters from line starting at 0000:01E5
           UNTIL $ reached in line
772  7: SEND RESET TO HOST
           send * to host using (748)
666  8: SET PIT BAUD RATE SPEED
           Use speed setting at 0000:014E to set up 8253
```

MONITOR SOURCE CODE LISTING

```
1                                       LIST ON
2                                       OUTPUT 2500AD
3                                       COMMENT %
4                                MON.ASM version 2.0  1st August 1990
5
6                                Written by John Guy for:
7
8                                Flight Electronics Ltd.,
9                                Flight House, Ascupart Street, Southampton,Hampshire, SO1 1LU
10
11                               This code may be copied, in its entirety, into EPROM for use by
12                               users wishing to interface their routines with this FLIGHT86
13                               Controller Board MONITOR software.
14
15                               This code must not be copied for gain (financial or otherwise)
16                               or for use with any other hardware without the express written
17                               consent of Flight Electronics Ltd..
18
19                               A Pseudo Code Listing is elsewhere in this manual.
20                                      %
21                                      ASSUME CS:CODE; DS:DATA; ES:DATA; SS:DATA
22
23                               ; ----- EQUATES & DEFINITIONS ---------------------------:
24
25                                                              ; Port address definitions
26                 0000:0000     PPIAA:  EQU     00h     ; U10 8255A Port A
27                 0000:0002     PPIAB:  EQU     02h     ;           Port B
28                 0000:0004     PPIAC:  EQU     04h     ;           Port C
29                 0000:0006     PPIAK:  EQU     06h     ;           Control Port
30                 0000:0001     PPIBA:  EQU     01h     ; U9  8255A Port A
31                 0000:0003     PPIBB:  EQU     03h     ;           Port B
32                 0000:0005     PPIBC:  EQU     05h     ;           Port C
33                 0000:0007     PPIBK:  EQU     07h     ;           Control Port
34                 0000:0008     CTC0:   EQU     08h     ; U8  8253  Count 0
35                 0000:000A     CTC1:   EQU     0Ah     ;           Count 1
36                 0000:000C     CTC2:   EQU     0Ch     ;           Count 2
37                 0000:000E     CTCK:   EQU     0Eh     ;           Mode Word
38                 0000:0010     PIC1:   EQU     10h     ; U11 8259A Reg 1
39                 0000:0012     PIC2:   EQU     12h     ;           Reg 2
40                 0000:0018     UARTD:  EQU     18h     ; U7  8251A Data
41                 0000:001A     UARTS:  EQU     1Ah     ;           Status & Control
42   0000:0000
43   0000:0000                             DATA
44   0000:0000                             ABSOLUTE
45   0000:0000                             ORIGIN 0000:0000        ; System Base address
46
47                 0000:0000     RAMSEG: EQU     0000h   ; start segment of monitor RAM
48                 0000:F000     ROMSEG: EQU     F000h   ; start address of this monitor EPROM
49                 0000:0050     USR0:   EQU     0050h   ; default user RAM segment
50
51                               ; INT0 to INT63 are available to the monitor and user programs.
52                               ; Those marked with * are essential to proper monitor operation.
53                               ; They are reset every time the monitor restarts.
54
55   0000:0000                   INT0:   DS      4       ; DIVIDE BY ZERO
56                                                       ; Sets ZDIF flag and returns
57   0000:0004                   INT1:   DS      4       ; SINGLE STEP *
58                                                       ; Used by official monitor single step
59                                                       ; routine. If used unofficially treated
60                                                       ; the same as an INT5.
```

```
0000:0008                INT2:   DS      4       ; NMI
                                                 ; Normally sets NMIF flag and returns.
                                                 ; Special case when COLD start or RESET
                                                 ; If serial comms not established then
                                                 ; may be used to call 2 standard loops
                                                 ; for our fault finding books.
0000:000C                INT3:   DS      4       ; BREAKPOINT *
                                                 ; Used by official monitor breakpoint
                                                 ; routine. If used unofficially treated
                                                 ; the same as an INT5.
0000:0010                INT4:   DS      4       ; OVERFLOW *
                                                 ; Sets OVRF flag and returns.
0000:0014                INT5:   DS      4       ; USER EXIT *
                                                 ; Official method of user terminating
                                                 ; programs and returning to monitor.
0000:0018                INT6:   DS      4       ; SERIAL *
                                                 ; Serial comms routines readily
                                                 ; available to the user.
0000:001C                INT7:   DS      4       ; EXPERIMENTS
                                                 ; access to experiment book answers.
0000:0020                        DS      96      ; Interrupt INT8 to INT31 may be safely
                                                 ; assigned by user programs here.

                         ; The following interrupt vectors are assigned to the 8259 PIC.
                         ; They are all available to user programs.

                                                 ; 8259 VECTOR MAP
0000:0080                INT32:  DS      4       ; IRQ0 8251A Rx
0000:0084                INT33:  DS      4       ; IRQ1 8251A Tx
0000:0088                INT34:  DS      4       ; IRQ2 8255A Even port A
0000:008C                INT35:  DS      4       ; IRQ3 8255A Even port B
0000:0090                INT36:  DS      4       ; IRQ4 8255A Odd port A
0000:0094                INT37:  DS      4       ; IRQ5 8255A Odd port B
0000:0098                INT38:  DS      4       ; IRQ6 8253  Timer 0
0000:009C                INT39:  DS      4       ; INT7 8253  Timer 1

0000:00A0                        DS      96      ; Interrupt INT40 to INT63 may be safely
                                                 ; assigned by user programs here.

                         ; From here to 0000:0400h is reserved for MONITOR usage
                         ; These locations will not change for future enhancements

0000:0100                        DS      40      ; reserved
0000:0128                USRAX:  DS      2       ; AX register  USER REGISTERS
0000:012A                USRBX:  DS      2       ; BX register
0000:012C                USRCX:  DS      2       ; CX register
0000:012E                USRDX:  DS      2       ; DX register
0000:0130                USRSP:  DS      2       ; SP register
0000:0132                USRBP:  DS      2       ; BP register
0000:0134                USRSI:  DS      2       ; SI register
0000:0136                USRDI:  DS      2       ; DI register
0000:0138                USRDS:  DS      2       ; DS register
0000:013A                USRES:  DS      2       ; ES register
0000:013C                USRSS:  DS      2       ; SS register
0000:013E                USRCS:  DS      2       ; CS register
0000:0140                USRIP:  DS      2       ; IP register
0000:0142                USRFG:  DS      2       ; FLAG register
0000:0144                CURSEG: DS      2       ; current user segment
0000:0146                CURADR: DS      2       ; current user address
0000:0148                RXCHR:  DS      1       ; last character received
```

```
0000:0149                       TXCHR:  DS      1       ; next character to go = 00 when sent
0000:014A                       ESCCHR: DS      1       ; 00 = normal,
                                                        ; 40 = @ escape code received

0000:014B                       ROMSUM: DS      1       ; actual ROM checksum
0000:014C                       RAMLST: DS      2       ; actual RAM last address
0000:014E                       BAUDRT: DS      2       ; baud rate settings for 8253
                                                        ; 0010 for 9600 baud, 0020 for 4800 etc

0000:0150                       CMDLTR: DS      1       ; command letter
0000:0151                       CMDSFX: DS      1       ; command suffix
0000:0152                       CMDPR1: DS      2       ; command line parameters
0000:0154                       CMDPR2: DS      2
0000:0156                       CMDPR3: DS      2
0000:0158                       CMDPR4: DS      2
0000:015A                       BPRECH: DS      1       ; exited at break point
                                                        ; flag single step for next GO
0000:015B                       BPFLAG: DS      1       ; breakpoint flag
0000:015C                       BPCODE: DS      1       ; byte removed from user program
0000:015D                       BPSEG:  DS      2       ; user set breakpoint segment
0000:015F                       BPADR:  DS      2       ; user set breakpoint address
0000:0161                       STDUMY: DS      1       ; single step flag to allow re-insertion
                                                        ; of breakpoint.
0000:0162                       STFLAG: DS      1       ; single step flag
0000:0163                       STSFX:  DS      1       ; single step suffix required
0000:0164                       EXTFLG: DS      1       ; 01 = single step              EXIT CODES
                                                        ; 02 = official break point
                                                        ; 03 = unofficial breakpoint
                                                        ; 04 = user via  INT 5
                                                        ; 05 = NMI interrupt occured
0000:0165                       INLNE:  DS      128     ; input command line
0000:01E5                       OUTLNE: DS      128     ; output returned line
0000:0265                       GETCOM: DS      2       ; COMND start address

0000:0400                               ORIGIN  0000:0400h
            0000:0400           TOPSTK: EQU     $       ; top of monitor stack
0000:0400                       DUMMY:  DS      1       ; used only by faultfinding routines

0000:0500                               ORIGIN  0000:0500h
            0000:0500           USRSTK: EQU     $       ; top of default user stack

0000:0000                               CODE
0000:0000                               RELATIVE

                                ; ----- USER SPACE in EPROM ----------------------------------
                                ;       ORIGIN  F000:0000h      ; for 27256
                                ;       ORIGIN  F000:8000h      ; for 27128
                                ;       ORIGIN  F000:C000h      ; for 2764  EPROM (all to DFFFh)

                                ; ----- EXPERIMENT BOOK SPACE --------------------------------
F000:E000                               ORIGIN  F000:E000h
                                ; Model answer routines will reside from E000h to E7FFh.
                                ; New EPROMs, and listing will be available with this book

F000:E000   B0 00               DOEXP:  MOV     AL, 0           ; experiment number
F000:E002   CD 07                       INT     7               ; entry point for exeriments
F000:E004   E9 89 09                    JMP     MLOOP           ; return to monitor loop

F000:E007   CF                  EXPER:  IRET                    ; in case called accidentally

                                ; ----- MONITOR CODE SPACE -----------------------------------
```

```
181  F000:E800                              ORIGIN  F000:E800h
182                                 ; User may alter this address to point to a location in ROM where
183                                 ; their program resides. This means calls may then be made to
184                                 ; the monitor commands leaving the user in control of the board.
185
186  F000:E800  90E9                USER:   DW      MLOOP           ; user boot address
187  F000:E802  4A 6F 68 6E 20 47   ME:     DB      'John Guy'      ; author's name
     F000:E808  75 79
188
189                                 ;.Gives suitable loop conditions for the Fault Finding Book
190                                 ; If NMI pressed before serial communications established
191
192  F000:E80A  C7 06 08 00 27 E8   NMIONE: MOV     WORD PTR INT2, OFFSET CS:NMITWO
193  F000:E810  90                  $1:     NOP                     ; E810h address required for
194  F000:E811  EB FD                       JMP     SHORT $1        ; faultfinding book
195
196  F000:E813  20 46 4C 49 47 48   UNIT:   DB      ' FLIGHT86 Board     '
     F000:E819  54 38 36 20 42 6F
     F000:E81F  61 72 64 20 20 20
     F000:E825  20 20
197
198                                 ; Gives suitable loop conditions for the Fault Finding Book
199                                 ; If NMI pressed again before serial communications established
200                                 ; only escape from here is by pressing RESET
201
202  F000:E827  C7 06 08 00 21 EB   NMITWO: MOV     WORD PTR INT2, OFFSET CS:NMIDEF
203  F000:E82D  B8 55 55                    MOV     AX, 5555h
204  F000:E830  A3 00 04            $2:     MOV     [DUMMY], AX     ; E830h address required for
205  F000:E833  90                          NOP                     ; faultfinding book
206  F000:E834  EB FA                       JMP     SHORT $2
207
208                                 ; ----- ACTUAL CODE START ------------------------------------
209
210  F000:E836  B8 00 00            COLD:   MOV     AX, RAMSEG      ; initialise segment pointers
211  F000:E839  8E D8                       MOV     DS, AX          ; to bottom of map
212  F000:E83B  8E C0                       MOV     ES, AX          ; access to base addresses
213  F000:E83D  8E D0                       MOV     SS, AX          ; and stack
214  F000:E83F  BC 00 04                    MOV     SP, TOPSTK
215  F000:E842  BF 28 01                    MOV     DI, USRAX       ; initialise user registers
216  F000:E845  B8 00 00                    MOV     AX, RAMSEG      ; point to first user register
217  F000:E848  AB                          STOSW                   ; AX register
218  F000:E849  AB                          STOSW                   ; BX register
219  F000:E84A  AB                          STOSW                   ; CX register
220  F000:E84B  AB                          STOSW                   ; DX register
221  F000:E84C  B8 00 05                    MOV     AX, USRSTK      ; SP works down from USER BASE
222  F000:E84F  AB                          STOSW                   ; gives a usable user stack
223  F000:E850  B8 00 00                    MOV     AX, RAMSEG
224  F000:E853  AB                          STOSW                   ; BP register
225  F000:E854  AB                          STOSW                   ; SI register
226  F000:E855  AB                          STOSW                   ; DI register
227  F000:E856  B8 50 00                    MOV     AX, USR0
228  F000:E859  AB                          STOSW                   ; DS register
229  F000:E85A  AB                          STOSW                   ; ES register
230  F000:E85B  B8 00 00                    MOV     AX, RAMSEG
231  F000:E85E  AB                          STOSW                   ; SS register
232  F000:E85F  B8 50 00                    MOV     AX, USR0
233  F000:E862  AB                          STOSW                   ; CS register
234  F000:E863  B8 00 00                    MOV     AX, RAMSEG
235  F000:E866  AB                          STOSW                   ; IP register
236  F000:E867  B8 00 00                    MOV     AX, 0000
```

```
237  F000:E86A  AB                          STOSW                   ; FLAG register
238
239  F000:E86B  FA                  WARM:   CLI                     ; belt and braces
240  F000:E86C  B8 00 00                    MOV     AX, RAMSEG      ; initialise segment pointers
241  F000:E86F  8E D8                       MOV     DS, AX          ; to bottom of map
242  F000:E871  8E C0                       MOV     ES, AX          ; access to base addresses
243  F000:E873  8E D0                       MOV     SS, AX          ; and stack
244  F000:E875  BC 00 04                    MOV     SP, TOPSTK
245                                                                 ;  Set up major INT vectors
246  F000:E878  C7 06 02 00 00 F0           MOV     WORD PTR INT0+2, ROMSEG
247  F000:E87E  C7 06 00 00 F0 FF           MOV     WORD PTR INT0, OFFSET CS:RESET  ; reset board
248  F000:E884  C7 06 06 00 00 F0           MOV     WORD PTR INT1+2, ROMSEG
249  F000:E88A  C7 06 04 00 96 E9           MOV     WORD PTR INT1, OFFSET CS:SSTEP
250  F000:E890  C7 06 0A 00 00 F0           MOV     WORD PTR INT2+2, ROMSEG
251  F000:E896  C7 06 08 00 0A E8           MOV     WORD PTR INT2, OFFSET CS:NMIONE ; for fault book
252  F000:E89C  C7 06 0E 00 00 F0           MOV     WORD PTR INT3+2, ROMSEG
253  F000:E8A2  C7 06 0C 00 A5 E9           MOV     WORD PTR INT3, OFFSET CS:BPOINT
254  F000:E8A8  C7 06 12 00 00 F0           MOV     WORD PTR INT4+2, ROMSEG
255  F000:E8AE  C7 06 10 00 F0 FF           MOV     WORD PTR INT4, OFFSET CS:RESET  ; reset board
256  F000:E8B4  C7 06 16 00 00 F0           MOV     WORD PTR INT5+2, ROMSEG
257  F000:E8BA  C7 06 14 00 BD E9           MOV     WORD PTR INT5, OFFSET CS:UEXIT
258  F000:E8C0  C7 06 1A 00 00 F0           MOV     WORD PTR INT6+2, ROMSEG
259  F000:E8C6  C7 06 18 00 E9 EB           MOV     WORD PTR INT6, OFFSET CS:SERIAL
260  F000:E8CC  C7 06 1E 00 00 F0           MOV     WORD PTR INT7+2, ROMSEG
261  F000:E8D2  C7 06 1C 00 07 E0           MOV     WORD PTR INT7, OFFSET CS:EXPER
262                                                                 ; set up 8259 isr routines
263  F000:E8D8  C7 06 82 00 00 F0           MOV     WORD PTR INT32+2, ROMSEG
264  F000:E8DE  C7 06 80 00 30 EB           MOV     WORD PTR INT32, OFFSET CS:IRQ0
265  F000:E8E4  C7 06 86 00 00 F0           MOV     WORD PTR INT33+2, ROMSEG
266  F000:E8EA  C7 06 84 00 31 EB           MOV     WORD PTR INT33, OFFSET CS:IRQ1
267  F000:E8F0  C7 06 8A 00 00 F0           MOV     WORD PTR INT34+2, ROMSEG
268  F000:E8F6  C7 06 88 00 32 EB           MOV     WORD PTR INT34, OFFSET CS:IRQ2
269  F000:E8FC  C7 06 8E 00 00 F0           MOV     WORD PTR INT35+2, ROMSEG
270  F000:E902  C7 06 8C 00 33 EB           MOV     WORD PTR INT35, OFFSET CS:IRQ3
271  F000:E908  C7 06 92 00 00 F0           MOV     WORD PTR INT36+2, ROMSEG
272  F000:E90E  C7 06 90 00 34 EB           MOV     WORD PTR INT36, OFFSET CS:IRQ4
273  F000:E914  C7 06 96 00 00 F0           MOV     WORD PTR INT37+2, ROMSEG
274  F000:E91A  C7 06 94 00 35 EB           MOV     WORD PTR INT37, OFFSET CS:IRQ5
275  F000:E920  C7 06 9A 00 00 F0           MOV     WORD PTR INT38+2, ROMSEG
276  F000:E926  C7 06 98 00 36 EB           MOV     WORD PTR INT38, OFFSET CS:IRQ6
277  F000:E92C  C7 06 9E 00 00 F0           MOV     WORD PTR INT39+2, ROMSEG
278  F000:E932  C7 06 9C 00 37 EB           MOV     WORD PTR INT39, OFFSET CS:IRQ7
279  F000:E938  E8 80 02                    CALL    RAMTST                  ; ensures a suitable
280                                                                         ; delay for host
281  F000:E93B  C7 06 44 01 50 00           MOV     WORD PTR CURSEG, USR0   ; initialise CS
282  F000:E941  C7 06 46 01 00 00           MOV     WORD PTR CURADR, 0000   ; and IP
283  F000:E947  C6 06 62 01 00              MOV     BYTE PTR STFLAG, 00     ; clear single step flag
284  F000:E94C  C6 06 58 01 00              MOV     BYTE PTR BPFLAG, 00     ; clear break point flag
285  F000:E951  C7 06 65 02 BD ED           MOV     WORD PTR GETCOM, OFFSET COMND   ; commands
286  F000:E957  B0 17                       MOV     AL, 17h         ; '00010111' initialise PIC
287  F000:E959  BA 10 00                    MOV     DX, PIC1        ; point to ICW1
288  F000:E95C  EE                          OUT     DX, AL          ; send ICW1
289  F000:E95D  B8 20 00                    MOV     AX, INT32/4     ; interrupt type in AL
290  F000:E960  0B C0                       OR      AX, AX          ; dummy
291  F000:E962  BA 12 00                    MOV     DX, PIC2        ; point to ICW2
292  F000:E965  EE                          OUT     DX, AL          ; send ICW2
293  F000:E966  B0 01                       MOV     AL, 01
294  F000:E968  EE                          OUT     DX, AL          ; send ICW4
295  F000:E969  B0 FF                       MOV     AL, FFh
296  F000:E96B  EE                          OUT     DX, AL          ; mask out 8259
```

```
297   F000:E96C   C7 06 4E 01 10 00          MOV     WORD PTR BAUDRT, 0010h  ; 9600 baud rate
298   F000:E972   B0 00                      MOV     AL, 0                   ; initialise the UART
299   F000:E974   CD 06                      INT     6
300   F000:E976   B0 07               $1:    MOV     AL, 7           ; send reset * to host
301   F000:E978   CD 06                      INT     6
302   F000:E97A   E8 98 03                   CALL    GETCMD          ; wait for !$ command
303   F000:E97D   3C 21                      CMP     AL, '!'
304   F000:E97F   75 F5                      JNE     $1
305                                                                  ; comms successful
306   F000:E981   E8 00 04                   CALL    NULLNE          ; send -$ back to host
307   F000:E984   C7 06 08 00 21 E8          MOV     WORD PTR INT2, OFFSET CS:NMIDEF ; normal NMI
308   F000:E98A   2E A1 00 E8                MOV     AX, USER        ; get user routine start address
309   F000:E98E   50                         PUSH    AX              ; will become the IP
310   F000:E98F   C3                         RET                     ; start execution (current CS)
311
312                                 ; ----- DEFAULT MONITOR LOOP ---------------------------------
313   F000:E990
314   F000:E990   E8 2A 04            MLOOP: CALL    COMND           ; loop waiting for host commands
315   F000:E993   E9 FA FF                   JMP     MLOOP
316
317                                 ; ----- SINGLE STEP EXIT -------------------------------------
318
319   F000:E996   1E                  SSTEP: PUSH    DS              ; save user DS on user stack
320   F000:E997   50                         PUSH    AX              ; save user AX on user stack
321   F000:E998   B8 00 00                   MOV     AX, RAMSEG
322   F000:E99B   8E D8                      MOV     DS, AX          ; data segment accessible again
323   F000:E99D   C6 06 64 01 01             MOV     BYTE PTR [EXTFLG], 01   ; set exit flag code
324   F000:E9A2   E9 24 00                   JMP     TOMON
325
326                                 ; ----- BREAKPOINT EXIT --------------------------------------
327
328   F000:E9A5   1E                  BPOINT: PUSH   DS              ; save user DS on user stack
329   F000:E9A6   50                         PUSH    AX              ; save user AX on user stack
330   F000:E9A7   B8 00 00                   MOV     AX, RAMSEG
331   F000:E9AA   8E D8                      MOV     DS, AX          ; data segment accessible again
332   F000:E9AC   B0 02                      MOV     AL, 02                  ; was it official
333   F000:E9AE   80 3E 5B 01 00             CMP     BYTE PTR [BPFLAG], 00   ; breakpoint set ?
334   F000:E9B3   75 02                      JNZ     $1
335   F000:E9B5   FE C0                      INC     AL              ; 03 = unofficial breakpoint
336   F000:E9B7   A2 64 01            $1:    MOV     BYTE PTR [EXTFLG], AL   ; set exit flag code
337   F000:E9BA   E9 0C 00                   JMP     TOMON           ; complete return to monitor
338
339                                 ; ----- USER EXIT --------------------------------------------
340
341                                 ; Graceful exit from user code and return to monitor
342                                 ; User stack will show flags, IP, CS  and DS on exit
343
344   F000:E9BD   1E                  UEXIT: PUSH    DS              ; save user DS on user stack
345   F000:E9BE   50                         PUSH    AX              ; save user AX on user stack
346   F000:E9BF   B8 00 00                   MOV     AX, RAMSEG
347   F000:E9C2   8E D8                      MOV     DS, AX          ; data segment accessible again
348   F000:E9C4   C6 06 64 01 04             MOV     BYTE PTR [EXTFLG], 04   ; set exit flag code
349   F000:E9C9   58                  TOMON: POP     AX              ; extract user AX from stack
350   F000:E9CA   A3 28 01                   MOV     [USRAX], AX     ; salt away user AX
351   F000:E9CD   58                         POP     AX              ; extract user DS from stack
352   F000:E9CE   A3 38 01                   MOV     [USRDS], AX     ; salt away user DS
353                                                                  ; and now do rest of registers
354   F000:E9D1   89 1E 2A 01                MOV     [USRBX], BX     ; BX
355   F000:E9D5   89 0E 2C 01                MOV     [USRCX], CX     ; CX
356   F000:E9D9   89 16 2E 01                MOV     [USRDX], DX     ; DX
```

```
357  F000:E9DD  89 2E 32 01              MOV     [USRBP], BP     ; BP
358  F000:E9E1  89 36 34 01              MOV     [USRSI], SI     ; SI
359  F000:E9E5  89 3E 36 01              MOV     [USRDI], DI     ; DI
360  F000:E9E9  8C C0                    MOV     AX, ES
361  F000:E9EB  A3 3A 01                 MOV     [USRES], AX     ; ES
362  F000:E9EE  58                       POP     AX              ; remove stored values off stack
363  F000:E9EF  A3 40 01                 MOV     [USRIP], AX     ; IP
364  F000:E9F2  A3 46 01                 MOV     [CURADR], AX    ; update current address
365  F000:E9F5  A3 54 01                 MOV     [CMDPR2], AX    ; ready for exit message
366  F000:E9F8  58                       POP     AX
367  F000:E9F9  A3 3E 01                 MOV     [USRCS], AX     ; CS
368  F000:E9FC  A3 44 01                 MOV     [CURSEG], AX    ; update current segment
369  F000:E9FF  A3 52 01                 MOV     [CMDPR1], AX    ; ready for exit message
370  F000:EA02  58                       POP     AX
371  F000:EA03  A3 42 01                 MOV     [USRFG], AX     ; Flags
372  F000:EA06  8C D0                    MOV     AX, SS          ; can now save user stack
373                                                              ; conditions PRIOR to exit.
374  F000:EA08  A3 3C 01                 MOV     [USRSS], AX     ; SS
375  F000:EA0B  89 26 30 01              MOV     [USRSP], SP     ; SP
376  F000:EA0F  E9 C6 08                 JMP     V13A            ; Version 1.3 patch
377  F000:EA12  90                V13AR: NOP                     ; and return here
378  F000:EA13  BC 00 04                 MOV     SP, TOPSTK
379                                                              ; ensure vectors intact
380  F000:EA16  C7 06 06 00 00 F0        MOV     WORD PTR INT1+2, ROMSEG
381  F000:EA1C  C7 06 04 00 96 E9        MOV     WORD PTR INT1, OFFSET CS:SSTEP
382  F000:EA22  C7 06 0E 00 00 F0        MOV     WORD PTR INT3+2, ROMSEG
383  F000:EA28  C7 06 0C 00 A5 E9        MOV     WORD PTR INT3, OFFSET CS:BPOINT
384  F000:EA2E  C7 06 16 00 00 F0        MOV     WORD PTR INT5+2, ROMSEG
385  F000:EA34  C7 06 14 00 BD E9        MOV     WORD PTR INT5, OFFSET CS:UEXIT
386  F000:EA3A  C7 06 1A 00 00 F0        MOV     WORD PTR INT6+2, ROMSEG
387  F000:EA40  C7 06 18 00 E9 EB        MOV     WORD PTR INT6, OFFSET CS:SERIAL
388  F000:EA46  C7 06 1E 00 00 F0        MOV     WORD PTR INT7+2, ROMSEG
389  F000:EA4C  C7 06 1C 00 07 E0        MOV     WORD PTR INT7, OFFSET CS:EXPER
390  F000:EA52  B0 00                    MOV     AL, 0           ; re-initialise the UART
391  F000:EA54  CD 06                    INT     6
392  F000:EA56  80 3E 64 01 01           CMP     BYTE PTR [EXTFLG], 1    ; is this return from
393  F000:EA5B  74 38                    JE      SSEXIT                  ; a single step ?
394  F000:EA5D  80 3E 64 01 02           CMP     BYTE PTR [EXTFLG], 2    ; official breakpoint?
395  F000:EA62  74 0E                    JE      OFBPEX
396  F000:EA64  80 3E 64 01 03           CMP     BYTE PTR [EXTFLG], 3    ; unofficial breakpoint?
397  F000:EA69  74 23                    JE      UNBPEX
398  F000:EA6B  B0 0B                    MOV     AL, 11          ; user exit message
399  F000:EA6D  E8 26 03                 CALL    USRERR          ; send CS and IP
400  F000:EA70  EB 5F                    JMP     SHORT EXRT
401
402  F000:EA72  C6 06 5A 01 FF    OFBPEX: MOV     BYTE PTR [BPRECH], FFh  ; set break point flag
403  F000:EA77  A1 40 01                 MOV     AX, [USRIP]     ; get IP
404  F000:EA7A  48                       DEC     AX              ; decrement (points to CC now)
405  F000:EA7B  A3 46 01                 MOV     [CURADR], AX    ; update current address
406  F000:EA7E  A3 40 01                 MOV     [USRIP], AX     ; update current IP
407  F000:EA81  A3 54 01                 MOV     [CMDPR2], AX    ; ready for exit message
408  F000:EA84  E8 D7 06                 CALL    CLRBRK          ; clear break point
409  F000:EA87  B0 09                    MOV     AL, 09          ; monitor breakpoint message
410  F000:EA89  E8 0A 03                 CALL    USRERR          ; send CS and IP
411  F000:EA8C  EB 43                    JMP     SHORT EXRT
412
413  F000:EA8E  B0 0A             UNBPEX: MOV     AL, 10          ; unofficial break point message
414  F000:EA90  E8 03 03                 CALL    USRERR          ; send CS and IP
415  F000:EA93  EB 3C                    JMP     SHORT EXRT
416
```

```
417  F000:EA95  C6 06 62 01 00     SSEXIT: MOV     BYTE PTR [STFLAG], 00h  ; clear single step flag
418  F000:EA9A  BB E5 01                   MOV     BX,OFFSET OUTLNE        ; point to start of line
419  F000:EA9D  A1 44 01                   MOV     AX, [CURSEG]            ; current segment
420  F000:EAA0  86 E0                      XCHG    AH, AL
421  F000:EAA2  E8 C8 00                   CALL    TOASC4
422  F000:EAA5  A1 46 01                   MOV     AX, [CURADR]            ; current address
423  F000:EAA8  86 E0                      XCHG    AH, AL
424  F000:EAAA  E8 C0 00                   CALL    TOASC4
425  F000:EAAD  06                         PUSH    ES
426  F000:EAAE  A1 44 01                   MOV     AX, [CURSEG]            ; address user area
427  F000:EAB1  8E C0                      MOV     ES, AX
428  F000:EAB3  8B 36 46 01                MOV     SI, [CURADR]
429  F000:EAB7  26 8A 04                   MOV     AL, ES:[SI]             ; get opcode byte
430  F000:EABA  07                         POP     ES
431  F000:EABB  E8 A4 00                   CALL    TOASC2
432  F000:EABE  80 3E 63 01 04             CMP     BYTE PTR [STSFX], 4     ; are registers needed ?
433  F000:EAC3  7C 09                      JL      $1
434  F000:EAC5  B9 0E 00                   MOV     CX, 14          ; number of registers to go
435  F000:EAC8  BE 28 01                   MOV     SI, USRAX       ; first register (ES = 0).
436  F000:EACB  E8 96 02                   CALL    SNDWRD
437  F000:EACE  E8 B6 02           $1:     CALL    ENDLNE
438  F000:EAD1  E9 BC FE           EXRT:   JMP     MLOOP           ; return to monitor command mode
439
440                                 ; ----- EXECUTE USER PROGRAM --------------------------------
441
442                                 ; GO command branches to here with user registers CS and IP set
443
444  F000:EAD4  C6 06 64 01 00     GOTOIT: MOV     BYTE PTR [EXTFLG], 0    ; clear exit flag
445  F000:EAD9  8B 1E 2A 01                MOV     BX, [USRBX]     ; BX
446  F000:EADD  8B 0E 2C 01                MOV     CX, [USRCX]     ; CX
447  F000:EAE1  8B 16 2E 01                MOV     DX, [USRDX]     ; DX
448  F000:EAE5  8B 2E 32 01                MOV     BP, [USRBP]     ; BP
449  F000:EAE9  8B 36 34 01                MOV     SI, [USRSI]     ; SI
450  F000:EAED  8B 3E 36 01                MOV     DI, [USRDI]     ; DI
451  F000:EAF1  A1 3A 01                   MOV     AX, [USRES]     ; ES
452  F000:EAF4  8E C0                      MOV     ES, AX
453                                                                 ; re-establish user stack
454  F000:EAF6  8B 26 30 01                MOV     SP, [USRSP]     ; SP
455  F000:EAFA  A1 3C 01                   MOV     AX, [USRSS]     ; SS
456  F000:EAFD  8E D0                      MOV     SS, AX
457                                                                 ; establish return conditions
458                                                                 ; on user stack
459  F000:EAFF  A1 42 01                   MOV     AX, [USRFG]     ; flags
460  F000:EB02  25 FF FE                   AND     AX, FEFFh       ; clear trap flag
461  F000:EB05  80 3E 62 01 00             CMP     BYTE PTR [STFLAG], 0    ; see if single stepping
462  F000:EB0A  74 03                      JE      $1
463
464  F000:EB0C  0D 00 01                   OR      AX, 0100h       ; single stepping so set trap
465  F000:EB0F  50                 $1:     PUSH    AX
466  F000:EB10  A1 3E 01                   MOV     AX, [USRCS]     ; CS
467  F000:EB13  50                         PUSH    AX
468  F000:EB14  A1 40 01                   MOV     AX, [USRIP]     ; IP
469  F000:EB17  50                         PUSH    AX
470  F000:EB18  A1 38 01                   MOV     AX, [USRDS]     ; save DS on user stack
471  F000:EB1B  50                         PUSH    AX
472  F000:EB1C  A1 28 01                   MOV     AX, [USRAX]     ; AX
473  F000:EB1F  1F                         POP     DS              ; DS
474  F000:EB20  CF                         IRET                    ; now excecute user routine.
475
476                                 ; ----- NON-MASKABLE INTERRUPT ROUTINE ----------------------
```

```
                                ; Default routine that only sets the EXTFLG to 05

F000:EB21  1E                   NMIDEF: PUSH    DS              ; save user DS & AX on stack
F000:EB22  50                           PUSH    AX
F000:EB23  B8 00 00                     MOV     AX, RAMSEG      ; monitor DS accessible again
F000:EB26  8E D8                        MOV     DS, AX
F000:EB28  C6 06 64 01 05               MOV     BYTE PTR [EXTFLG], 05h  ; set exit flag code
F000:EB2D  58                           POP     AX              ; recover user AX & DS
F000:EB2E  1F                           POP     DS
F000:EB2F  CF                           IRET

                                ; ----- DEFAULT 8259 ROUTINES --------------------------------

F000:EB30  CF                   IRQ0:   IRET
F000:EB31  CF                   IRQ1:   IRET
F000:EB32  CF                   IRQ2:   IRET
F000:EB33  CF                   IRQ3:   IRET
F000:EB34  CF                   IRQ4:   IRET
F000:EB35  CF                   IRQ5:   IRET
F000:EB36  CF                   IRQ6:   IRET
F000:EB37  CF                   IRQ7:   IRET

                                ; ----- ASCII CONVERSION ROUTINES ----------------------------

F000:EB38  3C 61                TOCAPS: CMP     AL, 'a'         ; converts AL to upper case
F000:EB3A  7C 06                        JL      $1
F000:EB3C  3C 7A                        CMP     AL,'z'
F000:EB3E  7F 02                        JG      $1
F000:EB40  2C 20                        SUB     AL, 20h         ; convert
F000:EB42  C3                   $1:     RET

F000:EB43  3C 41                TOLWR:  CMP     AL, 'A'         ; converts AL to lower case
F000:EB45  7C 06                        JL      $1
F000:EB47  3C 5A                        CMP     AL,'Z'
F000:EB49  7F 02                        JG      $1
F000:EB4B  04 20                        ADD     AL, 20h         ; convert
F000:EB4D  C3                   $1:     RET
                                                                ; Most sig nibble of AL to ascii
F000:EB4E  51                   MSNIB:  PUSH    CX
F000:EB4F  B1 04                        MOV     CL, 4
F000:EB51  D2 E8                        SHR     AL, CL          ; shift to ls nibble position
F000:EB53  59                           POP     CX
                                                                ; Least sig nibble of AL to ascii
F000:EB54  24 0F                LSNIB:  AND     AL, 0Fh         ; clear ms nibble
F000:EB56  04 30                        ADD     AL, 30h         ; convert to ascii
F000:EB58  3C 39                        CMP     AL, '9'
F000:EB5A  7E 02                        JLE     $1
F000:EB5C  04 07                        ADD     AL, 7h          ; if A to F finish conversion
F000:EB5E  88 07                $1:     MOV     [BX], AL        ; save character
F000:EB60  43                           INC     BX              ; leave pointing to next loc.
F000:EB61  C3                           RET
                                                                ; AL to ascii store at DS:BX
F000:EB62  8A E0                TOASC2: MOV     AH, AL          ; save it
F000:EB64  E8 E7 FF                     CALL    MSNIB           ; do most significant nibble
F000:EB67  8A C4                        MOV     AL, AH          ; recover
F000:EB69  E8 E8 FF                     CALL    LSNIB           ; do least significant nibble
F000:EB6C  C3                           RET
                                                                ; AX byte reversed, to ascii
                                                                ; store in 4 loc'tns after DS:BX
```

```
537   F000:EB6D   50
538   F000:EB6E   E8 F1 FF
539   F000:EB71   58
540   F000:EB72   8A C4
541   F000:EB74   E8 EB FF
542   F000:EB77   C3
543
544   F000:EB78   8A 07
545   F000:EB7A   2C 30
546   F000:EB7C   3C 09
547   F000:EB7E   7E 02
548   F000:EB80   2C 07
549   F000:EB82   43
550   F000:EB83   C3
551
552   F000:EB84   51
553   F000:EB85   E8 F0 FF
554   F000:EB88   8A E0
555   F000:EB8A   B1 04
556   F000:EB8C   D2 E4
557   F000:EB8E   E8 E7 FF
558   F000:EB91   02 C4
559   F000:EB93   59
560   F000:EB94   C3
561
562
563
564
565
566
567   F000:EB95   51
568   F000:EB96   E8 EB FF
569   F000:EB99   8A E8
570   F000:EB9B   E8 E6 FF
571   F000:EB9E   8A E5
572   F000:EBA0   59
573   F000:EBA1   C3
574
575
576
577   F000:EBA2   53
578   F000:EBA3   51
579   F000:EBA4   BB 02 E8
580   F000:EBA7   B9 00 00
581   F000:EBAA   2B C8
582   F000:EBAC   B8 00 00
583   F000:EBAF   2E 02 07
584   F000:EBB2   43
585   F000:EBB3   E2 FA
586   F000:EBB5   59
587   F000:EBB6   5B
588   F000:EBB7   A2 4B 01
589   F000:EBBA   C3
590
591   F000:EBBB   53
592   F000:EBBC   BB 00 00
593   F000:EBBF   8A 27
594   F000:EBC1   F6 17
595   F000:EBC3   8A 07
596   F000:EBC5   F6 D0
```

```
TOASC4: PUSH    AX              ; temporary
        CALL    TOASC2          ; convert AL
        POP     AX              ; recover
        MOV     AL, AH
        CALL    TOASC2          ; convert AH
        RET
                                ; ascii value at DS:BX to bin
TOBIN:  MOV     AL, [BX]        ; get ascii value
        SUB     AL, 30h         ; convert to binary
        CMP     AL, 9h
        JLE     $1
        SUB     AL, 7h          ; if A to F finish conversion
$1:     INC     BX              ; point to next location
        RET                     ; return in AL
                                ; 2 loctns at DS:BX from ascii
TOBIN2: PUSH    CX              ; for duration
        CALL    TOBIN
        MOV     AH, AL          ; save
        MOV     CL, 4
        SHL     AH, CL          ; put into ms nibble position
        CALL    TOBIN
        ADD     AL, AH          ; combine
        POP     CX
        RET                     ; return in AX

; 4 locations starting at DS:BX are converted from ascii into AX
; Enter with DS:BX pointing to storage location start address
; Exit with BX pointing to character after those converted
; WARNING - values '0' to '9' and 'A' to 'F' only

TOBIN4: PUSH    CX              ; for duration
        CALL    TOBIN2          ; convert first 2 bytes into AL
        MOV     CH, AL          ; save value so far
        CALL    TOBIN2          ; convert next 2 bytes into AL
        MOV     AH, CH          ; complete the word
        POP     CX
        RET

; ----- MEMORY TEST ROUTINES --------------------------------

ROMTST: PUSH    BX              ; Test ROM checksum
        PUSH    CX
        MOV     BX, OFFSET ME   ; start of MONITOR code
        MOV     CX, 0
        SUB     CX, BX          ; number of bytes to do
        MOV     AX, 0           ; initialise sum
$1:     ADD     AL, CS:[BX]     ; add in byte from ROM
        INC     BX              ; point to next
        LOOP    $1
        POP     CX
        POP     BX
        MOV     [ROMSUM], AL    ; save for later (should = 0)
        RET

RAMTST: PUSH    BX              ; Non-destructive RAM test
        MOV     BX,0            ; initialise pointer
$1:     MOV     AH,[BX]         ; get byte from RAM
        NOT     BYTE PTR [BX]   ; 1's complement it
        MOV     AL,[BX]         ; read it again
        NOT     AL              ; AH and AL should equal now
```

```
597   F000:EBC7   3A E0                  CMP     AH,AL           ; if equal must be RAM
598   F000:EBC9   75 05                  JNZ     $2              ; else not RAM
599   F000:EBCB   88 27                  MOV     [BX],AH         ; replace original value
600   F000:EBCD   43                     INC     BX              ; point to next
601   F000:EBCE   75 EF                  JNZ     $1              ; keep going
602
603   F000:EBD0   4B                $2:  DEC     BX              ; BX = last valid RAM address
604   F000:EBD1   89 1E 4C 01            MOV     [RAMLST], BX    ; save for later reports
605   F000:EBD5   5B                     POP     BX
606   F000:EBD6   C3                     RET
607
608                               ; ----- SERIAL COMMUNICATIONS ROUTINES -----------------------
609
610                                                       ; INT 6 routines. Enter operation in AL
611   F000:EBD7   14EC              SERTBL: DW      INIT    ; 0 to initialise the 8251
612   F000:EBD9   47EC                      DW      INSTAT  ; 1 to get status of RX comms
613   F000:EBDB   4CEC                      DW      INCHR   ; 2 to get character save in RXCHR
614                                                       ;   returns 00 if no char waiting
615   F000:EBDD   60EC                      DW      INLINE  ; 3 to get a line from serial ends in $
616                                                       ;   line at 0:INLNE returns CX = end
617   F000:EBDF   8EEC                      DW      OTSTAT  ; 4 see if 8251 ready to send
618   F000:EBE1   93EC                      DW      OTCHR   ; 5 to send char in memory TXCHR
619   F000:EBE3   B8EC                      DW      OTLINE  ; 6 to send line in 0:OUTLNE,
620                                                       ;   line must end in $ (not sent)
621                                                       ;   -$ added to end as terminator
622   F000:EBE5   D4EC                      DW      RSTLNE  ; 7 to send *$ reset to host
623   F000:EBE7   32EC                      DW      SBAUD   ; 8 set 8253 baud rate generator
624               F000:EBE9           SEREND: EQU     $
625
626   F000:EBE9   1E                SERIAL: PUSH    DS              ; execute command from table
627   F000:EBEA   57                        PUSH    DI
628   F000:EBEB   56                        PUSH    SI
629   F000:EBEC   53                        PUSH    BX
630   F000:EBED   50                        PUSH    AX              ; temporary
631   F000:EBEE   B4 00                     MOV     AH, 0           ; clear top byte
632   F000:EBF0   8B F0                     MOV     SI, AX          ; load pointer
633   F000:EBF2   B8 00 00                  MOV     AX, 0
634   F000:EBF5   8E D8                     MOV     DS, AX          ; RAM base
635   F000:EBF7   58                        POP     AX              ; recover
636   F000:EBF8   D1 E6                     SHL     SI, 1           ; x2 for table position
637   F000:EBFA   81 C6 D7 EB               ADD     SI, OFFSET CS:SERTBL
638   F000:EBFE   81 FE E9 EB               CMP     SI, OFFSET CS:SEREND    ; beyond end ?
639   F000:EC02   73 03                     JNC     $1
640   F000:EC04   2E FF 14                  CALL    CS:[SI]         ; execute routine
641   F000:EC07   5B                $1:     POP     BX
642   F000:EC08   5E                        POP     SI
643   F000:EC09   5F                        POP     DI
644   F000:EC0A   1F                        POP     DS
645   F000:EC0B   CF                        IRET
646
647   F000:EC0C   E6 1A             SUART:  OUT     UARTS, AL       ; set up UART
648   F000:EC0E   B9 00 02                  MOV     CX, 200h        ; delay to ensure hardware/host
649                                                       ; catches up, no handshake
650   F000:EC11   E2 FE             $1:     LOOP    $1              ; Approx 3us * 500 = 1.5 ms
651   F000:EC13   C3                        RET
652
653   F000:EC14   B0 00             INIT:   MOV     AL,0            ; send 3 times to ensure
654   F000:EC16   E8 F3 FF                  CALL    SUART           ; in command mode
655   F000:EC19   E8 F0 FF                  CALL    SUART
656   F000:EC1C   E8 ED FF                  CALL    SUART
```

```
F000:EC1F  B0 40              MOV     AL, 40h         ; put into command mode
F000:EC21  E8 E8 FF           CALL    SUART
F000:EC24  B0 4E              MOV     AL,.4Eh         ; 01001110
F000:EC26  E8 E3 FF           CALL    SUART
F000:EC29  B0 37              MOV     AL, 37h         ; 00110111
F000:EC2B  E8 DE FF           CALL    SUART
F000:EC2E  E8 01 00           CALL    SBAUD           ; set up speed
F000:EC31  C3                 RET

F000:EC32  B0 B6        SBAUD: MOV    AL, B6h         ; 10110110
F000:EC34  E6 0E              OUT     CTCK, AL        ; Counter 0, both, mode 3, Binary
F000:EC36  A0 4E 01           MOV     AL, BYTE PTR [BAUDRT]  ; ls byte
F000:EC39  E6 0C              OUT     CTC2, AL
F000:EC3B  A0 4F 01           MOV     AL, BYTE PTR [BAUDRT+1]  ; ms byte
F000:EC3E  E6 0C              OUT     CTC2, AL        ; start counter 2
F000:EC40  E8 9F 00           CALL    SRTDEL
F000:EC43  E8 9C 00           CALL    SRTDEL
F000:EC46  C3                 RET

F000:EC47  E4 1A        INSTAT: IN    AL, UARTS       ; get 8251 status
F000:EC49  24 02              AND     AL, 02h         ; is a character waiting ?
F000:EC4B  C3                 RET                     ; 02 if char or 00 if not

                        ; Read character from 8251 into RXCHR character buffer
                        ; Returns character in RXCHR or 00 in RXCHR if not one waiting

F000:EC4C  53           INCHR:  PUSH  BX
F000:EC4D  BB 48 01           MOV     BX, RXCHR       ; point to char storage
F000:EC50  E8 F4 FF           CALL    INSTAT          ; check status
F000:EC53  3C 00              CMP     AL, 00          ; is a character waiting
F000:EC55  74 05              JE      $1              ; if 00 then not one ready
F000:EC57  E4 18              IN      AL, UARTD       ; get character
F000:EC59  E8 DC FE           CALL    TOCAPS          ; convert to upper case
                                                      ; belts and braces in 'raw' mode
F000:EC5C  88 07        $1:   MOV     [BX], AL        ; save character in AL to RXCHR
F000:EC5E  5B                 POP     BX
F000:EC5F  C3                 RET

                        ; Saves serial data into INLNE will not return until completed
                        ; exit CX = number of chars
                        ; $ normal line termination - clear escape flag
                        ; @ escape code received - set flag - wait for $ terminator
                        ; interrupts killed while in operation

F000:EC60
F000:EC60  FA           INLINE: CLI
F000:EC61  53                 PUSH    BX
F000:EC62  FC                 CLD
F000:EC63  BB 65 01           MOV     BX, INLNE       ; point to start of line
F000:EC66  53                 PUSH    BX              ; save start address
F000:EC67  8B FB              MOV     DI, BX          ; set up pointer
F000:EC69  C6 06 4A 01 00     MOV     BYTE PTR ESCCHR, 00    ; clear escape flag
F000:EC6E  E8 DB FF     $1:   CALL    INCHR           ; get character - returned in AL
F000:EC71  3C 00              CMP     AL,00
F000:EC73  74 F9              JE      $1              ; wait until valid character
F000:EC75  AA                 STOSB                   ; store AL in line pointed by DI
F000:EC76  3C 40              CMP     AL,'@'          ; escape code
F000:EC78  75 07              JNE     $2              ; yes exit no change
F000:EC7A  C6 06 4A 01 40     MOV     BYTE PTR ESCCHR, 40h    ; set escape flag
F000:EC7F  EB ED              JMP     SHORT $1        ; wait for $ terminator
F000:EC81  3C 24        $2:   CMP     AL,'$'          ; end of line
```

```
717  F000:EC83  75 E9                    JNE     $1              ; no so get next
718  F000:EC85  4F                       DEC     DI              ; points to last character
719  F000:EC86  58                       POP     AX              ; recover start
720  F000:EC87  8B CF                    MOV     CX,DI           ; end
721  F000:EC89  2B C8                    SUB     CX,AX           ; return length in CX
722  F000:EC8B  5B                       POP     BX
723  F000:EC8C  FB                       STI
724  F000:EC8D  C3                       RET                     ; exit finished
725
726                                 ; checks status of 8251 if ready to send a character returns 01
727                                 ; if not ready returns 00, in AL
728
729  F000:EC8E  E4 1A            OTSTAT: IN      AL, UARTS
730  F000:EC90  24 01                    AND     AL, 01h         ; Is 8251 ready to send
731  F000:EC92  C3                       RET                     ; change to 81h to implement DSR
732
733                                 ; sends character in location TXCHR
734                                 ; if successful TXCHR = 0 if unsuccessful TXCHR unchanged
735
736  F000:EC93  BB 49 01         OTCHR:  MOV     BX, OFFSET TXCHR        ; point to storage
737  F000:EC96  E8 F5 FF                 CALL    OTSTAT
738  F000:EC99  3C 00                    CMP     AL, 00h
739  F000:EC9B  74 0C                    JE      $1
740  F000:EC9D  8A 07                    MOV     AL, [BX]        ; get character from TXCHR
741  F000:EC9F  E6 18                    OUT     UARTD, AL       ; send it
742  F000:ECA1  C6 07 00                 MOV     BYTE PTR [BX], 00       ; clear character sent
743  F000:ECA4  51                       PUSH    CX
744  F000:ECA5  E8 40 00                 CALL    OTDEL           ; ensure hardware catches up
745  F000:ECA8  59                       POP     CX
746  F000:ECA9  C3               $1:     RET
747
748  F000:ECAA  A2 49 01         OUTIT:  MOV     TXCHR, AL       ; send char in AL
749  F000:ECAD  E8 E3 FF                 CALL    OTCHR
750  F000:ECB0  80 3E 49 01 00           CMP     BYTE PTR TXCHR, 0
751  F000:ECB5  75 F3                    JNE     OUTIT           ; wait until gone
752  F000:ECB7  C3                       RET
753
754                                 ; will not return until a line is completely sent.
755                                 ; enter with start of line in BX - terminates in $
756
757  F000:ECB8  E8 27 00         OTLINE: CALL    SRTDEL
758  F000:ECBB  FC                       CLD
759  F000:ECBC  BE E5 01                 MOV     SI, OFFSET OUTLNE       ; point to line start
760  F000:ECBF  AC               $1:     LODSB                   ; get character from [SI] inc SI
761  F000:ECC0  3C 24                    CMP     AL, '$'         ; is it end of line ?
762  F000:ECC2  74 05                    JE      $2
763  F000:ECC4  E8 E3 FF                 CALL    OUTIT           ; will not return until gone
764  F000:ECC7  EB F6                    JMP     SHORT $1        ; do next character
765
766  F000:ECC9  B0 2D            $2:     MOV     AL, '-'         ; add in terminator characters
767  F000:ECCB  E8 DC FF                 CALL    OUTIT
768  F000:ECCE  B0 24                    MOV     AL, '$'
769  F000:ECD0  E8 D7 FF                 CALL    OUTIT
770  F000:ECD3  C3                       RET
771
772  F000:ECD4  E8 0B 00         RSTLNE: CALL    SRTDEL          ; reset indicator to host
773  F000:ECD7  B0 2A                    MOV     AL, '*'         ; reset flag
774  F000:ECD9  E8 CE FF                 CALL    OUTIT
775  F000:ECDC  B0 24                    MOV     AL, '$'         ; line terminator
776  F000:ECDE  E8 C9 FF                 CALL    OUTIT
```

```
F000:ECE1   C3                      RET
F000:ECE2   B9 00 20        SRTDEL: MOV     CX, 2000h       ; delay to ensure host waiting
F000:ECE5   E2 FE           $1:     LOOP    $1              ; Approx 3us * 8000 approx 25ms
F000:ECE7   C3                      RET
F000:ECE8   8B 0E 4E 01     OTDEL:  MOV     CX, [BAUDRT]    ; get current speed (V1.3)
F000:ECEC   B8 28 00                MOV     AX, 40          ; 40 for 4.77 mHz machine
F000:ECEF   F7 E1                   MUL     CX              ; 20 for 10 mHz machine. etc.
F000:ECF1   8B C8                   MOV     CX, AX          ; gives the host
F000:ECF3   E2 FE           $1:     LOOP    $1              ; time to catch up
F000:ECF5   C3                      RET
F000:ECF6   33 C0                   XOR     AX, AX          ; padding to keep addresses
F000:ECF8   33 C0                   XOR     AX, AX          ; same as for V1.2

                            ; ----- COMMAND ROUTINES ------------------------------------

F000:ECFA   F9ED            CMDTBL: DW      ESCAPE  ; 0 (@) abandon current command (not GO)
F000:ECFC   FCED                    DW      WRESET  ; 1 (A) Reset the target &
                                                    ;       initialise all variables
F000:ECFE   01EE                    DW      TEST    ; 2 (B) test target
                                                    ;       0 = RAM         1 = ROM
                                                    ;       2 = Production  3 = Speed
F000:ED00   8EEE                    DW      REGIS   ; 3 (C) interrogate register contents
                                                    ;       0 = get all     1 = get specific
                                                    ;       2 = update specific
F000:ED02   F2EE                    DW      MEMORY  ; 4 (D) access memory
                                                    ;       0 or 1 = access contents
                                                    ;       2 = new data for address
                                                    ;       3 = step to next address
                                                    ;       4 = step back an address
                                                    ;       5 or 6 dump memory block
F000:ED04   CAEF                    DW      FILL    ; 5 (E) fill memory block
                                                    ;       0 or 1 = fill byte data
                                                    ;       2 or 3 = fill word data
F000:ED06   35F0                    DW      INPORT  ; 6 (F) read data port
                                                    ;       0 = byte data  1 = word data
F000:ED08   5FF0                    DW      OTPORT  ; 7 (G) write data to port
                                                    ;       0 = byte data  1 = word data
F000:ED0A   83F0                    DW      GO      ; 8 (H) execute code
                                                    ;       0 = from current CS:IP
                                                    ;       1 = from IP specified
                                                    ;       2 = from CS:IP specified
F000:ED0C   BAF0                    DW      BREAK   ; 9 (I) access breakpoints
                                                    ;       0 = display state
                                                    ;       1 or 2 set breakpoint
                                                    ;       3 = remove breakpoint
F000:ED0E   83F1                    DW      SINGLE  ; 10 (J) single step code
                                                    ;       0 = from IP specified
                                                    ;       1 = from CS:IP specified
                                                    ;       2 = from IP specified + regs
                                                    ;       3 = from CS:IP specified + regs
F000:ED10   C5F1                    DW      UPLOAD  ; 11 (K) upload code
                                                    ;       0 = use current CS
                                                    ;       1 = from CS specified
F000:ED12   22F2                    DW      DNLOAD  ; 12 (L) down load code
F000:ED14   0D              CMDLEN: DB      ($-CMDTBL)/2

                            ; ---- ASSOCIATED SUBROUTINES -------------------------------
```

```
837
838  F000:ED15  BB 65 01        GETCMD: MOV   BX, OFFSET INLNE; point to start of line DS = 0
839  F000:ED18  B0 03                   MOV   AL,3           ; input serial line function
840  F000:ED1A  CD 06                   INT   6              ; exit length of line in CX
841  F000:ED1C  BB 65 01                MOV   BX, OFFSET INLNE; point to start of line
842  F000:ED1F  8A 07                   MOV   AL, [BX]       ; get command letter
843  F000:ED21  C3                      RET
844
845  F000:ED22  A1 3E 01        GETPRO: MOV   AX, [USRCS]    ; get user current CS:
846  F000:ED25  A3 52 01                MOV   [CMDPR1], AX
847  F000:ED28  EB 06                   JMP   SHORT GETX
848
849  F000:ED2A  E8 68 FE        GETPR1: CALL  TOBIN4
850  F000:ED2D  A3 52 01                MOV   CMDPR1, AX     ; segment address
851  F000:ED30  E8 62 FE        GETX:   CALL  TOBIN4
852  F000:ED33  A3 54 01                MOV   CMDPR2, AX     ; start address
853  F000:ED36  E8 5C FE                CALL  TOBIN4
854  F000:ED39  A3 56 01                MOV   CMDPR3, AX     ; end address
855  F000:ED3C  E8 56 FE                CALL  TOBIN4
856  F000:ED3F  A3 58 01                MOV   CMDPR4, AX     ; data
857  F000:ED42  C3                      RET
858
859  F000:ED43  BB E5 01        SNDDTA: MOV   BX,OFFSET OUTLNE       ; point to start of line
860  F000:ED46  A1 52 01                MOV   AX, [CMDPR1]
861  F000:ED49  86 E0                   XCHG  AH, AL                 ; byte reverse
862  F000:ED4B  E8 1F FE                CALL  TOASC4
863  F000:ED4E  A1 54 01                MOV   AX, [CMDPR2]
864  F000:ED51  86 E0                   XCHG  AH, AL                 ; byte reverse
865  F000:ED53  E8 17 FE                CALL  TOASC4
866  F000:ED56  E8 01 00                CALL  SNDBYT                 ; send stream of data.
867  F000:ED59  C3                      RET
868                                                 ; enter with numb bytes in CX
869  F000:ED5A  26 8A 04        SNDBYT: MOV   AL, ES:[SI]    ; get first byte
870  F000:ED5D  46                      INC   SI              ; point to next byte
871  F000:ED5E  E8 01 FE                CALL  TOASC2          ; convert to ascii & put in line
872  F000:ED61  E2 F7                   LOOP  SNDBYT
873  F000:ED63  C3                      RET
874                                                 ; enter with numb words in CX
875  F000:ED64  26 8B 04        SNDWRD: MOV   AX, ES:[SI]    ; get first word
876  F000:ED67  86 E0                   XCHG  AH, AL          ; byte reverse
877  F000:ED69  46                      INC   SI              ; point to next word
878  F000:ED6A  46                      INC   SI
879  F000:ED6B  E8 FF FD                CALL  TOASC4          ; convert to ascii & put in line
880  F000:ED6E  E2 F4                   LOOP  SNDWRD
881  F000:ED70  C3                      RET
882
883  F000:ED71  B0 06           NOTAVL: MOV   AL,06           ; command not available code
884  F000:ED73  EB 02                   JMP   SHORT EXTERR
885
886  F000:ED75  B0 00           NOTVAL: MOV   AL, 00                 ; command invalid code
887  F000:ED77  BB E5 01        EXTERR: MOV   BX,OFFSET OUTLNE       ; point to start of line
888  F000:ED7A  C6 07 3F                MOV   BYTE PTR [BX], '?'     ; error indicator
889  F000:ED7D  43                      INC   BX
890  F000:ED7E  E8 E1 FD                CALL  TOASC2                 ; convert error code
891  F000:ED81  E9 2C 00                JMP   SNDERR
892
893  F000:ED84  BB E5 01        NULLNE: MOV   BX,OFFSET OUTLNE       ; exit with null line
894  F000:ED87  C6 07 24        ENDLNE: MOV   BYTE PTR [BX], '$'     ; line terminator
895  F000:ED8A  BB E5 01                MOV   BX, OFFSET OUTLNE
896  F000:ED8D  B0 06                   MOV   AL, 6                  ; output serial line
```

```
F000:ED8F   CD 06                   INT     6
F000:ED91   B0 00                   MOV     AL, 0                   ; indicates ok
F000:ED93   C3                      RET
                                            ; segment in CMDPR1 address in CMDPR2
F000:ED94   B0 02           WRFAIL: MOV     AL, 02                  ; write failed code
F000:ED96   BB E5 01        USRERR: MOV     BX,OFFSET OUTLNE        ; point to start of line
F000:ED99   C6 07 3F                MOV     BYTE PTR [BX], '?'      ; error indicator
F000:ED9C   43                      INC     BX
F000:ED9D   E8 C2 FD                CALL    TOASC2                  ; error number
F000:EDA0   A1 52 01                MOV     AX, [CMDPR1]            ; error segment
F000:EDA3   86 E0                   XCHG    AH, AL
F000:EDA5   E8 C5 FD                CALL    TOASC4
F000:EDA8   A1 54 01                MOV     AX, [CMDPR2]            ; recover error address
F000:EDAB   86 E0                   XCHG    AH, AL
F000:EDAD   E8 BD FD                CALL    TOASC4
F000:EDB0   C6 07 24        SNDERR: MOV     BYTE PTR [BX], '$'      ; line terminator
F000:EDB3   BB E5 01                MOV     BX,OFFSET OUTLNE
F000:EDB6   B0 06                   MOV     AL, 6                   ; output serial line
F000:EDB8   CD 06                   INT     6
F000:EDBA   B0 FF                   MOV     AL, FFh                 ; indicates problem
F000:EDBC   C3                      RET

                            ; ---- PARSE COMMAND LINE -----------------------------------

F000:EDBD   E8 55 FF        COMND:  CALL    GETCMD          ; letter in AL suffix in AH
F000:EDC0   BB 65 01        OKCMD:  MOV     BX, OFFSET INLNE; point to start of line
F000:EDC3   A0 4A 01                MOV     AL, [ESCCHR]    ; is escape flag set ?
F000:EDC6   3C 40                   CMP     AL, '@'
F000:EDC8   75 02                   JNE     NOTESC
F000:EDCA   88 07                   MOV     [BX], AL        ; stuff escape code into line
F000:EDCC   8A 07           NOTESC: MOV     AL, [BX]        ; get command letter
F000:EDCE   3C 21                   CMP     AL, '!'         ; is it ! code
F000:EDD0   75 02                   JNE     NOTEX
F000:EDD2   B0 40                   MOV     AL, '@'         ; kid into thinking it's an @
F000:EDD4   2C 40           NOTEX:  SUB     AL, '@'         ; commands in range '@ to L'
F000:EDD6   A2 50 01                MOV     [CMDLTR], AL    ; save in parameter block
F000:EDD9   43                      INC     BX
F000:EDDA   8A 07                   MOV     AL, [BX]        ; get command suffix
F000:EDDC   2C 30                   SUB     AL, 30h         ; suffixes in range '0 to 9'
F000:EDDE   A2 51 01                MOV     [CMDSFX], AL    ; save in parameter block
F000:EDE1   43                      INC     BX
F000:EDE2   A0 50 01                MOV     AL, [CMDLTR]    ; load pointer
F000:EDE5   98                      CBW                     ; clear top byte of AX
F000:EDE6   2E 3A 06 14 ED          CMP     AL, CS:CMDLEN   ; is it beyond end ?
F000:EDEB   73 88                   JNC     NOTVAL
F000:EDED   D1 E0                   SHL     AX, 1                   ; x2 = table position
F000:EDEF   8B F0                   MOV     SI, AX                  ; set up pointer
F000:EDF1   81 C6 FA EC             ADD     SI, OFFSET CS:CMDTBL    ; add to start of table
F000:EDF5   2E FF 14                CALL    CS:[SI]                 ; execute routine
F000:EDF8   C3                      RET

                            ; ----- ESC @ ----------------------------------------------

F000:EDF9   E9 88 FF        ESCAPE: JMP     NULLNE                  ; exit with null line
F000:EDFC

                            ; ----- RESET A ---------------------------------------------

F000:EDFC   EA 6B E8 00 F0  WRESET: JMP     FAR WARM        ; same as hardware reset except
                                                            ; user registers left intact
```

```
                                ; ----- TEST B ----------------------------------------------

F000:EE01  A0 51 01       TEST: MOV   AL, [CMDSFX]
F000:EE04  3C 00              CMP   AL, 0           ; RAM test
F000:EE06  74 12              JE    TRAM
F000:EE08  3C 01              CMP   AL, 1           ; ROM test
F000:EE0A  74 32              JE    TROM
F000:EE0C  3C 02              CMP   AL, 2           ; production test
F000:EE0E  74 7A              JE    TPROD
F000:EE10  3C 03              CMP   AL, 3           ; set speed
F000:EE12  74 56              JE    TSPED
F000:EE14  58                 POP   AX              ; remove return address
F000:EE15  B0 08              MOV   AL, 08          ; suffix overflow
F000:EE17  E9 5D FF           JMP   EXTERR

F000:EE1A  E8 9E FD       TRAM: CALL  RAMTST
F000:EE1D  BB E5 01           MOV   BX, OFFSET OUTLNE       ; point to start of line
F000:EE20  A1 4C 01           MOV   AX, [RAMLST]
F000:EE23  3D 00 00           CMP   AX, 0                   ; ie no ram fitted
F000:EE26  75 05              JNE   $1
F000:EE28  B0 01              MOV   AL, 01                  ; no ram fitted 'error'
F000:EE2A  E9 4A FF           JMP   EXTERR
F000:EE2D  B8 00 00       $1: MOV   AX, 0000                ; segment always 0h
F000:EE30  E8 3A FD           CALL  TOASC4
F000:EE33  A1 4C 01           MOV   AX, [RAMLST]            ; get last RAM address
F000:EE36  86 E0              XCHG  AH, AL                  ; byte reverse
F000:EE38  E8 32 FD           CALL  TOASC4
F000:EE3B  E9 49 FF           JMP   ENDLNE

F000:EE3E  E8 61 FD       TROM: CALL  ROMTST
F000:EE41  BB E5 01           MOV   BX, OFFSET OUTLNE       ; point to start of line
F000:EE44  2E A1 F5 FF        MOV   AX, CS:[ROM1]           ; get ROM start address
F000:EE48  86 E0              XCHG  AH, AL                  ; byte reverse
F000:EE4A  E8 20 FD           CALL  TOASC4
F000:EE4D  A0 4B 01           MOV   AL, [ROMSUM]            ; actual rom checksum
F000:EE50  E8 0F FD           CALL  TOASC2
F000:EE53  1E                 PUSH  DS
F000:EE54  07                 POP   ES
F000:EE55  0E                 PUSH  CS
F000:EE56  1F                 POP   DS
F000:EE57  B9 08 00           MOV   CX, 8           ; number of characters to go
F000:EE5A  BE F7 FF           MOV   SI, OFFSET FLIGHT
F000:EE5D  FC                 CLD
F000:EE5E  AC             $1: LODSB                 ; get the byte
F000:EE5F  26 88 07           MOV   ES:[BX], AL
F000:EE62  43                 INC   BX
F000:EE63  E2 F9              LOOP  $1
F000:EE65  06                 PUSH  ES
F000:EE66  1F                 POP   DS
F000:EE67  E9 1D FF           JMP   ENDLNE

F000:EE6A  E8 17 FD       TSPED: CALL TOBIN2          ; get speed
F000:EE6D  3C 06              CMP   AL, 6
F000:EE6F  77 15              JA    $2              ; invalid speed
F000:EE71  8A C8              MOV   CL, AL
F000:EE73  B8 10 00           MOV   AX, 0010h       ; 153.6 kHz, x16 mode
F000:EE76  80 F9 00           CMP   CL, 0
F000:EE79  74 02              JE    $1              ; 9600 required
F000:EE7B  D3 E0              SHL   AX, CL          ; x setting
F000:EE7D  A3 4E 01       $1: MOV   WORD PTR BAUDRT, AX
```

```
1017  F000:EE80  E8 AF FD              CALL    SBAUD
1018  F000:EE83  E9 FE FE              JMP     NULLNE
1019
1020  F000:EE86  58            $2:     POP     AX              ; remove return address
1021  F000:EE87  E9 EB FE              JMP     NOTVAL          ; invalid command
1022
1023  F000:EE8A  58            TPROD:  POP     AX              ; remove return address
1024  F000:EE8B  E9 E3 FE              JMP     NOTAVL
1025
1026                           ; ----- REGISTER C ----------------------------------------
1027                           ; Registers    a  b  c  d  e  f  g  h  i  j  k  l  m  n
1028                           ;              AX BX CX DX SP BP SI DI DS ES SS CS IP FLAG
1029
1030  F000:EE8E  A0 51 01      REGIS:  MOV     AL, [CMDSFX]
1031  F000:EE91  3C 00                 CMP     AL, 0           ; dump all registers
1032  F000:EE93  74 0A                 JE      REG0
1033  F000:EE95  3C 03                 CMP     AL, 3           ; (1 or 2) specific registers
1034  F000:EE97  7C 15                 JL      REG12
1035  F000:EE99  58                    POP     AX
1036  F000:EE9A  B0 08                 MOV     AL, 08          ; suffix overflow
1037  F000:EE9C  E9 D8 FE              JMP     EXTERR
1038
1039  F000:EE9F  BB E5 01      REG0:   MOV     BX, OFFSET OUTLNE       ; point to start of line
1040  F000:EEA2  B9 0E 00              MOV     CX, 14                  ; number to go
1041  F000:EEA5  BE 28 01              MOV     SI, USRAX               ; first register (ES=0)
1042  F000:EEA8  E8 B9 FE              CALL    SNDWRD
1043  F000:EEAB  E9 D9 FE              JMP     ENDLNE
1044
1045  F000:EEAE  8A 07         REG12:  MOV     AL, BYTE PTR [BX]       ; get register number
1046  F000:EEB0  A2 58 01              MOV     BYTE PTR [CMDPR4], AL   ; save it
1047  F000:EEB3  43                    INC     BX                      ; point to data if any
1048  F000:EEB4  98                    CBW
1049  F000:EEB5  25 0F 00              AND     AX, 0Fh                 ; extract 1 to E
1050  F000:EEB8  3D 0E 00              CMP     AX, 0Eh                 ; make sure
1051  F000:EEBB  7F 31                 JG      REGE
1052  F000:EEBD  3D 00 00              CMP     AX, 00
1053  F000:EEC0  74 2C                 JE      REGE
1054  F000:EEC2  D1 E0                 SHL     AX, 1                   ; register number x 2
1055  F000:EEC4  BE 26 01              MOV     SI, USRAX-2             ; 2 before table start
1056  F000:EEC7  03 F0                 ADD     SI, AX                  ; point to register
1057  F000:EEC9  80 3E 51 01 01        CMP     BYTE PTR [CMDSFX], 1    ; get data only ?
1058  F000:EECE  74 09                 JE      REGR
1059  F000:EED0  E8 C2 FC              CALL    TOBIN4                  ; extract data
1060  F000:EED3  26 89 04              MOV     ES:[SI], AX             ; new data
1061  F000:EED6  E9 AB FE              JMP     NULLNE
1062
1063  F000:EED9  BB E5 01      REGR:   MOV     BX, OFFSET OUTLNE       ; point to start of line
1064  F000:EEDC  A0 58 01              MOV     AL, BYTE PTR [CMDPR4]   ; recover register numb
1065  F000:EEDF  E8 61 FC              CALL    TOLWR                   ; make lower case again
1066  F000:EEE2  88 07                 MOV     [BX], AL                ; insert into line
1067  F000:EEE4  43                    INC     BX
1068  F000:EEE5  B9 01 00              MOV     CX, 1                   ; only one reg to go
1069  F000:EEE8  E8 79 FE              CALL    SNDWRD
1070  F000:EEEB  E9 99 FE              JMP     ENDLNE
1071
1072  F000:EEEE  58            REGE:   POP     AX
1073  F000:EEEF  E9 83 FE              JMP     NOTVAL                  ; invalid register
1074
1075                           ; ----- MEMORY D ----------------------------------------
1076
```

```
1077  F000:EEF2  A0 51 01          MEMORY: MOV     AL, [CMDSFX]
1078  F000:EEF5  3C 00                     CMP     AL, 0   ; 1 byte memory dump no segment supplied
1079  F000:EEF7  74 1E                     JE      MEM0
1080  F000:EEF9  3C 01                     CMP     AL, 1   ; 1 byte memory dump segment supplied
1081  F000:EEFB  74 1F                     JE      MEM1
1082  F000:EEFD  3C 02                     CMP     AL, 2   ; write data to current address
1083  F000:EEFF  74 39                     JE      MEM2
1084  F000:EF01  3C 03                     CMP     AL, 3   ; increment current address
1085  F000:EF03  74 5C                     JE      MEM34
1086  F000:EF05  3C 04                     CMP     AL, 4   ; decrement current address
1087  F000:EF07  74 58                     JE      MEM34
1088  F000:EF09  3C 05                     CMP     AL, 5   ; memory dump no segment supplied
1089  F000:EF0B  74 72                     JE      MEM5
1090  F000:EF0D  3C 06                     CMP     AL, 6   ; memory dump segment supplied
1091  F000:EF0F  74 73                     JE      MEM6
1092  F000:EF11  58                        POP     AX
1093  F000:EF12  B0 08                     MOV     AL, 08          ; suffix overflow
1094  F000:EF14  E9 60 FE                  JMP     EXTERR
1095
1096  F000:EF17  E8 08 FE          MEM0:   CALL    GETPR0          ; no cs
1097  F000:EF1A  EB 03                     JMP     SHORT  MEMQ
1098  F000:EF1C  E8 0B FE          MEM1:   CALL    GETPR1          ; cs supplied
1099  F000:EF1F  06                MEMQ:   PUSH    ES
1100  F000:EF20  A1 52 01                  MOV     AX, [CMDPR1]    ; get segment address
1101  F000:EF23  A3 44 01                  MOV     [CURSEG], AX    ; save current segment
1102  F000:EF26  8E C0                     MOV     ES, AX
1103  F000:EF28  8B 36 54 01               MOV     SI, [CMDPR2]    ; get start address = ES:SI
1104  F000:EF2C  89 36 46 01               MOV     [CURADR], SI    ; save current address
1105  F000:EF30  B9 01 00                  MOV     CX,1            ; one byte to go
1106  F000:EF33  E8 0D FE                  CALL    SNDDTA          ; put into line
1107  F000:EF36  07                        POP     ES
1108  F000:EF37  E9 4D FE                  JMP     ENDLNE
1109
1110  F000:EF3A  E8 47 FC          MEM2:   CALL    TOBIN2
1111  F000:EF3D  8A D0                     MOV     DL, AL                  ; get data supplied
1112  F000:EF3F  06                        PUSH    ES
1113  F000:EF40  A1 44 01                  MOV     AX, [CURSEG]            ; get current segment
1114  F000:EF43  A3 52 01                  MOV     [CMDPR1],AX             ; pseudo line data
1115  F000:EF46  8E C0                     MOV     ES, AX
1116  F000:EF48  8B 1E 46 01               MOV     BX, [CURADR]            ; get current address
1117  F000:EF4C  89 1E 54 01               MOV     [CMDPR2], BX            ; pseudo command
1118  F000:EF50  26 88 17                  MOV     BYTE PTR ES:[BX], DL    ; store data
1119  F000:EF53  26 8A 37                  MOV     DH, BYTE PTR ES:[BX]    ; did it work
1120  F000:EF56  07                        POP     ES
1121  F000:EF57  3A D6                     CMP     DL, DH
1122  F000:EF59  75 02                     JNE     MEM2E                   ; error
1123  F000:EF5B  EB C2                     JMP     SHORT MEMQ              ; return normal line
1124
1125  F000:EF5D  58                MEM2E:  POP     AX
1126  F000:EF5E  E9 33 FE                  JMP     WRFAIL
1127
1128  F000:EF61  A1 44 01          MEM34:  MOV     AX, [CURSEG]            ; get current segment
1129  F000:EF64  A3 52 01                  MOV     [CMDPR1],AX             ; pseudo line data
1130  F000:EF67  A1 46 01                  MOV     AX, [CURADR]            ; get current address
1131  F000:EF6A  80 3E 51 01 03            CMP     BYTE PTR [CMDSFX], 3    ; is it increment ?
1132  F000:EF6F  75 03                     JNE     MEM34S
1133  F000:EF71  05 02 00                  ADD     AX, 2                   ; increment by 2
1134  F000:EF74  2D 01 00          MEM34S: SUB     AX, 1                   ; decrement by 1
1135  F000:EF77  A3 54 01                  MOV     [CMDPR2], AX            ; pseudo line data
1136  F000:EF7A  A3 46 01                  MOV     [CURADR], AX            ; update current address
```

```
1137  F000:EF7D  EB A0                    JMP     SHORT MEMQ              ; return new values
1138
1139  F000:EF7F  E8 A0 FD          MEM5:  CALL    GETPR0                  ; no cs supplied
1140  F000:EF82  EB 03                    JMP     SHORT MEMD
1141  F000:EF84  E8 A3 FD          MEM6:  CALL    GETPR1                  ; cs supplied
1142  F000:EF87  81 26 54 01 F0 FF MEMD:  AND     WORD PTR [CMDPR2], FFF0h ; ensure ls nibble = 0
1143  F000:EF8D  81 0E 56 01 0F 00         OR      WORD PTR [CMDPR3], 000Fh ; ensure ls nibble = F
1144  F000:EF93  06                MEMS:  PUSH    ES                      ; can now use line again
1145  F000:EF94  A1 52 01                  MOV     AX, [CMDPR1]            ; get segment address
1146  F000:EF97  8E C0                     MOV     ES, AX
1147  F000:EF99  8B 36 54 01               MOV     SI, [CMDPR2]            ; get start addr = ES:SI
1148  F000:EF9D  B9 10 00                  MOV     CX, 16                  ; number of chars to go
1149  F000:EFA0  E8 A0 FD                  CALL    SNDDTA
1150  F000:EFA3  07                        POP     ES
1151  F000:EFA4  83 06 54 01 10            ADD     WORD PTR [CMDPR2], 10h  ; update start address
1152  F000:EFA9  A1 54 01                  MOV     AX, [CMDPR2]            ; special case
1153  F000:EFAC  3D 00 00                  CMP     AX, 0                   ; test for over flow
1154  F000:EFAF  74 09                     JE      MEMX
1155  F000:EFB1  A1 56 01                  MOV     AX,[CMDPR3]             ; for next line
1156  F000:EFB4  3B 06 54 01               CMP     AX,[CMDPR2]             ; is start > end ?
1157  F000:EFB8  77 03                     JA      MEMWT
1158  F000:EFBA  E9 CA FD          MEMX:  JMP     ENDLNE
1159
1160  F000:EFBD  E8 C7 FD          MEMWT: CALL    ENDLNE                  ; send this line
1161  F000:EFC0  E8 52 FD                  CALL    GETCMD                  ; get command letter
1162  F000:EFC3  3C 21                     CMP     AL,'!'
1163  F000:EFC5  74 CC                     JE      MEMS                    ; return next line
1164  F000:EFC7  E9 F6 FD                  JMP     OKCMD                   ; new command so abandon
1165
1166                               ; ----- Fill E ----------------------------------------------
1167
1168  F000:EFCA  A0 51 01          FILL:  MOV     AL, [CMDSFX]
1169  F000:EFCD  24 01                    AND     AL, 01
1170  F000:EFCF  3C 00                    CMP     AL, 0           ; no CS; byte or word
1171  F000:EFD1  74 0A                    JE      FILL0
1172  F000:EFD3  3C 01                    CMP     AL, 1           ; CS; byte or word
1173  F000:EFD5  74 0B                    JE      FILL1
1174  F000:EFD7  58                       POP     AX
1175  F000:EFD8  B0 08                    MOV     AL, 08          ; suffix overflow
1176  F000:EFDA  E9 9A FD                 JMP     EXTERR
1177
1178  F000:EFDD  E8 42 FD          FILL0: CALL    GETPR0                  ; no CS supplied
1179  F000:EFE0  EB 03                    JMP     SHORT FILLS
1180  F000:EFE2  E8 45 FD          FILL1: CALL    GETPR1                  ; CS supplied
1181  F000:EFE5  06                FILLS: PUSH    ES
1182  F000:EFE6  A1 52 01                  MOV     AX, [CMDPR1]            ; get segment address
1183  F000:EFE9  8E C0                     MOV     ES, AX
1184  F000:EFEB  8B 1E 54 01               MOV     BX, [CMDPR2]            ; get start address
1185  F000:EFEF  8B 0E 56 01               MOV     CX, [CMDPR3]            ; get end address
1186  F000:EFF3  2B CB                     SUB     CX, BX
1187  F000:EFF5  83 C1 01                  ADD     CX, 1                   ; number of bytes
1188  F000:EFF8  80 3E 51 01 02            CMP     BYTE PTR [CMDSFX], 2    ; 0-1 = byte; 2-3 = word
1189  F000:EFFD  7C 16                     JL      FILLB
1190  F000:EFFF  A1 58 01                  MOV     AX, [CMDPR4]            ; get the data
1191  F000:F002  83 C1 01                  ADD     CX, 1                   ; ensure words done
1192  F000:F005  D1 E9                     SHR     CX, 1                   ; divide by 2
1193  F000:F007  26 89 07          FILLW: MOV     ES:[BX], AX             ; store word
1194  F000:F00A  26 3B 07                  CMP     AX, ES:[BX]             ; is it ok
1195  F000:F00D  75 1D                     JNE     FILLE                   ; error
1196  F000:F00F  43                        INC     BX
```

```
1197  F000:F010  43                    INC     BX
1198  F000:F011  E2 F4                 LOOP    FILLW
1199  F000:F013  EB 10                 JMP     SHORT  FILLF
1200  F000:F015
1201  F000:F015  A0 59 01        FILLB: MOV     AL, [CMDPR4+1]          ; get data
1202  F000:F018  26 88 07        FLB:   MOV     BYTE PTR ES:[BX], AL    ; store it
1203  F000:F01B  26 8A 27              MOV     AH, BYTE PTR ES:[BX]    ; did it work
1204  F000:F01E  3A C4                 CMP     AL, AH
1205  F000:F020  75 0A                 JNE     FILLE                   ; error
1206  F000:F022  43                    INC     BX
1207  F000:F023  E2 F3                 LOOP    FLB
1208  F000:F025  07              FILLF: POP     ES
1209  F000:F026  BB E5 01              MOV     BX, OFFSET OUTLNE       ; start of line
1210  F000:F029  E9 5B FD              JMP     ENDLNE                  ; return -$
1211
1212  F000:F02C  89 1E 54 01     FILLE: MOV     [CMDPR2], BX            ; save failed address
1213  F000:F030  07                    POP     ES
1214  F000:F031  58                    POP     AX                      ; remove return address
1215  F000:F032  E9 5F FD              JMP     WRFAIL
1216  F000:F035
1217                             ; ----- PORT INPUT F ----------------------------------------
1218
1219  F000:F035  E8 5D FB        INPORT: CALL    TOBIN4
1220  F000:F038  8B D0                 MOV     DX, AX          ; port address
1221  F000:F03A  A0 51 01              MOV     AL, [CMDSFX]
1222  F000:F03D  3C 00                 CMP     AL, 0           ; byte to be read
1223  F000:F03F  74 0A                 JE      INP0
1224  F000:F041  3C 01                 CMP     AL, 1           ; word to be read
1225  F000:F043  74 10                 JE      INP1
1226  F000:F045  58                    POP     AX
1227  F000:F046  B0 08                 MOV     AL, 08          ; suffix overflow
1228  F000:F048  E9 2C FD              JMP     EXTERR
1229
1230  F000:F04B  EC              INP0:  IN      AL, DX                  ; now get the data
1231  F000:F04C  BB E5 01              MOV     BX,OFFSET OUTLNE        ; point to start of line
1232  F000:F04F  E8 10 FB              CALL    TOASC2
1233  F000:F052  E9 32 FD              JMP     ENDLNE
1234
1235  F000:F055  ED              INP1:  IN      AX, DX                  ; now get the data
1236  F000:F056  BB E5 01              MOV     BX,OFFSET OUTLNE        ; point to start of line
1237  F000:F059  E8 11 FB              CALL    TOASC4
1238  F000:F05C  E9 28 FD              JMP     ENDLNE
1239
1240                             ; ----- PORT OUTPUT G ---------------------------------------
1241  F000:F05F
1242  F000:F05F  E8 33 FB        OTPORT: CALL    TOBIN4
1243  F000:F062  8B D0                 MOV     DX, AX          ; port address
1244  F000:F064  A0 51 01              MOV     AL, [CMDSFX]
1245  F000:F067  3C 00                 CMP     AL, 0           ; byte to be written
1246  F000:F069  74 0A                 JE      OUTP0
1247  F000:F06B  3C 01                 CMP     AL, 1           ; word to be written
1248  F000:F06D  74 0D                 JE      OUTP1
1249  F000:F06F  58                    POP     AX
1250  F000:F070  B0 08                 MOV     AL, 08          ; suffix overflow
1251  F000:F072  E9 02 FD              JMP     EXTERR
1252
1253  F000:F075  E8 0C FB        OUTP0:  CALL    TOBIN2          ; get data to go
1254  F000:F078  EE                    OUT     DX, AL          ; now send the byte
1255  F000:F079  E9 08 FD              JMP     NULLNE
1256
```

```
1257  F000:F07C  E8 16 FB          OUTP1:  CALL    TOBIN4          ; get data to go
1258  F000:F07F  EF                         OUT     DX, AX          ; now send the word
1259  F000:F080  E9 01 FD                   JMP     NULLNE
1260
1261                               ; ----- GO H ----------------------------------------------
1262
1263  F000:F083  A0 51 01          GO:     MOV     AL, [CMDSFX]
1264  F000:F086  3C 00                     CMP     AL, 0           ; use user CS:IP
1265  F000:F088  74 28                     JE      GOTT
1266  F000:F08A  3C 01                     CMP     AL, 1           ; use user CS supplied IP
1267  F000:F08C  74 0A                     JE      GO1
1268  F000:F08E  3C 02                     CMP     AL, 2           ; use supplied CS:IP
1269  F000:F090  74 0B                     JE      GO2
1270  F000:F092  58                       POP     AX
1271  F000:F093  B0 08                     MOV     AL, 08          ; suffix overflow
1272  F000:F095  E9 DF FC                  JMP     EXTERR
1273
1274  F000:F098  E8 87 FC          GO1:    CALL    GETPR0          ; no CS supplied
1275  F000:F09B  EB 03                     JMP     SHORT GOS
1276  F000:F09D  E8 8A FC          GO2:    CALL    GETPR1          ; CS supplied
1277  F000:F0A0  A1 52 01          GOS:    MOV     AX, [CMDPR1]
1278  F000:F0A3  A3 3E 01                  MOV     [USRCS], AX     ; set CS
1279  F000:F0A6  A3 44 01                  MOV     [CURSEG], AX
1280  F000:F0A9  A1 54 01                  MOV     AX, [CMDPR2]
1281  F000:F0AC  A3 40 01                  MOV     [USRIP], AX     ; set IP
1282  F000:F0AF  A3 46 01                  MOV     [CURADR], AX
1283  F000:F0B2  C6 06 62 01 00    GOTT:   MOV     BYTE PTR [STFLAG], 00h  ; clear single step flag
1284  F000:F0B7  E9 1A FA                  JMP     GOTOIT  ; execute user code using current CS:IP
1285
1286                               ; ----- BREAK POINT I ----------------------------------------
1287
1288  F000:F0BA  A0 51 01          BREAK:  MOV     AL, [CMDSFX]
1289  F000:F0BD  3C 00                     CMP     AL, 0           ; display breakpoints set
1290  F000:F0BF  74 12                     JE      BRK0
1291  F000:F0C1  3C 01                     CMP     AL, 1           ; set no cs supplied
1292  F000:F0C3  74 59                     JE      BRK1
1293  F000:F0C5  3C 02                     CMP     AL, 2           ; set cs supplied
1294  F000:F0C7  74 5A                     JE      BRK2
1295  F000:F0C9  3C 03                     CMP     AL, 3           ; remove break points
1296  F000:F0CB  74 3D                     JE      BRK3
1297  F000:F0CD  58                       POP     AX
1298  F000:F0CE  B0 08                     MOV     AL, 08h         ; suffix overflow
1299  F000:F0D0  E9 A4 FC                  JMP     EXTERR
1300
1301  F000:F0D3  80 3E 5B 01 00    BRK0:   CMP     BYTE PTR [BPFLAG], 00   ; is one set ?
1302  F000:F0D8  74 20                     JE      BRK0E
1303  F000:F0DA  BB E5 01                  MOV     BX, OFFSET OUTLNE       ; point to start of line
1304  F000:F0DD  B0 31                     MOV     AL, '1'                 ; breakpoint number = 1
1305  F000:F0DF  88 07                     MOV     [BX], AL
1306  F000:F0E1  43                        INC     BX
1307  F000:F0E2  B0 31                     MOV     AL, '1'                 ; '1' = set
1308  F000:F0E4  88 07                     MOV     [BX], AL
1309  F000:F0E6  43                        INC     BX
1310  F000:F0E7  A1 5D 01                  MOV     AX, [BPSEG]             ; segment address
1311  F000:F0EA  86 E0                     XCHG    AH, AL
1312  F000:F0EC  E8 7E FA                  CALL    TOASC4
1313  F000:F0EF  A1 5F 01                  MOV     AX, [BPADR]             ; address
1314  F000:F0F2  86 E0                     XCHG    AH, AL
1315  F000:F0F4  E8 76 FA                  CALL    TOASC4
1316  F000:F0F7  E9 8D FC                  JMP     ENDLNE
```

```
1317
1318  F000:F0FA  BB E5 01          BRK0E:  MOV     BX,OFFSET OUTLNE        ; point to start of line
1319  F000:F0FD  B0 31                     MOV     AL, '1'                 ; breakpoint number
1320  F000:F0FF  88 07                     MOV     [BX], AL
1321  F000:F101  43                        INC     BX
1322  F000:F102  B0 30                     MOV     AL, '0'                 ; '0' = not set
1323  F000:F104  88 07                     MOV     [BX], AL
1324  F000:F106  43                        INC     BX
1325  F000:F107  E9 7D FC                  JMP     ENDLNE
1326
1327  F000:F10A  8A 07             BRK3:   MOV     AL, [BX]                ; get breakpoint number
1328  F000:F10C  3C 30                     CMP     AL, '0'                 ; '0' = all break points
1329  F000:F10E  74 08                     JE      $1
1330  F000:F110  3C 31                     CMP     AL, '1'                 ; '1' for breakpoint 1
1331  F000:F112  74 04                     JE      $1
1332  F000:F114  58                        POP     AX
1333  F000:F115  E9 59 FC                  JMP     NOTAVL                  ; 'oops
1334
1335  F000:F118  E8 43 00          $1:     CALL    CLRBRK                  ; clear breakpoint
1336  F000:F11B  E9 66 FC                  JMP     NULLNE
1337
1338  F000:F11E  E8 01 FC          BRK1:   CALL    GETPR0          ; no CS supplied
1339  F000:F121  EB 03                     JMP     SHORT BRKS
1340  F000:F123  E8 04 FC          BRK2:   CALL    GETPR1          ; CS supplied
1341  F000:F126  E8 35 00          BRKS:   CALL    CLRBRK          ; clear it in case already set
1342  F000:F129  06                        PUSH    ES
1343  F000:F12A  A1 52 01                  MOV     AX, [CMDPR1]            ; set segment address
1344  F000:F12D  8E C0                     MOV     ES, AX
1345  F000:F12F  8B 36 54 01               MOV     SI, [CMDPR2]            ; get address
1346  F000:F133  26 8A 04                  MOV     AL, ES:[SI]             ; get current opcode
1347  F000:F136  A2 3C 01                  MOV     BYTE PTR [BPCODE], AL   ; save it
1348  F000:F139  26 C6 04 CC               MOV     BYTE PTR ES:[SI], CCh   ; install breakpoint
1349  F000:F13D  26 8A 04                  MOV     AL, ES:[SI]             ; check it went ok
1350  F000:F140  3C CC                     CMP     AL, CCh
1351  F000:F142  75 13                     JNE     BRKSE
1352  F000:F144  C6 06 5B 01 FF            MOV     BYTE PTR [BPFLAG], FFh  ; set breakpoint flag
1353  F000:F149  89 36 5F 01               MOV     [BPADR], SI             ; save address
1354  F000:F14D  8C C0                     MOV     AX, ES
1355  F000:F14F  8C 06 5D 01               MOV     [BPSEG], ES             ; save segment
1356  F000:F153  07                        POP     ES
1357  F000:F154  E9 2D FC                  JMP     NULLNE
1358
1359  F000:F157  07                BRKSE:  POP     ES
1360  F000:F158  58                        POP     AX
1361  F000:F159  B0 03                     MOV     AL, 03                  ; cannot set breakpoint
1362  F000:F15B  E9 38 FC                  JMP     USRERR
1363                                                                        ; clear break point
1364  F000:F15E  80 3E 5B 01 00    CLRBRK: CMP     BYTE PTR [BPFLAG], 0    ; is one set ?
1365  F000:F163  74 1D                     JE      $3                      ; no so do nothing
1366  F000:F165  06                        PUSH    ES
1367  F000:F166  C6 06 5B 01 00            MOV     BYTE PTR [BPFLAG], 0    ; clear flag what ever
1368  F000:F16B  A1 5D 01                   MOV     AX, [BPSEG]             ; segment address
1369  F000:F16E  8E C0                      MOV     ES, AX                  ; set segment address
1370  F000:F170  8B 36 5F 01                MOV     SI, [BPADR]             ; set address
1371  F000:F174  26 8A 04                   MOV     AL, BYTE PTR ES:[SI]    ; get code already there
1372  F000:F177  3C CC                      CMP     AL, CCh                 ; is it break point code
1373  F000:F179  75 06                      JNE     $2                      ; no so do nothing more
1374  F000:F17B  A0 5C 01                   MOV     AL, [BPCODE]            ; get removed code
1375  F000:F17E  26 88 04                   MOV     BYTE PTR ES:[SI], AL    ; restore original code
1376  F000:F181  07                $2:     POP     ES
```

```
1377  F000:F182  C3                  $3:     RET
1378
1379                                ; ----- SINGLE STEP J ------------------------------------

1380
1381  F000:F183  A0 51 01           SINGLE: MOV     AL, [CMDSFX]
1382  F000:F186  A2 63 01                   MOV     [STSFX], AL     ; save for return from execution
1383  F000:F189  24 03                      AND     AL, 03          ; so only need to test 2 bits
1384  F000:F18B  3C 00                      CMP     AL, 0           ; nothing specified
1385  F000:F18D  74 22                      JE      SNOR
1386  F000:F18F  3C 01                      CMP     AL, 1           ; from IP specified
1387  F000:F191  74 0A                      JE      SNG0
1388  F000:F193  3C 02                      CMP     AL, 2           ; from CS:IP specified
1389  F000:F195  74 08                      JE      SNG1
1390  F000:F197  58                        POP     AX
1391  F000:F198  B0 08                      MOV     AL, 08h         ; suffix overflow
1392  F000:F19A  E9 DA FB                   JMP     EXTERR
1393
1394  F000:F19D  E8 82 FB           SNG0:   CALL    GETPR0          ; no CS supplied
1395  F000:F1A0  EB 03                      JMP     SHORT SNGS
1396  F000:F1A2  E8 85 FB           SNG1:   CALL    GETPR1          ; CS supplied
1397  F000:F1A5  A1 52 01           SNGS:   MOV     AX, [CMDPR1]
1398  F000:F1A8  A3 3E 01                   MOV     [USRCS], AX     ; set CS
1399  F000:F1AB  A1 54 01                   MOV     AX, [CMDPR2]
1400  F000:F1AE  A3 40 01                   MOV     [USRIP], AX     ; set IP
1401  F000:F1B1  A1 3E 01           SNOR:   MOV     AX, [USRCS]
1402  F000:F1B4  A3 44 01                   MOV     [CURSEG], AX
1403  F000:F1B7  A1 40 01                   MOV     AX, [USRIP]
1404  F000:F1BA  A3 46 01                   MOV     [CURADR], AX
1405  F000:F1BD  C6 06 62 01 FF             MOV     BYTE PTR [STFLAG], FFh  ; set single step flag
1406  F000:F1C2  E9 0F F9                   JMP     GOTOIT
1407
1408                                ; ----- UPLOAD K ------------------------------------------

1409
1410  F000:F1C5  A0 51 01           UPLOAD: MOV     AL, [CMDSFX]
1411  F000:F1C8  3C 00                      CMP     AL, 0           ; no segment specified
1412  F000:F1CA  74 0A                      JE      UP0
1413  F000:F1CC  3C 01                      CMP     AL, 1           ; segment specified
1414  F000:F1CE  74 0B                      JE      UP1
1415  F000:F1D0  58                         POP     AX              ; remove return address
1416  F000:F1D1  B0 08                      MOV     AL, 08          ; suffix overflow
1417  F000:F1D3  E9 A1 FB                   JMP     EXTERR
1418
1419  F000:F1D6  E8 49 FB           UP0:    CALL    GETPR0          ; no CS supplied
1420  F000:F1D9  EB 03                      JMP     SHORT UPS
1421  F000:F1DB  E8 4C FB           UP1:    CALL    GETPR1          ; CS supplied
1422  F000:F1DE  06                 UPS:    PUSH    ES
1423  F000:F1DF  A1 52 01                   MOV     AX, [CMDPR1]    ; get segment address
1424  F000:F1E2  8E C0                      MOV     ES, AX
1425  F000:F1E4  8B 36 54 01                MOV     SI, [CMDPR2]    ; get start = ES:SI
1426  F000:F1E8  8B CE                      MOV     CX, SI
1427  F000:F1EA  81 C9 0F 00                OR      CX, 000Fh       ; possible end of line
1428  F000:F1EE  39 0E 56 01                CMP     [CMDPR3], CX
1429  F000:F1F2  77 04                      JA      UPMR
1430  F000:F1F4  8B 0E 56 01                MOV     CX, [CMDPR3]    ; finished
1431  F000:F1F8  2B 0E 54 01        UPMR:   SUB     CX, [CMDPR2]    ; number of chars to go
1432  F000:F1FC  83 C1 01                   ADD     CX, 1
1433  F000:F1FF  51                         PUSH    CX
1434  F000:F200  E8 40 FB                   CALL    SNDDTA
1435  F000:F203  59                         POP     CX
1436  F000:F204  07                         POP     ES
```

```
1437  F000:F205  01 0E 54 01          ADD     [CMDPR2], CX    ; update for next line
1438  F000:F209  E9 D6 00             JMP     V13B            ; Version 1.3 patch
1439  F000:F20C  3B 06 54 01  V13BR:  CMP     AX, [CMDPR2]    ; is start > end ?
1440  F000:F210  73 03                JAE     UPWT
1441  F000:F212  E9 72 FB            JMP     ENDLNE
1442
1443  F000:F215  E8 0F FB     UPWT:   CALL    ENDLNE          ; send this line
1444  F000:F218  E8 FA FA             CALL    GETCMD          ; returns command letter
1445  F000:F21B  3C 21                CMP     AL, '!'
1446  F000:F21D  74 BF                JE      UPS
1447  F000:F21F  E9 9E FB             JMP     OKCMD           ; new command so abandon
1448
1449                            ; ----- DOWNLOAD L ----------------------------------------
1450                            ; format nnaaaattxxxxyyyy etc  (checksum follows)
1451                            ; tt = 00 data follows; tt = 01 end record ; tt = 02 xxxx = CS:
1452                            ; tt = 03 xxxxyyyy = CS:IP start address
1453
1454  F000:F222  4B           DNLOAD: DEC     BX
1455  F000:F223  E8 5E F9             CALL    TOBIN2
1456  F000:F226  98                   CBW
1457  F000:F227  A3 50 01             MOV     [CMDLTR], AX    ; nn number of data records
1458  F000:F22A  E8 68 F9             CALL    TOBIN4
1459  F000:F22D  A3 52 01             MOV     [CMDPR1], AX    ; aaaa address
1460  F000:F230  E8 51 F9             CALL    TOBIN2
1461  F000:F233  98                   CBW
1462  F000:F234  A3 54 01             MOV     [CMDPR2], AX    ; tt record type
1463  F000:F237  E8 5B F9             CALL    TOBIN4
1464  F000:F23A  A3 56 01             MOV     [CMDPR3], AX    ; xxxx segment address
1465  F000:F23D  E8 55 F9             CALL    TOBIN4
1466  F000:F240  A3 58 01             MOV     [CMDPR4], AX    ; yyyy start address
1467  F000:F243  A1 54 01             MOV     AX, [CMDPR2]
1468  F000:F246  3C 00                CMP     AL, 0           ; data record
1469  F000:F248  74 32                JE      DWN0
1470  F000:F24A  3C 01                CMP     AL, 1           ; end record
1471  F000:F24C  74 67                JE      DWN1
1472  F000:F24E  3C 02                CMP     AL, 2           ; segment address record
1473  F000:F250  74 69                JE      DWN2
1474  F000:F252  3C 03                CMP     AL, 3           ; start CS:IP record
1475  F000:F254  74 74                JE      DWN3
1476  F000:F256  58                   POP     AX
1477  F000:F257  B0 07                MOV     AL, 07          ; invalid hex line
1478  F000:F259  E9 1B FB             JMP     EXTERR
1479                                                          ; line checksum routine
1480  F000:F25C  33 D2        CHKCHK: XOR     DX,DX           ; clear sum
1481  F000:F25E  B9 05 00             MOV     CX,5            ; always 5 bytes in line
1482  F000:F261  03 0E 50 01          ADD     CX, [CMDLTR]    ; gives the total in line
1483  F000:F265  BB 66 01             MOV     BX, INLNE + 1   ; start of data
1484  F000:F268  E8 19 F9     $1:     CALL    TOBIN2          ; get byte
1485  F000:F26B  02 D0                ADD     DL,AL           ; add into total
1486  F000:F26D  E2 F9                LOOP    $1              ; (allow to overflow)
1487  F000:F26F  80 FA 00             CMP     DL, 0           ; is checksum correct ?
1488  F000:F272  74 07                JE      $2
1489  F000:F274  58                   POP     AX              ; this return address
1490  F000:F275  58                   POP     AX              ; main routine address
1491  F000:F276  B0 04                MOV     AL, 04          ; checksum error
1492  F000:F278  E9 FC FA             JMP     EXTERR
1493  F000:F27B  C3           $2:     RET
1494
1495  F000:F27C  E8 DD FF     DWN0:   CALL    CHKCHK
1496  F000:F27F  8B 36 52 01          MOV     SI, [CMDPR1]    ; get aaaa
```

```
1497  F000:F283  06                      PUSH    ES
1498  F000:F284  A1 44 01                MOV     AX, [CURSEG]    ; get current segment
1499  F000:F287  8E C0                   MOV     ES, AX
1500  F000:F289  8B 0E 50 01             MOV     CX, [CMDLTR]    ; number of data bytes
1501  F000:F28D  BB 6E 01                MOV     BX, INLNE + 9   ; start of data
1502  F000:F290  E8 F1 F8          $1:   CALL    TOBIN2          ; get byte
1503  F000:F293  26 88 04                MOV     ES:[SI], AL     ; store it
1504  F000:F296  26 8A 24                MOV     AH, ES:[SI]
1505  F000:F299  3A E0                   CMP     AH, AL          ; was it stored ok ?
1506  F000:F29B  75 08                   JNE     $2
1507  F000:F29D  89 36 46 01             MOV     [CURADR], SI    ; update current address
1508  F000:F2A1  46                      INC     SI
1509  F000:F2A2  E2 EC                   LOOP    $1
1510  F000:F2A4  07                      POP     ES
1511  F000:F2A5  E9 DC FA                JMP     NULLNE
1512
1513  F000:F2A8  8C 06 52 01       $2:    MOV     [CMDPR1], ES    ; save segment of failure
1514  F000:F2AC  89 36 54 01             MOV     [CMDPR2], SI    ; save address of failure
1515  F000:F2B0  07                      POP     ES
1516  F000:F2B1  58                      POP     AX              ; main routine address
1517  F000:F2B2  E9 DF FA                JMP     WRFAIL          ; write fail error
1518
1519  F000:F2B5  E8 A4 FF          DWN1: CALL    CHKCHK
1520  F000:F2B8  E9 C9 FA                JMP     NULLNE          ; no further action end record
1521
1522  F000:F2BB  E8 9E FF          DWN2: CALL    CHKCHK
1523  F000:F2BE  A1 56 01          DWN22: MOV    AX, [CMDPR3]    ; get xxxx
1524  F000:F2C1  A3 3E 01                MOV     [USRCS], AX     ; update user CS
1525  F000:F2C4  A3 44 01                MOV     [CURSEG], AX    ; update current segment address
1526  F000:F2C7  E9 BA FA                JMP     NULLNE
1527
1528  F000:F2CA  E8 8F FF          DWN3: CALL    CHKCHK
1529  F000:F2CD  A1 58 01                MOV     AX, [CMDPR4]    ; get yyyy
1530  F000:F2D0  A3 40 01                MOV     [USRIP], AX     ; update user IP
1531  F000:F2D3  A3 46 01                MOV     [CURADR], AX    ; update current segment address
1532  F000:F2D6  EB E6                   JMP     SHORT DWN22
1533
1534  F000:F2D8  B8 00 00          V13A: MOV     AX, RAMSEG      ; V1.3 patches
1535  F000:F2DB  8E C0                   MOV     ES, AX          ; access to base addresses
1536  F000:F2DD  8E D0                   MOV     SS, AX          ; and stack
1537  F000:F2DF  E9 30 F7                JMP     V13AR           ; end of patch V13A
1538  F000:F2E2  A1 54 01          V13B: MOV     AX, [CMDPR2]    ; special case
1539  F000:F2E5  3D 00 00                CMP     AX, 0           ; test for overflow
1540  F000:F2E8  75 03                   JNE     $1
1541  F000:F2EA  E9 9A FA                JMP     ENDLNE
1542  F000:F2ED  A1 56 01          $1:   MOV     AX, [CMDPR3]
1543  F000:F2F0  E9 19 FF                JMP     V13BR           ; end of patch V13B
1544  F000:F2F3                          BLKB    FFC8h - $, FFh  ; ensure full of FF's
1545
1546                               ; ----- COPYRIGHT NOTICE ----------------------------------
1547  F000:FFC6                          ORG     F000:FFC6h
1548  F000:FFC6  43 43 6F 6F 70 70       DB      'CCooppyyrriigghhtt  '  ; copyright message
1549  F000:FFDA  46 46 6C 6C 69 69       DB      'FFlliigghhtt  11999900';in both EPROMs
1550
1551                               ; ----- RESET VECTOR and BEYOND -----------------------------
1552  F000:FFF0  EA 36 E8 00 F0    RESET: JMP    FAR COLD        ; same as FFFF:0000h
1553  F000:FFF5  00C0              ROM1:  DW     C000h           ; ROM start address (2764)
1554  F000:FFF7  46 4C 49 47 48 54 FLIGHT: DB    'FLIGHT'        ; our name
1555  F000:FFFD  32 30             VERS:  DB     '20'            ; version number
1556  F000:FFFF  81                CHKSUM: DB    81h             ; ROM checksum (2764)
```

MONITOR SOURCE CODE SYMBOL TABLE

Symbol	Defined	Value
BAUDRT	126	0000:014E
BPADR	139	0000:015F
BPCODE	137	0000:015C
BPFLAG	136	0000:0158
BPOINT	328	F000:E9A5
BPRECH	134	0000:015A
BPSEG	138	0000:015D
BREAK	1288	F000:F0BA
BRK0	1301	F000:F0D3
BRK0E	1318	F000:F0FA
BRK1	1338	F000:F11E
BRK2	1340	F000:F123
BRK3	1327	F000:F10A
BRKS	1341	F000:F126
BRKSE	1359	F000:F157
CHKCHK	1480	F000:F25C
CHKSUM	1556	F000:FFFF
CLRBRK	1364	F000:F15E
CMDLEN	834	F000:ED14
CMDLTR	128	0000:0150
CMDPR1	130	0000:0152
CMDPR2	131	0000:0154
CMDPR3	132	0000:0156
CMDPR4	133	0000:0158
CMDSFX	129	0000:0151
CMDTBL	795	F000:ECFA
CODE	Pre	F000:E800
COLD	210	F000:E836
COMND	921	F000:EDBD
CTC0	34	= 0000:0008
CTC1	35	= 0000:000A
CTC2	36	= 0000:000C
CTCK	37	= 0000:000E
CURADR	119	0000:0146
CURSEG	118	0000:0144
DATA	Pre	0000:0000
DNLOAD	1454	F000:F222
DOEXP	173	F000:E000
DUMMY	155	0000:0400
DWN0	1495	F000:F27C
DWN1	1519	F000:F2B5
DWN2	1522	F000:F2BB
DWN22	1523	F000:F2BE
DWN3	1528	F000:F2CA
ENDLNE	894	F000:ED87
ESCAPE	950	F000:EDF9
ESCCHR	122	0000:014A
EXPER	177	F000:E007
EXRT	438	F000:EAD1
EXTERR	887	F000:ED77
EXTFLG	144	0000:0164
EXTRA	Pre	0000:0000
FILL	1168	F000:EFCA
FILL0	1178	F000:EFDD
FILL1	1180	F000:EFE2
FILLB	1201	F000:F015
FILLE	1212	F000:F02C
FILLF	1208	F000:F025
FILLS	1181	F000:EFE5
FILLW	1193	F000:F007
FLB	1202	F000:F018
FLIGHT	1554	F000:FFF7
GETCMD	838	F000:ED15
GETCOM	151	0000:0265
GETPR0	845	F000:ED22
GETPR1	849	F000:ED2A
GETX	851	F000:ED30
GO	1263	F000:F083
GO1	1274	F000:F098
GO2	1276	F000:F09D
GOS	1277	F000:F0A0
GOTOIT	444	F000:EAD4
GOTT	1283	F000:F0B2
INCHR	683	F000:EC4C
INIT	653	F000:EC14
INLINE	701	F000:EC60
INLNE	149	0000:0165
INP0	1230	F000:F04B
INP1	1235	F000:F055
INPORT	1219	F000:F035
INSTAT	676	F000:EC47
INT0	55	0000:0000
INT1	57	0000:0004
INT2	61	0000:0008
INT3	67	0000:000C
INT32	88	0000:0080
INT33	89	0000:0084
INT34	90	0000:0088
INT35	91	0000:008C
INT36	92	0000:0090
INT37	93	0000:0094
INT38	94	0000:0098
INT39	95	0000:009C
INT4	71	0000:0010
INT5	73	0000:0014
INT6	76	0000:0018
INT7	79	0000:001C
IRQ0	491	F000:EB30
IRQ1	492	F000:EB31
IRQ2	493	F000:EB32
IRQ3	494	F000:EB33
IRQ4	495	F000:EB34
IRQ5	496	F000:EB35
IRQ6	497	F000:EB36
IRQ7	498	F000:EB37
LSNIB	521	F000:EB54
ME	187	F000:E802
MEM0	1096	F000:EF17
MEM1	1098	F000:EF1C
MEM2	1110	F000:EF3A
MEM2E	1125	F000:EF5D
MEM34	1128	F000:EF61
MEM34S	1134	F000:EF74
MEM5	1139	F000:EF7F
MEM6	1141	F000:EF84
MEMD	1142	F000:EF87
MEMORY	1077	F000:EEF2
MEMQ	1099	F000:EF1F
MEMS	1144	F000:EF93
MEMWT	1160	F000:EFBD

Symbol	Defined	Value	Symbol	Defined	Value
MEMX	1158	F000:EFBA	SRTDEL	779	F000:ECE2
MLOOP	314	F000:E990	SSEXIT	417	F000:EA95
MSNIB	516	F000:EB4E	SSTEP	319	F000:E996
NMIDEF	480	F000:EB21	STACK	Pre	0000:0000
NMIONE	192	F000:E80A	STDUMY	140	0000:0161
NMITWO	202	F000:E827	STFLAG	142	0000:0162
NOTAVL	883	F000:ED71	STSFX	143	0000:0163
NOTESC	927	F000:EDCC	SUART	647	F000:EC0C
NOTEX	931	F000:EDD4	TEST	959	F000:EE01
NOTVAL	886	F000:ED75	TOASC2	530	F000:EB62
NULLNE	893	F000:ED84	TOASC4	537	F000:EB60
OFBPEX	402	F000:EA72	TOBIN	544	F000:EB78
OKCMD	922	F000:EDC0	TOBIN2	552	F000:EB84
OTCHR	736	F000:EC93	TOBIN4	567	F000:EB95
OTDEL	783	F000:ECE8	TOCAPS	502	F000:EB38
OTLINE	757	F000:ECB8	TOLWR	509	F000:EB43
OTPORT	1242	F000:F05F	TOMON	349	F000:E9C9
OTSTAT	729	F000:EC8E	TOPSTK	154	= 0000:0400
OUTIT	748	F000:ECAA	TPROD	1023	F000:EE8A
OUTLNE	150	0000:01E5	TRAM	972	F000:EE1A
OUTP0	1253	F000:F075	TROM	986	F000:EE3E
OUTP1	1257	F000:F07C	TSPED	1008	F000:EE6A
PIC1	38	= 0000:0010	TXCHR	121	0000:0149
PIC2	39	= 0000:0012	UARTD	40	= 0000:0018
PPIAA	26	= 0000:0000	UARTS	41	= 0000:001A
PPIAB	27	= 0000:0002	UEXIT	344	F000:E9BD
PPIAC	28	= 0000:0004	UNBPEX	413	F000:EA8E
PPIAK	29	= 0000:0006	UNIT	196	F000:E813
PPIBA	30	= 0000:0001	UP0	1419	F000:F1D6
PPIBB	31	= 0000:0003	UP1	1421	F000:F1DB
PPIBC	32	= 0000:0005	UPLOAD	1410	F000:F1C5
PPIBK	33	= 0000:0007	UPMR	1431	F000:F1F8
RAMLST	125	0000:014C	UPS	1422	F000:F1DE
RAMSEG	47	= 0000:0000	UPWT	1443	F000:F215
RAMTST	591	F000:EBBB	USER	186	F000:E800
REG0	1039	F000:EE9F	USR0	49	= 0000:0050
REG12	1045	F000:EEAE	USRAX	104	0000:0128
REGE	1072	F000:EEEE	USRBP	109	0000:0132
REGIS	1030	F000:EE8E	USRBX	105	0000:012A
REGR	1063	F000:EED9	USRCS	115	0000:013E
RESET	1552	F000:FFF0	USRCX	106	0000:012C
ROM1	1553	F000:FFF5	USRDI	111	0000:0136
ROMSEG	48	= 0000:F000	USRDS	112	0000:0138
ROMSUM	124	0000:014B	USRDX	107	0000:012E
ROMTST	577	F000:EBA2	USRERR	902	F000:ED96
RSTLNE	772	F000:ECD4	USRES	113	0000:013A
RXCHR	120	0000:0148	USRFG	117	0000:0142
SBAUD	666	F000:EC32	USRIP	116	0000:0140
SEREND	624	= F000:EBE9	USRSI	110	0000:0134
SERIAL	626	F000:EBE9	USRSP	108	0000:0130
SERTBL	611	F000:EBD7	USRSS	114	0000:013C
SINGLE	1381	F000:F183	USRSTK	158	= 0000:0500
SNDBYT	869	F000:ED5A	V13A	1534	F000:F2D8
SNDDTA	859	F000:ED43	V13AR	377	F000:EA12
SNDERR	912	F000:EDB0	V13B	1538	F000:F2E2
SNDWRD	875	F000:ED64	V13BR	1439	F000:F20C
SNG0	1394	F000:F19D	VERS	1555	F000:FFFD
SNG1	1396	F000:F1A2	WARM	239	F000:E86B
SNGS	1397	F000:F1A5	WRESET	954	F000:EDFC
SNOR	1401	F000:F1B1	WRFAIL	901	F000:ED94

FLIGHT86.MSG FILE

To give a degree of flexibility the file FLIGHT86.MSG may be edited allowing users to modify messages and even command letters. If using MSDOS, this file MUST be in the current active directory. If using CP/M the file must be in the default user number area.

FLIGHT86.MSG is an ASCII file that may be edited with most word processors using their 'program' or 'non-document' mode; an extract is listed overleaf. Extreme care must be taken that the position of messages is not changed.

Before starting to make changes be sure you have a printed hard copy and a BACKUP copy of the file. If translating, replace each phrase by a phrase of the same meaning. Be careful to preserve the positions of columns and the meanings of rows. Change only ONE item at a time and test the software thoroughly after each change.

Lines commencing with { are treated as comment lines and are ignored, you may include as many of these as you wish to assist the reader. Sections MUST be terminated by a line starting with {#.

The COMMANDS - Section One

Each command uses 2 lines. DO NOT change their ORDER.

The first line is used for the command meaning followed by the command letter in column 18 (this MUST be maintained). An explanation of the command syntax (not to exceed 40 characters) should follow, starting in column 20.

The second line is up to 79 characters of help for use with the Help command.

The OPTIONS - Section Two

The OPTION letters are used in the command tails. DO NOT change their ORDER.

Ensure your option letter cannot be mistaken for a single digit Hex address. (This can give unexpected results)

The final option is the response expected for a Yes/No question. You may use this to give 2 choices for an exit instead of the upper/lower case choice.

ERROR & SYSTEM Messages - Section Three

These are the messages displayed as a result of an error condition or as instructions to the user. DO NOT change their ORDER.

The maximum length of any message is 79 characters. Some messages are joined together so test your translation thoroughly. This is not easy with error messages unless you can 'force' the error to occur.

Serial Port - Section Four

This is the communications port to be used if using an MSDOS machine.

Make sure your printer is NOT redirected to the serial port to be used as this can produce unexpected results. It is quite acceptable to use the printer LST redirected to COM1 and then enter COM2 for the communications port.

File Extension - Section Five

Intel Hex files saved to disk are given the extension specified here. It must be one to three characters. We urge you NOT to use HEX as you may accidentally overwrite a file created by the assembler. Similarly don't use an extension that occurs regularly on the disk (ie COM, EXE, ASM etc) for the same reason.

Structure of FLIGHT86.MSG file

This is an extract from the FLIGHT86.MSG file to indicate the style of entry expected, the important aspects have been highlighted. Please take a print out of the file for the complete story.

```
{FLIGHT86.MSG
{ some messages and instructions here
{ --- COMMANDS ----------------------------------------
{ some messages and instructions here
Escape          Esc
Stops current command (except GO)
{ some messages and instructions here
Reset           X
'Warm' reset of controller board, leaving user registers intact
{ some messages and instructions here
  . more commands here
{ some messages and instructions here
Quit            Q
Terminates host software, return to operating system
{
{# -- End of COMMANDS -------Start of OPTIONS --------------------
{ some messages and instructions here
{ note about option letter
M
  . more option letters
{ note about option letter
Y
y
{# -- End of OPTIONS ---------- Start of ERROR MESSAGES ------------
{ some messages and instructions here a list of messages follows
Invalid Command
No RAM found ! !
  . many more messages here
No Breakpoint Set
{# -- End of ERROR MESSAGES -- Start of SYSTEM MESSAGES ------------
{ some messages and instructions here
RAM found at
    Block(s) of RAM found at
  . many more messages here
No ROM found ! !
{# -- End of SYSTEM MESSAGES -- Start of SERIAL PORT ---------------
{ some messages and instructions here
COM1
{# -- End of SERIAL PORT ----- Start of DEFAULT FILE EXTENSION -----
{ some messages and instructions here
HX
{# -- End of message file ----------------------------------------
```

CHIP DATA

The following pages give a precis of the data for the major chips used by the FLIGHT 86 controller board. This means that certain modes of operation are not described because they are not applicable to this board. We suggest you obtain the chip data sheets from the manufacturer if you need more detailed information. Much of the information you will need, for most applications, is extracted from these data sheets and presented here for easier assimilation.

Each chip description follows a similar format. First there is a block diagram of the chip alongside a physical pin layout diagram.

The manufacturers maximum current requirement for the chip is stated, this allows an estimate of the maximum current required by the board to be made.

Any programmable registers accessible are listed with the types of activity allowed and their Port addresses on the FLIGHT 86 controller board.

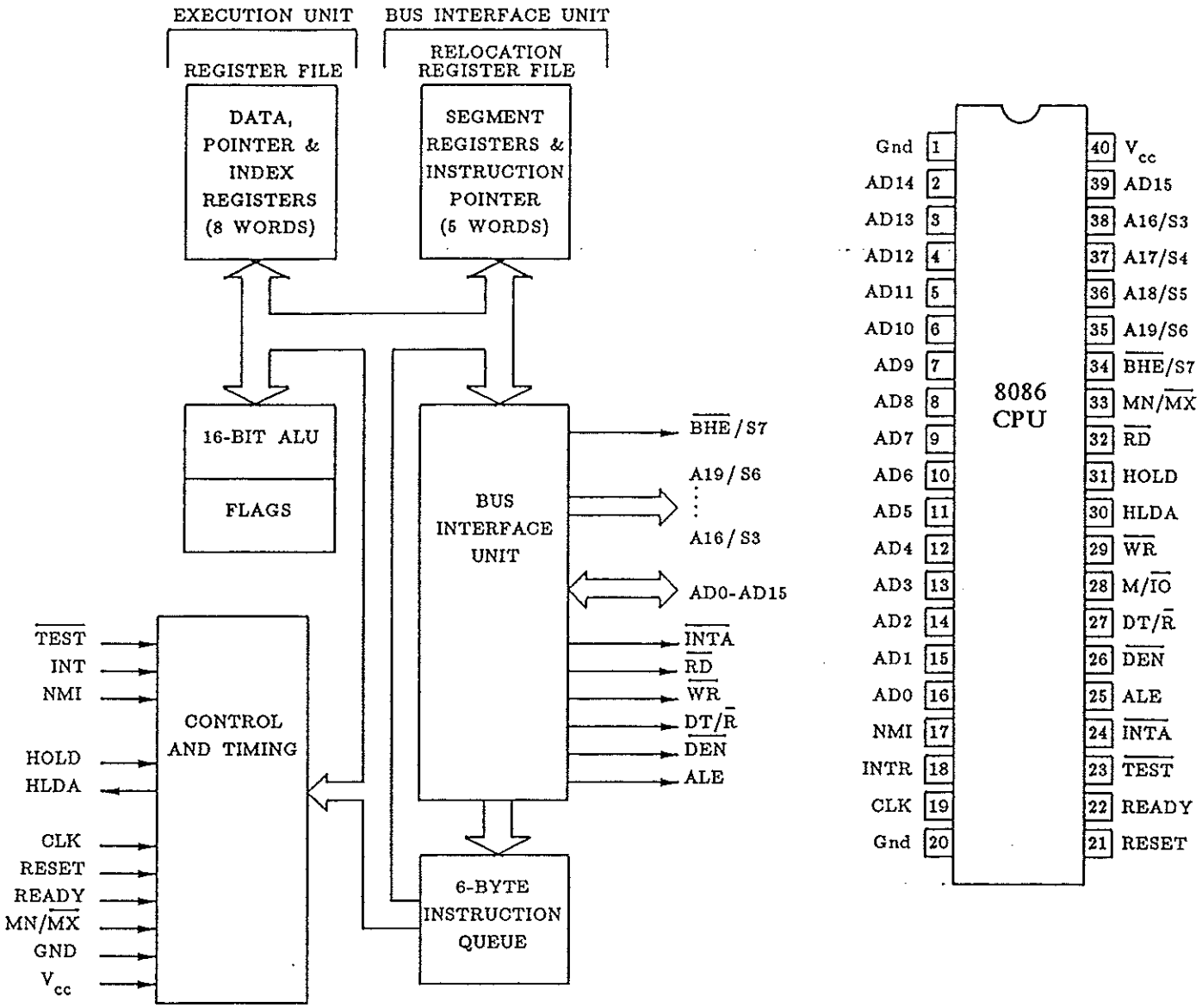
The function and operation of each pin on the chip then follows.

If appropriate the general operation of the chip is described together with any programming information.

At the end of this section, the pin out diagrams of all the other chips used on the board is supplied.

Our Experiment Book makes use of these major chips and should be studied for a deeper insight into their operation. Our Fault Finding book makes reference to some of these devices, particularly from the hardware point of view.

8086 CPU (MINIMUM MODE)



Maximum current requirement - 340 mA

8086 Pin description (minimum mode)

AD0 - AD15 Address/Data Bus (input and output).
The low 16 multiplexed address and data lines. They are tri-state during an interrupt acknowledge or hold.

A16/S3 - A19/S6 High Address Bus (output)
High 4 multiplexed address and status lines. Tri-state during a hold.

$\overline{BHE}$/S7 Bus High Enable (output)
Multiplexed with a status line. It is used, with A0, to condition the chip select functions and to enable data onto the odd half of the data bus (D8 - D15). It is tri-state during a hold.

ALE Address Latch Enable (output).
Used to latch the address information from the Address/data bus. It is pulsed high during T1 of any bus cycle. It is never tri-stated.

$\overline{RD}$ Read (output).
Strobed low when the 8086 is performing a memory or I/O read cycle. Active low during the T2, T3 and Tw cycles. It is tri-state during a hold.

$\overline{WR}$ Write (output).
Strobed low when the 8086 is performing a memory or I/O write cycle. Active low during the T2, T3 and Tw cycles. It is tri-state during a hold.

M/$\overline{IO}$ Memory Input/Output Status (output).
High during a memory access, low during an I/O. Tri-state during a hold.
READY Ready (input).
Acknowledgement from the addressed device that it has completed the data transfer. Synchronised by the 8284A CGD.
HOLD Hold Request (input).
If another device requires access to the 8086 bus it takes this line high. The 8086 issues a HLDA in response.
HLDA Hold Acknowledge (output).
After the 8086 receives a hold request it takes the HLDA line high during the T1 clock cycle and will tri-state the buses and most control lines.
RESET Reset (input).
A high forces the 8086 to abandon the current activity, and causes the buses and most output control lines to go tri-state.
When returned low the 8086 starts execution from FFFF:0000.
INTR Interrupt request (input).
This line is sampled during the last clock cycle of each instruction. If it is high the 8086 issues INTA pulses to enable the interrupting device to place a vector address on the bus. The action may be masked by software.
$\overline{INTA}$ Interrupt Acknowledge (output).
Used as a read strobe in response to an interrupt request. Strobed low during the T2, T3 and Tw cycles.
NMI Non-maskable Interrupt (input).
A low to high transition (release of the NMI button) causes an INT2 instruction to be executed. The action cannot be masked by software.
CLK Clock (input).
Provides basic timing signal, ideally with a 33% duty cycle. The frequency must not fall below 2 MHz.
Vcc +5 V power supply.
Gnd 0 V power supply return
MN/$\overline{MX}$ Minimum/Maximum mode control (input). Wired high on FLIGHT 86 board.
When high the 8086 operates in the minimum mode.
$\overline{TEST}$ Test (input). Wired low on FLIGHT 86 board.
When high the 8086 idles. It may be examined by software.
DT/$\overline{R}$ Data Bus Transmit/Receive (output). Not used on FLIGHT 86 board.
Used to control direction of a bus transceiver. Tri-state during a hold.
$\overline{DEN}$ Data Enable (output). Not used on FLIGHT 86 board.
Used to enable a bus transceiver. Tri-state during a hold.

General Operation

The 8086 is split logically into two processing units, the BIU (Bus Interface Unit) and the EU (Execution Unit) as shown on the block diagram, page 72. They can, and do, interact but for the most part are separate processors.

The BIU handles the instruction fetch operations, utilising a 6-byte FIFO (First In First Out) buffer instruction queue to allow instructions to be pre-fetched while the EU is processing previous instructions.

Whenever there are two bytes of space in the queue, the BIU will attempt a fetch memory cycle. This reduces the 'dead-time' on the system bus. The EU extracts the instructions from the queue as required.

The queue is flushed whenever a branch instruction is executed, the first instruction to enter the queue will be available to the EU.

Memory operands are passed through the BIU, when required, for processing by the EU, and the results are returned to the BIU for storage.

The BIU also handles the basic bus control signals.

Bus Operation

Refer to the timing waveforms opposite when reading these notes. $\overline{DEN}$ and DT/$\overline{R}$ are shown for completeness, they are not used on the EIOLP board. Each processor bus cycle consists of 4 clock cycles, T1, T2, T3 and T4. The 8086 outputs the address during T1 and data transfer occurs during T3 and T4.

If the READY line is low then WAIT states will be inserted between T3 and T4. You cannot insert WAIT, Tw, states on the EIOLP board because the RDY lines of the 8284A are hard wired.

T1 or 'Idle' states, occur during 8086 processing, when the queue is full and no memory operands are required.

Each cycle starts in T1 with the ALE (Address Latch Enable) signal from the 8086. The falling edge of the ALE is used to latch the multiplexed bus signals AD0 to AD15, A16/S3 to A19/S6 and $\overline{BHE}$/S7 with a set of address latches to provide the lines A0 to A19 and BHE. See the circuit on page 12.

During the four clock cycles the M/$\overline{IO}$ line indicates; '1' for a memory or '0' for an I/O operation.

The DT/$\overline{R}$ and $\overline{DEN}$ lines are normally provided for external bus drivers

Read Operation

M/$\overline{IO}$ will be high. During T2 the address is removed from the multiplexed address/data bus, which goes tri-state. The $\overline{RD}$ line goes low, after allowing time for the bus to float, causing the addressed device to place its data on the bus some time later (depending on the device access time). The 8086 will read the data from the bus prior to raising the $\overline{RD}$ line again.

Write Operation

M/$\overline{IO}$ will be high. During T2 the address is removed from the multiplexed address/data bus, and the 8086 places the data to be written to the addressed device on the bus. The data is valid until the end of T4 when the bus is tristated. The $\overline{WR}$ line goes low at the start of T2, rather earlier than the $\overline{RD}$ line, giving the addressed device as much time as possible to read the data.

I/O Operations

The timing of I/O operations is the same as memory read/write operations, except M/$\overline{IO}$ will be low.

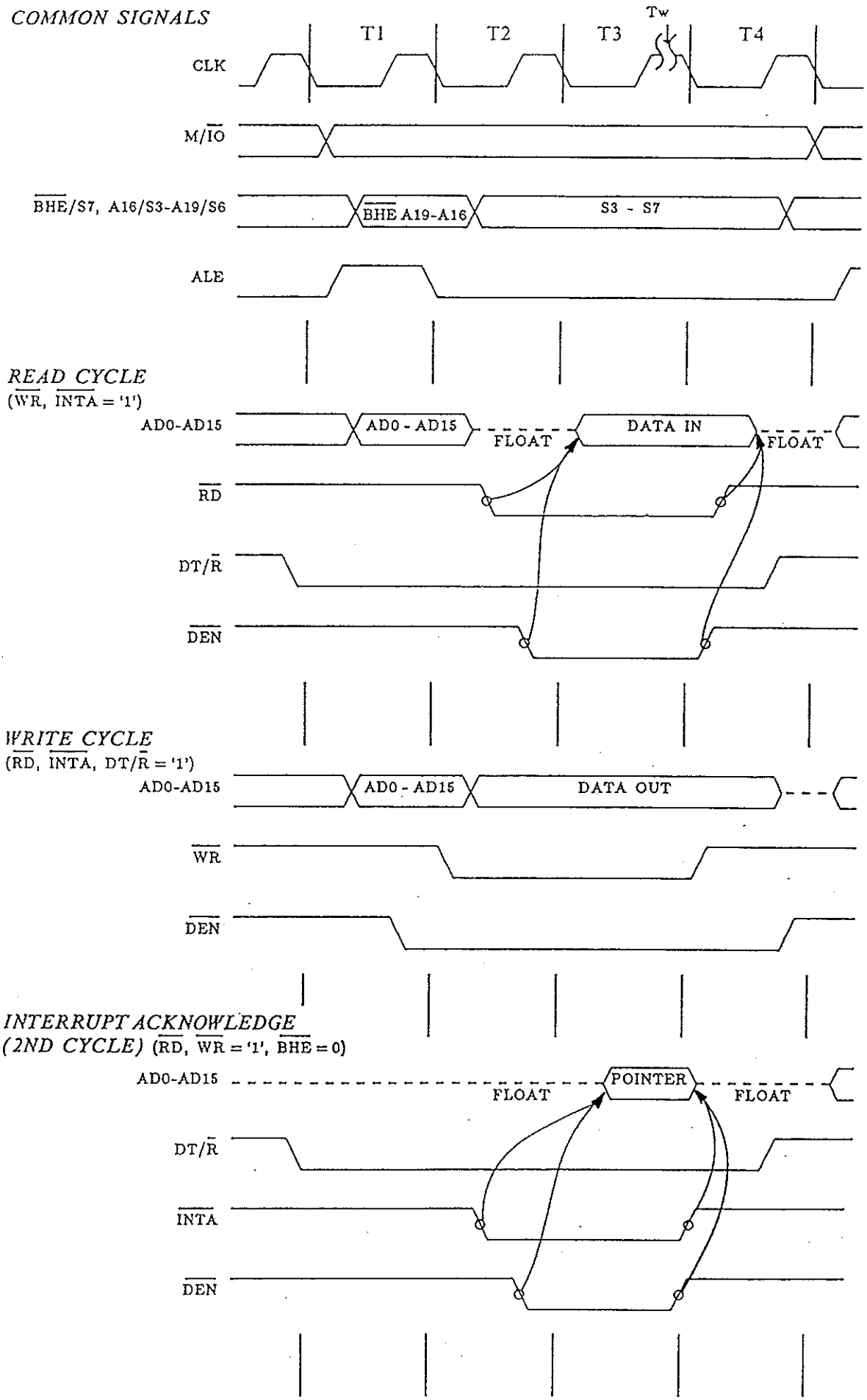
Status Lines

The status lines are used to indicate internal events to the outside world. These lines are valid during the T2 to T4 time. S3 and S4 indicate the segment used in forming the address, as shown below:

S3	S4	
0	0	Extra segment
0	1	Stack segment
1	0	Code segment
1	1	Data segment

S5 indicates the state of the flag register interrupt enable bit.
S6 is always 0
S7 is a spare status bit.

8086 TIMING WAVEFORMS (MINIMUM MODE)



Reset & Initialisation

The RESET signal is provided by the 8284A on the FLIGHT 86 controller board. The 8284A ensures the RESET is high for the required minimum of four clock cycles. Whilst the RESET line is high the 8086 is 'dormant' with most of the data and control lines in the tri-state condition. A low on the RESET line for a minimum of 10 clock cycles, causes the 8086 to start execution from address FFFF0 (FFFF:0000).

Interrupts

The 8086 handles both software and hardware interrupts. Both use a look-up table of interrupt types, in RAM, of start addresses for the appropriate ISR (Interrupt Service Routine). See page 36 for the assignments available on the FLIGHT 86 board. Each element in the table is 4 bytes and comprises of the Offset and then the Segment address of the ISR. These must be initialised BEFORE use. The software interrupts are the INT N instructions where N is the type number.

All interrupts result in the current Code segment, Instruction Pointer and the flag register being PUSHED onto the stack. The Interrupt Enable Flag is then reset to disable further non-maskable interrupts from occurring. The IRET instruction POPs the flag conditions PRIOR to the interrupt, before returning from the ISR.

Non-Maskable Interrupt Operation

A low to high transition on this input, sustained for at least 2 clock cycles, is latched in the 8086. This will cause an INT 2 instruction to be executed, after the current instruction has been completed. The subsequent high to low transition may occur at any time, although it is important that this signal is free from 'glitches' or 'bounce'. The FLIGHT 86 board has used a Schmitt buffer on the NMI button to provide a bounce free signal. See the circuit on page 12.

The FLIGHT 86 software has three NMI routines, built into the monitor ROM. These are documented in the pseudocode listing, on page 39.

Maskable Interrupt Operations

The INTR line is level sensitive and is sampled during T4. If it is high the 8086 issues two $\overline{\text{INTA}}$ pulses. During the first pulse the multiplexed address/data bus floats and this warns the interrupting device to be ready for the second $\overline{\text{INTA}}$ pulse. The second pulse is treated as the $\overline{\text{RD}}$ line, and the interrupting device places an interrupt type on the bus. The 8086 multiplies this type number by 4 and uses this as the address of the pointer to the interrupt service routine. The second pulse timing is shown on the timing diagrams of page 75.

On-board interrupts are handled by the 8259A, PIC.

HALT

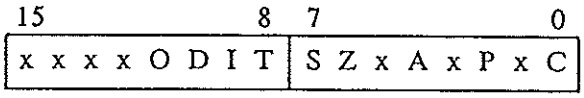
When a software HALT instruction has been executed, the processor issues one ALE signal WITHOUT the qualifying control line signals. Only an external interrupt request signal or a RESET will bring the 8086 out of this condition.

8086 Register Summary

The 8086 has fourteen 16-bit registers accessible to the programmer; 4 of these may also be used as 8-bit registers.

	16-bits		8-bits
Accumulator	AX	or	AH and AL
Base	BX	or	BH and BL
Counter	CX	or	CH and CL
Data	DX	or	DH and DL
Stack Pointer	SP	(uses SS register)	
Base Pointer	BP	(uses SS register)	
Source Index	SI	(uses ES register)	
Destination Index	DI	(uses DS register)	
Code Segment	CS		
Data Segment	DS		
Stack Segment	SS		
Extra Segment	ES		
Instruction Pointer	IP	(uses CS register)	

Status Flags



Hi flag : Lo flag (=8085)

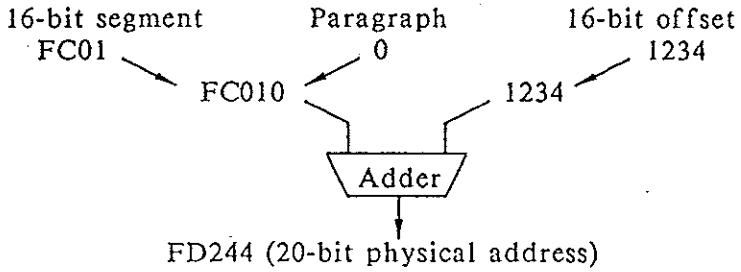
Flag definitions:

x	Don't care	
O	Overflow	Set if signed result too large for number of bits allowed
D	Direction	If set string instructions auto decrement else increment
I	Interrupt Enable	If set allows maskable interrupts to occur
T	Trap	If set CPU is in single step mode
S	Sign	Made equal to high order bit of result
Z	Zero	Set if result is zero
A	Auxiliary carry	Set on bit 3, low nibble, if BCD carry or borrow
P	Parity	Set if low 8-bits contain an even number of ones
C	Carry	Set on highest bit (7 or 15) carry or borrow

Physical Address Calculation

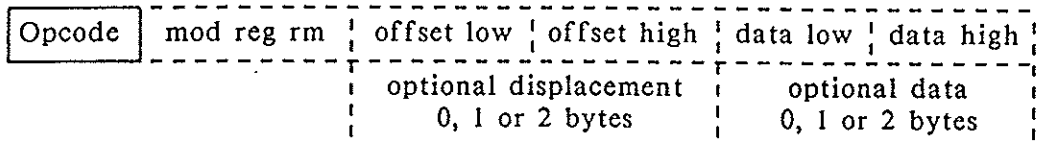
The 8086 has 20 address lines, with an addressing range of 00000 to FFFFF, giving access to 1 Mbyte of memory. The physical 20-bit address is calculated from the 16-bit segment address and the 16-bit logical offset. The segment address is that contained in the segment register being used and is always on a 16-bit paragraph boundary (ie. a 0 is tacked on the end).

The diagram below may help show how the segment and offset are combined; it assumes the segment is FC01 and the offset is 1234:



8086 Operand Summary

The 8086 software instructions are formed by a sequential series of one to six bytes, shown in general terms:



The instruction always starts with an opcode, these are listed in the matrix opposite and in more detail on the following pages. The 8-bit **mod reg rm** field, if used, specifies the operands of the instruction. This is followed by any displacement bytes (0, 1 or 2) finally any data bytes (0, 1 or 2).

A single byte segment register override prefix may be added to the instruction to specify which segment is to be used. This byte occurs before the opcode.

The **mod reg rm** field is defined as:

mod (mode)		**reg** (register)			**rm** (register or memory)		
D7	D6	D5	D4	D3	D2	D1	D0

The mod and rm fields determine the register and displacement values as shown in the table below (EA times in parentheses):

	MEMORY ACCESS						REGISTER ACCESS	
mod value	00		01		10		11	
rm value							word	byte
000	[BX+SI]	(7)	[BX+SI+disp8]	(11)	[BX+SI+disp16]	(11)	AX	AL
001	[BX+DI]	(8)	[BX+DI+disp8]	(12)	[BX+DI+disp16]	(12)	CX	CL
010	[BP+SI]	(8)	[BP+SI+disp8]	(12)	[BP+SI+disp16]	(12)	DX	DL
011	[BP+DI]	(7)	[BP+DI+disp8]	(11)	[BP+DI+disp16]	(11)	BX	BL
100	[SI]	(5)	[SI+disp8]	(9)	[SI+disp16]	(9)	SP	AH
101	[DI]	(5)	[DI+disp8]	(9)	[DI+disp16]	(9)	BP	CH
110	direct	(6)	[BP+disp8]	(9)	[BP+disp16]	(9)	SI	DH
111	[BX]	(5)	[BX+disp8]	(9)	[BX+disp16]	(9)	DI	BH

Notes:

mod 00	if rm = 110 then the displacement is a direct memory address
mod 01	disp8, 00 to FF, is a signed 2's complement number (-80 to +7F)
mod 10	disp16, 0000 to FFFF, is also a 2's complement (-8000 to +7FFF)
	If 0 is used some assemblers treat this as mod 00 while others insert 00 or 0000 as the displacement value.

Effective Address (EA) calculations

In the Instruction set summary starting on page 80 many of the instruction timings include an EA time. This depends on the type of instruction and the calculation should include the following considerations:

Add 2 cycles for segment override prefix.
Add 6 cycles for a displacement only
Other times are shown in the table above in parentheses.

8086 INSTRUCTION SET MATRIX

HI \ LO	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	ADD b.f.rm	ADD w.f.rm	ADD b.t.rm	ADD w.t.rm	ADD b.i.a	ADD w.i.a	PUSH ES	POP ES	OR b.f.rm	OR w.f.rm	OR b.t.rm	OR w.t.rm	OR b.i.a	OR w.i.a	PUSH CS	POP CS
1	ADC b.f.rm	ADC w.f.rm	ADC b.t.rm	ADC w.t.rm	ADC b.i.a	ADC w.i.a	PUSH SS	POP SS	SBB b.f.rm	SBB w.f.rm	SBB b.t.rm	SBB w.t.rm	SBB b.i.a	SBB w.i.a	PUSH DS	POP DS
2	AND b.f.rm	AND w.f.rm	AND b.t.rm	AND w.t.rm	AND b.i.a	AND w.i.a	ES so	DAA	SUB b.f.rm	SUB w.f.rm	SUB b.t.rm	SUB w.t.rm	SUB b.i.a	SUB w.i.a	CS so	DAS
3	XOR b.f.rm	XOR w.f.rm	XOR b.t.rm	XOR w.t.rm	XOR b.i.a	XOR w.i.a	SS so	AAA	CMP b.f.rm	CMP w.f.rm	CMP b.t.rm	CMP w.t.rm	CMP b.i.a	CMP w.i.a	DS so	AAS
4	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI
5	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI
6																
7	JO	JNO	JB JNAE JC	JNB JAE JNC	JE JZ	JNE JNZ	JBE JNA	JNBE JA	JS	JNS	JP JPE	JNP JPO	JL JNGE	JNL JGE	JLE JNG	JNLE JG
8	Immd b.rm *	Immd w.rm *	Immd b.id.rm *	Immd w.id.rm *	TEST b.rm	TEST w.rm	XCHG b.rm	XCHG w.rm	MOV b.f.rm	MOV w.f.rm	MOV b.t.rm	MOV w.t.rm	MOV sr.f.rm	LEA	MOV sr.t.rm	POP rm
9	NOP	XCHG CX	XCHG DX	XCHG BX	XCHG SP	XCHG BP	XCHG SI	XCHG DI	CBW	CWD	CALL l.d	WAIT	PUSHF	POPF	SAHF	LAHF
A	MOV m→AL	MOV m→AX	MOV AL→m	MOV AX→m	MOVS b	MOVS w	CMPS b	CMPS w	TEST b.i.a	TEST w.i.a	STOS b	STOS w	LODS b	LODS w	SCAS b	SCAS w
B	MOV i→AL	MOV i→CL	MOV i→DL	MOV i→BL	MOV i→AH	MOV i→CH	MOV i→DH	MOV i→BH	MOV i→AX	MOV i→CX	MOV i→DX	MOV i→BX	MOV i→SP	MOV i→BP	MOV i→SI	MOV i→DI
C			RET n+SP	RET	LES	LDS	MOV b.i.rm	MOV w.i.rm			RET l.n+SP	RET l	INT 3	INT n≠3	INTO	IRET
D	Shift b *	Shift w *	Shift b.v *	Shift w.v *	AAM 0A	AAD 0A		XLAT	ESC 0	ESC 1	ESC 2	ESC 3	ESC 4	ESC 5	ESC 6	ESC 7
E	LOOPNZ LOOPNE si	LOOPZ LOOPE si	LOOP si	JCXZ si	IN b	IN w	OUT b	OUT w	CALL si.d	JMP d	JMP l.d	JMP si.d	IN b.v	IN w.v	OUT b.v	OUT w.v
F	LOCK		REPNZ REPNE	REP REPE REPZ	HLT	CMC	Grp1 b.rm *	Grp1 w.rm *	CLC	STC	CLI	STI	CLD	STD	Grp2 b.rm *	Grp2 w.rm *

Specials	reg value in mod reg rm							
*	000	001	010	011	100	101	110	111
Immd	ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
Shift	ROL	ROR	RCL	RCR	SHL/SAL	SHR		SAR
Grp1	TEST		NOT	NEG	MUL	IMUL	DIV	IDIV
Grp2	INC	DEC	CALL id	CALL l.id	JMP id	JMP l.id	PUSH	

Abbreviations used:

id	indirect	d	direct
i	immediate	r	register
a	accumulator	m	memory
b	byte operation	v	variable
w	word operation	n	number
so	segment override		
t	to CPU register		
sr	segment register		
f	from CPU register		
rm	EA is second byte		
si	short - intrasegment		
l	long - intersegment		
is	immediate byte sign		

8086 Instruction Set Summary - by group

Each 8086 instruction type is listed with a precis of operation, the opcode and operands with approximate clock timings.

DATA TRANSFER GROUP

	Operation	Opcode	Operands					Clocks
MOV move data								
register to register	reg8 ← reg8	8A	mod reg rm					2
	reg16 ← reg16	8B	mod reg rm					2
register to memory	mem8 ← reg8	88	mod reg rm	Lomem	Himem			9+EA
	mem16 ← reg16	89	mod reg rm	Lomem	Himem			9+EA
memory to register	reg8 ← mem8	8A	mod reg rm	Lomem	Himem			8+EA
	reg16 ← mem16	8B	mod reg rm	Lomem	Himem			8+EA
memory to AX/AL	AL ← mem8	A0	Lomem	Himem				10
	AX ← mem16	A1	Lomem	Himem				10
AX/AL to memory	mem8 ← AL	A2	Lomem	Himem				10
	mem16 ← AX	A3	Lomem	Himem				10
Immediate to register	reg8 ← imm8	B0+reg	Data					4
	reg16 ← imm16	B8+reg	Lodata	Hidata				4
Immediate to memory, direct	mem8 ← imm8	C6	mod 000 rm	Lomem	Himem	Data		10+EA
	mem16 ← imm16	C7	mod 000 rm	Lomem	Himem	Lodata	Hidata	10+EA
Immediate to memory, indirect	[reg16] ← imm8	C6	mod 000 rm	Data				10+EA
	[reg16] ← imm16	C7	mod 000 rm	Lodata	Hidata			10+EA
Memory to/from seg register	mem16 ← seg reg	8C	mod seg rm	Lomem	Himem			9+EA
	seg reg ← mem16	8E	mod seg rm	Lomem	Himem			8+EA
Register to/from seg register	reg16 ← seg reg	8C	mod seg rm					2
	seg reg ← reg16	8E	mod seg rm					2
PUSH memory onto the stack	[SP] ← mem16, SP--	FF	mod 110 rm	Lomem	Himem			16+EA
word register	[SP] ← reg16, SP--	50+reg						11
segment register	[SP] ← seg reg, SP--	06+seg						10
PUSHF push flags onto stack	[SP] ← flags, SP--	9C						10
POP memory from stack	mem16 ← [SP], SP++	8F	mod 000 rm	Lomem	Himem			17+EA
word register	reg16 ← [SP], SP++	58+reg						8
segment register	seg reg ← [SP], SP++	07+seg						8
POPF pop flags from stack	flags ← [SP], SP++	9D						8
XCHG exchange								
Register with register	reg8 ⟷ reg8	86	mod reg rm					4
	reg16 ⟷ reg16	87	mod reg rm					4
Word register with AX	AX ⟷ reg16	90+reg						3
Memory with register	mem8 ⟷ reg8	86	mod reg rm	Lomem	Himem			17+EA
	mem16 ⟷ reg16	87	mod reg rm	Lomem	Himem			17+EA
IN input from fixed port	AL ← port8	E4	Port					10
	AX ← port8	E5	Port					10
from port (defined in DX)	AL ← [DX]	EC						8
	AX ← [DX]	ED						8
OUT output to fixed port	port8 ← AL	E6	Port					10
	port8 ← AX	E7	Port					10
to port (defined in DX)	[DX] ← AL	EE						8
	[DX] ← AX	EF						8
XLAT table look up translation	AL ← table entry	D7						11
LDS load pointer to DS	DS:reg16 ← mem32	C5	mod reg rm	Lomem	Himem			16+EA
LEA load effective address	reg16 ← mem16	8D	mod reg rm	Lomem	Himem			2+EA
LES load pointer to ES	ES:reg16 ← mem32	C4	mod reg rm	Lomem	Himem			16+EA
LAHF load AH with flags	AH ← lo flags	9F						4
SAHF store AH into flags	lo flags ← AH	9E						4

ARITHMETIC GROUP

	Operation	Opcode	Operands					Clock
ADD register plus register	reg8 ← reg8+reg8	02	mod reg rm					3
	reg16 ← reg16+reg16	03	mod reg rm					3
memory plus register	mem8 ← mem8+reg8	00	mod reg rm	Lomem	Himem			16+EA
	mem16 ← mem16+reg16	01	mod reg rm	Lomem	Himem			16+EA
register plus memory	reg8 ← reg8+mem8	02	mod reg rm	Lomem	Himem			9+EA
	reg16 ← reg16+mem16	03	mod reg rm	Lomem	Himem			9+EA
AX/AL plus immediate	AL ← AL+imm8	04	Data					4
	AX ← AX+imm16	05	Lodata	Hidata				4
memory plus immediate	byte[BX] ← byte[BX]+imm8	80	mod 000 rm	Lodata				4
(indirect)	word[BX] ← word[BX]+imm8	83	mod 000 rm	Lodata				4
	word[BX] ← word[BX]+imm16	81	mod 000 rm	Lodata	Hidata			4
memory plus immediate	mem8 ← mem8+imm8	80	mod 000 rm	Lomem	Himem	Data		17+EA
(direct)	mem16 ← mem16+imm8	83	mod 000 rm	Lomem	Himem	Data		17+EA
	mem16 ← mem16+imm16	81	mod 000 rm	Lomem	Himem	Lodata	Hidata	17+EA
ADC addition with carry								
Memory/register plus memory/register	same as ADD except +CY; opcode = 1x							
Immediate plus AX/AL	same as ADD except +CY; opcode = 1x							
Immediate plus memory/register	same as ADD except | mod010rm |; +CY; opcode same							
INC increment word register	reg16 ← reg16+1	40+reg						2
byte register	reg8 ← reg8+1	FE	mod 000 rm					3
memory	mem8 ← mem8+1	FE	mod 000 rm	Lomem	Himem			15+EA
	mem16 ← mem16+1	FF	mod 000 rm	Lomem	Himem			15+EA
AAA ascii adjust for addition	adjust AL, flags AH	37						4
DAA decimal adjust for addition	adjust AL, flags AH	27						4
SUB register from register	reg8 ← reg8 - reg8	2A	mod reg rm					3
	reg16 ← reg16 - reg16	2B	mod reg rm					3
register from memory	mem8 ← mem8 - reg8	28	mod reg rm	Lomem	Himem			16+EA
	mem16 ← mem16 - reg16	29	mod reg rm	Lomem	Himem			16+EA
memory from register	reg8 ← reg8 - mem8	2A	mod reg rm	Lomem	Himem			9+EA
	reg16 ← reg16 - mem16	2B	mod reg rm	Lomem	Himem			9+EA
Immediate from AX/AL	AL ← AL - imm8	2C	Data					4
	AX ← AX - imm16	2D	Lodata	Hidata				4
Immediate from memory	byte[BX] ← byte[BX]- imm8	80	mod 101 rm	Data				4
(indirect)	word[BX] ← word[BX]- imm8	83	mod 101 rm	Data				4
	word[BX] ← word[BX]- imm16	81	mod 101 rm	Lodata	Hidata			4
Immediate from memory	mem8 ← mem8 - imm8	80	mod 101 rm	Lomem	Himem	Data		17+EA
(direct)	mem16 ← mem16 - imm8	83	mod 101 rm	Lomem	Himem	Data		17+EA
	mem16 ← mem16 - imm16	81	mod 101 rm	Lomem	Himem	Lodata	Hidata	17+EA
SBB subtraction with borrow								
Memory/register less memory/register	same as SUB except -CY; opcode = 1x							
AX/AL less immediate	same as SUB except -CY; opcode = 1x							
memory/register less immediate	same as SUB except | mod011rm |; -CY; opcode same							
CMP compare 2 operands								
register with register	flags ← reg8 - reg8	3A	mod reg rm					3
	flags ← reg16 - reg16	3B	mod reg rm					3
register with memory	flags ← mem8 - reg8	38	mod reg rm	Lomem	Himem			9+EA
	flags ← mem16 - reg16	39	mod reg rm	Lomem	Himem			9+EA
memory with register	flags ← reg8 - mem8	3A	mod reg rm	Lomem	Himem			9+EA
	flags ← reg16 - mem16	3B	mod reg rm	Lomem	Himem			9+EA
AX/AL with immediate	flags ← AL - imm8	3C	Data					4
	flags ← AX - imm16	3D	Lodata	Hidata				4
memory with immediate	flags ← byte[BX] - imm8	80	mod 111 rm	Lodata				4
(indirect)	flags ← word[BX] - imm8	83	mod 111 rm	Lodata				4
	flags ← word[BX] - imm16	81	mod 111 rm	Lodata	Hidata			4
memory with immediate	flags ← mem8 - imm8	80	mod 111 rm	Lomem	Himem	Data		10+EA
(indirect)	flags ← mem16 - imm8	83	mod 111 rm	Lomem	Himem	Data		10+EA
	flags ← mem16 - imm16	81	mod 111 rm	Lomem	Himem	Lodata	Hidata	10+EA

ARITHMETIC GROUP *continued*

	Operation	Opcode	Operands					Clock
DEC decrement word register	reg16 ← reg16 - 1	48+reg						2
byte register	reg8 ← reg8 - 1	FE	mod 001 rm					3
memory	mem8 ← mem8 - 1	FE	mod 001 rm	Lomem	Himem			15+EA
	mem16 ← mem16 - 1	FF	mod 001 rm	Lomem	Himem			15+EA
NEG two's complement	reg8 ← 00h - reg8	F6	mod 011 rm					3
	reg16 ← 0000h - reg16	F7	mod 011 rm					3
	mem8 ← 00h - mem8	F6	mod 011 rm	Lomem	Himem			16+EA
	mem16 ← 0000h - mem16	F7	mod 011 rm	Lomem	Himem			16+EA
AAS ascii adjust subtraction	adjust AL, flags AH	3F						8
DAS decimal adjust subtraction	adjust AL, flags updated	2F						4
MUL unsigned multiplication	AX ← AL*reg8	F6	mod 100 rm					77
	DX:AX ← AX*reg16	F7	mod 100 rm					113
	AX ← AL*mem8	F6	mod 100 rm	Lomem	Himem			83+EA
	DX:AX ← AX*mem16	F7	mod 100 rm	Lomem	Himem			139+EA
IMUL signed multiplication	same as MUL except		mod 101 rm					MUL+21
AAM ascii adjust, multiplication	adjust AX	D4	0A					83
DIV unsigned division	AX ← AX/reg8	F6	mod 110 rm					90
	DX:AX ← AX/reg16	F7	mod 110 rm					162
	AX ← AX/mem8	F6	mod 110 rm	Lomem	Himem			96+EA
	DX:AX ← AX/mem16	F7	mod 110 rm	Lomem	Himem			168+EA
IDIV signed division	same as DIV	except	mod 111 rm					DIV+22
AAD ascii adjust for division	adjust AX	D5	0A					60
CBW convert byte to word	AX ← AL	98						2
CWD convert word to double	DX:AX ← AX	99						5

LOGIC GROUP

	Operation	Opcode	Operands					Clock
NOT one's complement	reg8 ← FFh - reg8	F6	mod 010 rm					3
	reg16 ← FFFFh - reg16	F7	mod 010 rm					3
	mem8 ← FFh - mem8	F6	mod 010 rm	Lomem	Himem			16+EA
	mem16 ← FFFFh - mem16	F7	mod 010 rm	Lomem	Himem			16+EA
SAL (SHL) shift arithmetic/logical left								
Memory/register by 1	shift reg8 by 1	D0	mod 100 rm					2
	shift reg16 by 1	D1	mod 100 rm					2
	shift mem8 by 1	D0	mod 100 rm	Lomem	Himem			15+EA
	shift mem16 by 1	D1	mod 100 rm	Lomem	Himem			15+EA
Memory/register by [CL]	shift reg8 by [CL]	D2	mod 100 rm					8+4/bit
	shift reg16 by [CL]	D3	mod 100 rm					8+4/bit
	shift mem8 by [CL]	D2	mod 100 rm	Lomem	Himem			20+EA+4/bit
	shift mem16 by [CL]	D3	mod 100 rm	Lomem	Himem			20+EA+4/bit
SHR logical shift right	similar to SAL	except	mod 101 rm					
SAR shift arithmetic right	similar to SAL	except	mod 111 rm					
ROL rotate left	similar to SAL	except	mod 000 rm					
ROR rotate right	similar to SAL	except	mod 001 rm					
RCL rotate left through carry	similar to SAL	except	mod 010 rm					
RCR rotate right through carry	similar to SAL	except	mod 011 rm					

LOGIC GROUP *continued*

	Operation	Code	Operands					Clock
AND logical								
register AND register	reg8 ← reg8 AND reg8	22	mod reg rm					3
	reg16 ← reg16 AND reg16	23	mod reg rm					3
memory AND register	mem8 ← mem8 AND reg8	20	mod reg rm	Lomem	Himem			16+EA
	mem16 ← mem16 AND reg16	21	mod reg rm	Lomem	Himem			16+EA
register AND memory	reg8 ← reg8 AND mem8	22	mod reg rm	Lomem	Himem			9+EA
	reg16 ← reg16 AND mem16	23	mod reg rm	Lomem	Himem			9+EA
AX/AL AND immediate	AL ← AL AND imm8	24	Data					4
	AX ← AX AND imm16	25	Lodata	Hidata				4
memory AND immediate (indirect)	byte[BX] ← byte[BX] AND imm8	80	mod 100 rm	Lodata				4
	word[BX] ← wordBX] AND imm16	81	mod 100 rm	Lodata				4
memory AND immediate (direct)	mem8 ← mem8 AND imm8	80	mod 100 rm	Lomem	Himem	Data		17+EA
	mem16 ← mem16 AND imm16	83	mod 100 rm	Lomem	Himem	Lodata	Hidata	17+EA
TEST logical compare								
register AND register	flags ← reg8 AND reg8	84	mod reg rm					3
	flags ← reg16 AND reg16	85	mod reg rm					3
memory/register AND memory/register	flags ← reg8 AND mem8	84	mod reg rm	Lomem	Himem			9+EA
	flags ← reg16 AND mem16	85	mod reg rm	Lomem	Himem			9+EA
AX/AL AND immediate	flags ← AL AND imm8	A8	Data					4
	flags ← AX AND imm16	A9	Lodata	Hidata				4
memory AND immediate (indirect)	flags ← byte [BX] AND imm8	F6	mod 000 rm	Lodata				5
	flags ← word [BX] AND imm16	F7	mod 000 rm	Lodata	Hidata			5
memory AND immediate (direct)	flags ← mem8 AND imm8	F6	mod 000 rm	Lomem	Himem	Data		11+EA
	flags ← mem16 AND imm16	F7	mod 000 rm	Lomem	Himem	Lodata	Hidata	11+EA
OR logical inclusive OR								
register OR register	reg8 ← reg8 OR reg8	0A	mod reg rm					3
	reg16 ← reg16 OR reg16	0B	mod reg rm					3
memory OR register	mem8 ← mem8 OR reg8	0A	mod reg rm	Lomem	Himem			16+EA
	mem16 ← mem16 OR reg16	08	mod reg rm	Lomem	Himem			16+EA
register OR memory	reg8 ← reg8 OR mem8	09	mod reg rm	Lomem	Himem			9+EA
	reg16 ← reg16 OR mem16	0B	mod reg rm	Lomem	Himem			9+EA
AX/AL OR immediate	AL ← AL OR imm8	0C	Data					4
	AX ← AX OR imm16	0D	Lodata	Hidata				4
memory OR immediate (indirect)	flags ← byte [BX] OR imm8	80	mod 001 rm	Lodata				5
	flags ← word [BX] OR imm16	81	mod 001 rm	Lodata	Hidata			5
memory OR immediate (direct)	mem8 ← mem8 OR imm8	80	mod 001 rm	Lomem	Himem	Data		17+EA
	mem16 ← mem16 OR imm16	81	mod 001 rm	Lomem	Himem	Lodata	Hidata	17+EA
XOR logical exclusive OR								
register XOR register	reg8 ← reg8 XOR reg8	32	mod reg rm					3
	reg16 ← reg16 XOR reg16	33	mod reg rm					3
memory XOR register	mem8 ← mem8 XOR reg8	30	mod reg rm	Lomem	Himem			16+EA
	mem16 ← mem16 XOR reg16	31	mod reg rm	Lomem	Himem			16+EA
register XOR memory	reg8 ← reg8 XOR mem8	32	mod reg rm	Lomem	Himem			9+EA
	reg16 ← reg16 XOR mem16	33	mod reg rm	Lomem	Himem			9+EA
AX/AL XOR immediate	AL ← AL XOR imm8	34	Data					4
	AX ← AX XOR imm16	35	Lodata	Hidata				4
memory XOR immediate (indirect)	byte[BX] ← byte[BX] XOR imm8	80	mod 110 rm	Lodata				4
	word[BX] ← word[BX] XORimm16	81	mod 110 rm	Lodata	Hidata			4
memory XOR immediate (direct)	mem8 ← mem8 XOR imm8	80	mod 110 rm	Lomem	Himem	Data		17+EA
	mem16 ← mem16 XOR imm16	81	mod 110 rm	Lomem	Himem	Lodata	Hidata	17+EA

CONTROL TRANSFER GROUP

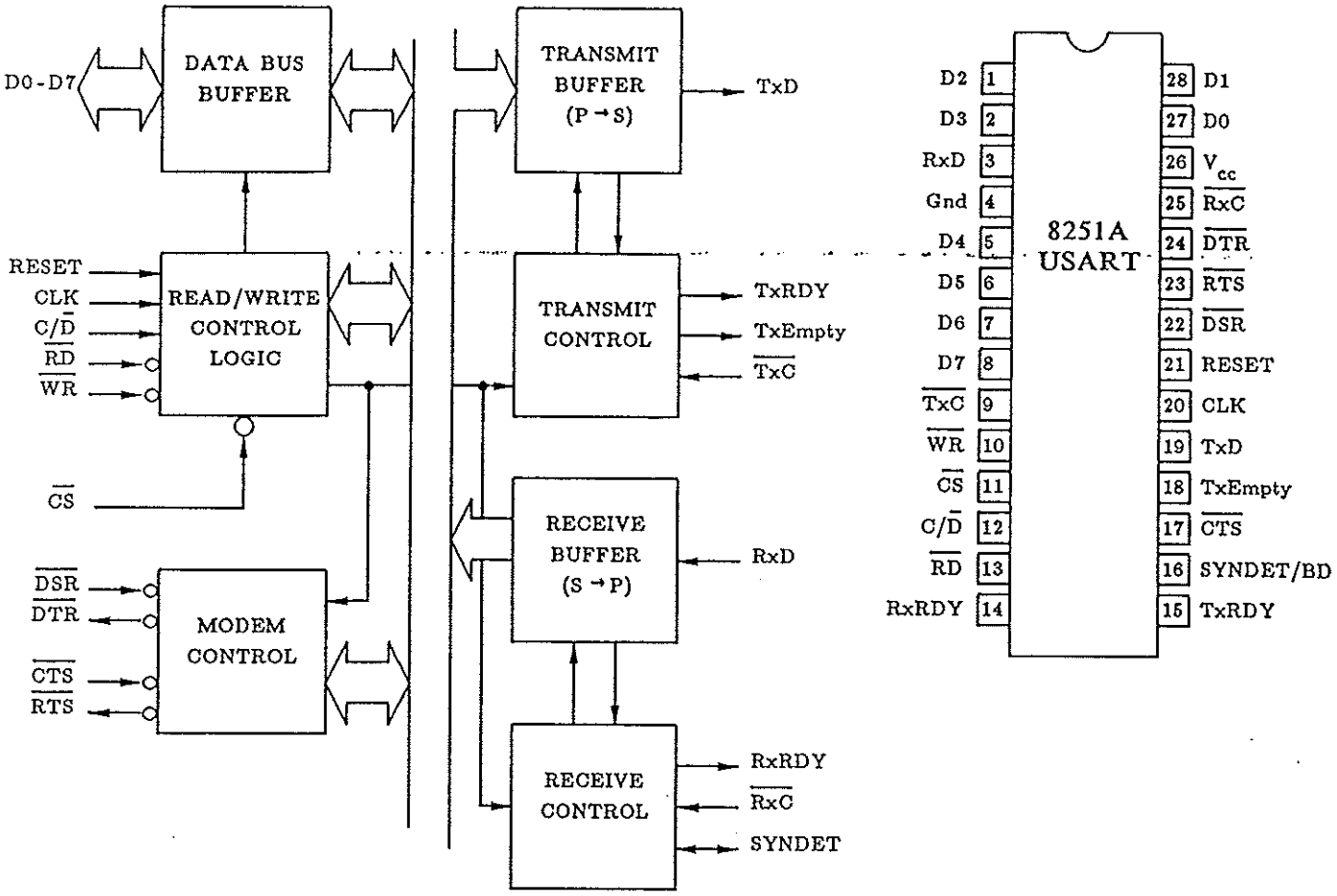
	Operation	Code	Operands				Clock
CALL a subroutine							
within segment, IP relative	[SP] ← IP, IP ← IP+disp	E8	Lodisp	Hidisp			19
indirect	[SP] ← IP, IP ← IP+reg16	FF	mod 010 rm				16
	[SP] ← IP, IP ← mem16	FF	mod 010 rm	Lomem	Himem		21+EA
inter segment, direct	[SP] ← IP, [SP - 2] ← IP, CS ← seg, IP ← off	9A	Lo off	Hi off	Lo seg	Hi seg	28
indirect	[SP] ← IP, [SP - 2] ← IP, IP ← [mem16], CS ← [mem16+2]	FF	mod 011 rm				37+EA
RET return from subroutine							
within segment	IP ← [SP]	C3					8
inter segment	CS ← [SP], IP ← [SP+2]	CB					12
RET data return add data to SP							
within segment	IP ← [SP], SP+data	C2	Lodata	Hidata			18
inter segment	CS ← [SP], IP ← [SP+2], SP+data	CA	Lodata	Hidata			17
JMP							
within segment, IP relative	IP ← IP+disp16	E9	Lodisp	Hidisp			15
	IP ← IP+disp8	EB	Lodisp				15
indirect	IP ← reg16	FF	mod 100 rm				11
	IP ← mem16	FF	mod 100 rm	Lomem	Himem		18+EA
inter segment, direct	CS ← seg, IP ← off	EA	Lo off	Lo off	Lo seg	Hi seg	15
indirect	IP ← [mem16], CS ← [mem16+2]	FF	mod 101 rm				24+EA
J condition jump on 'condition'							
JA, JNBE	above, not below or equal	77	disp				16 or 4
JAE, JNB, JNC	above or equal, not below, not carry	73	disp				16 or 4
JB, JNAE, JC	below, not above or equal, carry	72	disp				16 or 4
JBE, JNA	below or equal, not above	76	disp				16 or 4
JE, JZ	equal, zero	74	disp				16 or 4
JG, JNLE	greater, not less or equal	7F	disp				16 or 4
JGE, JNL	greater or equal, not less	7D	disp				16 or 4
JL, JNGE	less, not greater or equal	7C	disp				16 or 4
JLE, JNG	less or equal, not greater	7E	disp				16 or 4
JNE, JNZ	not equal, not zero	75	disp				16 or 4
JNO	not overflow	71	disp				16 or 4
JNP, JPO	not parity, parity odd	7B	disp				16 or 4
JNS,	not sign	79	disp				16 or 4
JO	overflow	70	disp				16 or 4
JP, JPE	parity, parity even	7A	disp				16 or 4
JS	sign	78	disp				16 or 4
JCXZ	if CX = 0 (no flag test)	E3	disp				18 or 6
LOOP until CX = 0	CX ← CX-1	E2	mod reg rm				17 or 5
LOOPE/LOOPZ ZF must be 1	as above	E1	mod reg rm				18 or 6
LOOPNE/LOOPNZ ZF must be 0	as above	E0	mod reg rm				19 or 5
INT 3 single step interrupt		CC					52
INT n interrupt type specified		CD	n				51
INTO interrupt on overflow	INT 4 if OF set	CE					53 or 4
IRET return from interrupt		CF					24

STRING MANIPULATION GROUP

	Operation	Opcode	Operands				Clock
REP repeat next instruction	repeat until CX = 0	F3					2
REPE/REPZ ZF must be 1	repeat until CX = 0	F3					2
REPNE/REPNZ ZF must be 0	repeat until CX = 0	F2					2
CMPSB compare byte string	flags ← [SI] to [DI]	A6					22
CMPSW compare word string	flags ← [SI] to [DI]	A7					22
LODSB load byte string	AL ← [SI]	AC					12
LODSW load word string	AX ← [SI]	AD					12
MOVSB move byte string	[DI] ← [SI]	A4					18
MOVSW move word string	[DI] ← [SI]	A5					18
SCASB scan byte string	flags ← [DI] - AL	AE					15
SCASW scan word string	flags ← [DI] - AX	AF					15
STOSB store byte string	[DI] ← AL	AA					11
STOSW store word string	[DI] ← AX	AB					11

PROCESSOR CONTROL GROUP

	Operation	Opcode	Operands				Clock
CLC clear carry flag	CF ← 0	F8					2
CMC complement carry flag	CF ← $\overline{CF}$	F5					2
STC set carry flag	CF ← 1	F9					2
CLD clear direction flag	DF ← 0	FC					2
STD set direction flag	DF ← 1	FD					2
CLI clear interrupt enable flag	IF ← 0	FA					2
STI set interrupt enable flag	IF ← 1	FB					2
HLT halt	halt	F4					2
WAIT wait for TEST line		9B					3+5n
ESC n to external device, register	data bus	D8+n	mod xxx rm				2
memory	data bus	D8+n	mod xxx rm	Lomem	Himem		8+EA
LOCK assert bus lock prefix	lock next instr	F0					2
NOP do nothing	AX ⟷ AX	90					3



Maximum current requirement - 100 mA

Register Addresses

Register	Activity allowed	C/D	Actual Port address
Data	Read/Write	0	18
Mode	Read/Write	1	1A
Control/Status	Write only	1	1A

Function and Pin descriptions

The 8251 is shown on the circuit diagram on page 18.

INTERFACE TO 8086 BUS

D0 - D7 Data Bus connections to low 8-bits of the board data bus.

RD & WR Control lines from the 8086

CS Chip Select, decoded to port address 00011xxx.

C/D Connected to A1 of the board address bus, selects the internal control or data registers.

RESET A high forces an 'idle' mode. A command reset does the same.

CLK Internal device timing signal, must be > 30 times baud rate. Connected to PCLK on the FLIGHT 86 Controller Board

MODEM CONTROL

$\overline{DSR}$ Data Set Ready input, can be tested using a status read.

$\overline{DTR}$ Data Terminal Ready output, can be set using the command word.

$\overline{RTS}$ Request to Send output, can be set using the command word.

$\overline{CTS}$ Clear to send input. Provided the Tx enable bit is '0', a low on this enables the 8251A to transmit serial data.

TRANSMITTER BUFFER

Converts parallel data from Data Bus to a serial bit stream.

TxRDY Transmitter Ready indicates when ready to accept another data byte.

TxEmpty Transmitter Empty goes high when no characters to send.
Not implemented on controller board

$\overline{TxC}$ Transmitter Clock controls rate of transmission of the serial data stream. Actual speed depends upon software multiplication factor.

TxD Transmitter Data output pin for the serial data stream.

RECEIVER BUFFER

Accepts serial data and converts it to parallel ready for sending to the data bus

RxRDY Receiver Ready indicates when ready to accept another data byte.

$\overline{RxC}$ Receiver Clock controls rate of transmission of the serial data stream. Actual speed depends upon software multiplication factor.

RxD Receiver Data input pin for the serial data stream.

SYNDET/BD Is only used with synchronous data transfers, it may be input or output for synchronising the data characters.
It is not implemented on the controller board.

General Operation

The 8251A may operate its transmitter and receiver independently. It may even operate them at different speeds. On the controller board they must both work at the same speed, software programmable using the 8253 clock. Although it is capable of operating in the sychronous mode it is probable that most work will be carried out in the asynchronous mode. Asynchronous operation only will be described in this note.

The programming is straight forward, but needs some 'tricks' to work under all circumstances.

Once programmed, the transmitter indicates it is ready to accept a character from the CPU by raising the TxRdy line. This will not occur until the TxEnable bit has been set in the Command instruction and has received a Clear To Send input.

Likewise, once the receiver has a complete character from the serial line in its buffer, it raises the RxRdy line to tell the CPU it can take it. Until the CPU reads the character, any more will inhibited from being sent down the serial line.

The control words are split into 2 formats:

The Mode Instruction, which MUST follow a reset (internal or external), determines the general operation of the 8251A.

The Command Instruction defines the detailed operation.

Programming

After a system RESET the MODE register must be the first to be set up. Because the 8251A may be in an unknown state it is normal practise to send the byte 00 to the control address three times. This will guarantee the 8251A command register is active. Now the byte value 40, bit 6 set, is sent to activate the mode register. The serial word format may now be set up.

The baud rate factor is the multiple of the input clocks required to satisfy the 'bit time' for the serial data. On the FLIGHT 86 board a factor of 16 is convenient to match the range available from the 8253. If parity is enabled, the parity bit is not considered as one of the data bits. If less than 8 data bits are defined, the other most significant bits are 'zeroed' before transfer to the data bus, and ignored by the transmitter.

Once the mode has been programmed the 8251A will revert to the command register. All bytes sent to the command register should keep bit 6 reset. The command register controls the transmitter and receiver control lines.

The 8251A Status

The CPU can examine the status register at any time to determine the current condition of the 8251A. This is particularly important when duplex communication and interrupt routines are being used. The status of the transmitter and receiver are readily interpreted by analysis of this register.

Under most circumstances the state of the three error flags (overrun, framing and parity) can be ignored. This is because most simple systems can do nothing about the error, anyway. The 8251A continues working normally for the next data byte even though an error was detected.

Sending Data

Provided the transmitter is enabled, as soon as a data byte is written to the data register, the 8251A will convert this to a serial form and send it to the TxD pin in the format specified. If the receiving device is not ready the word will be held and sent as soon as possible.

The serial data always starts with a start bit, in the receive mode this synchronises the data transfer to the centre of the bit times. The TxD output is held high unless a character is being sent. Thus the start bit is always the first low transition of one bit time.

Receiving Data

Provided the receiver is enabled, the 8251A will receive a serial word sent to it from another device, and will store the word in a buffer. The CPU can identify if a character has been received and read the data byte from the buffer. Once the data byte has been read the buffer is flushed and the data lost, as far as the 8251A is concerned.

Asynchronous Mode Instruction Format

Action Options		*D7* S2	*D6* S1	*D5* EP	*D4* PEN	*D3* L2	*D2* L1	*D1* B2	*D0* B1
Baud rate factor	Synchronous							0	0
	x 1							0	1
	x 16							1	0
	x 64							1	1
Character length	5 bits					0	0		
	6 bits					0	1		
	7 bits					1	0		
	8 bits					1	1		
Parity	Disable			x	0				
	Odd enable			0	1				
	Even enable			1	1				
Stop bits	invalid	0	0						
	1	0	1						
	1½	1	0						
	2	1	1						

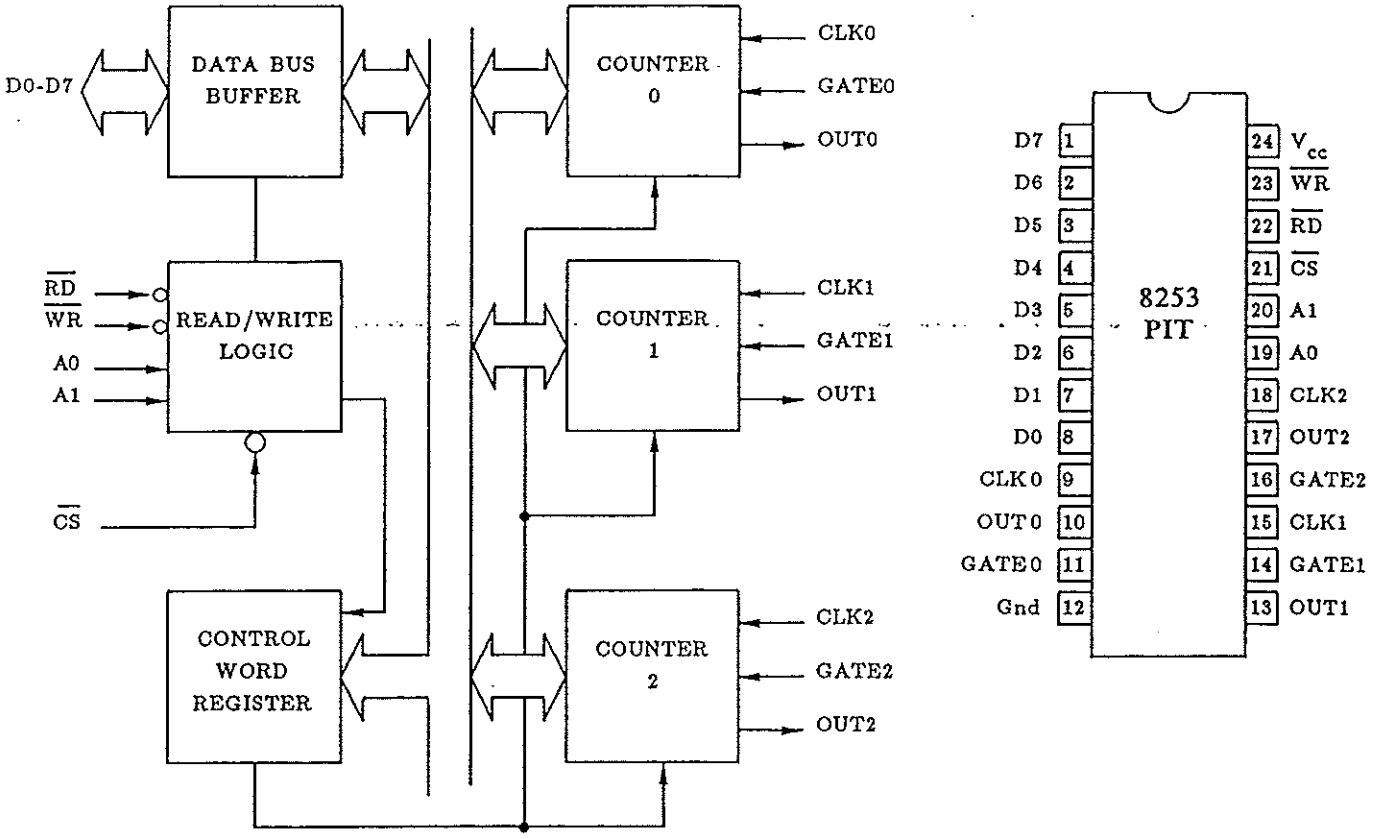
Command Instruction Format

Action Options		*D7* EH	*D6* IR	*D5* RTS	*D4* ER	*D3* SBRK	*D2* RxE	*D1* DTR	*D0* TxEN
Transmitter	Disable								0
	Enable								1
Force $\overline{\text{DTR}}$ pin to	'1'							0	
	'0'							1	
Receiver	Disable						0		
	Enable						1		
Send break	Normal operation					0			
	Force TxD low					1			
Reset					1				
Force $\overline{\text{RTS}}$ pin to	'1'			0					
	'0'			1					
Internal Reset	Stay in command		0						
	Force to Mode		1						
Enter Hunt mode	Asynch ignores	x							

Status Read Format

Action Options		*D7* DSR	*D6* SYN	*D5* FE	*D4* OE	*D3* PE	*D2* TxE	*D1* RxRDY	*D0* TxRDY
Transmitter Ready									1
Receiver Buffer Full								1	
Transmitter Buffer Empty							1		
Parity Error (8251A still works)						1			
Overrun Error (8251A still works)					1				
Framing Error (8251A still works)				1					
Used for synchronous mode only			x						
DSR is	'0'	1							
	'1'	0							

8253 PROGRAMMABLE INTERVAL TIMER



Maximum current requirement - 140 mA

Register Addresses

Register	Activity allowed	A1	A0	Actual Port address
Counter 0	Read/Write	0	0	08
Counter 1	Read/Write	0	1	0A
Counter 2	Read/Write	1	0	0C
Mode Control	Write only	1	1	0E

Function and Pin descriptions

The 8253 is shown on the circuit diagram on page 18.

INTERFACE TO 8086 BUS

D0 - D7 — Data Bus connections to low 8-bits of the board data bus.
$\overline{RD}$ & $\overline{WR}$ — Control lines from the 8086.
$\overline{CS}$ — Chip Select, decoded to port address 00001xxx.
A1 & A0 — Connected to A2 & A1 of the board address, selects internal registers.

COUNTERS - Each of the three counters is identical

CLKx — Counter clock source input
GATEx — Hardware counter control, can be used to inhibit the count. They are permanently enabled on the controller board.
OUTx — Counter output signal. Action determined by mode programmed.

General Operation

The action of each counter is independent so may be changed at any time. Note that on power-up the mode, count and output of all counters is undefined.

The Mode control register is used to determine the mode, size and type of count for each counter to be used. See the table below. The mode register is "write only" so software cannot determine the mode previously set.

The counter register is not loaded until the next falling edge of its counter clock input. If the register is read before this time the result will be invalid. The counters are down counters and are decremented on the falling edge of the counter input clock. Loading zeros in the count register will result in the maximum binary count of 2^{16}, or 10^4 in BCD.

Control Mode word format

Action Options		*D7* *D6* SC1 SC0	*D5* *D4* RL1 RL0	*D3* M2	*D2* M1	*D1* M0	*D0* BCD
Counter type	16-bit binary						0
	4 decade BCD						1
Count Mode	0			0	0	0	
	1			0	0	1	
	2			x	1	0	
	3			x	1	1	
	4			1	0	0	
	5			1	0	1	
Latch Counter			0 0				
Read/Load least significant byte			0 1				
most significant byte			1 0				
ls byte first then ms byte			1 1				
Select Counter	0	0 0					
	1	0 1					
	2	1 0					
Invalid Code		1 1					

Programming

The number of bytes (one or two) to be loaded or read is set by the mode command and this must be adhered to when loading or reading the counter values. If 2 bytes are to be used, the least significant byte is loaded or read first.

The only sure way of reading a stable count 'on the fly' is to use the mode command with both RL1 and RL0 set to '0', to latch the current register count, and then to read the counter value(s).

Mode 0 - Interrupt on Terminal Count

The output goes low after the mode set operation, and remains low while counting until the count decrements to zero (terminal count). The output will go high and remain high until the next mode is set or a new count is loaded. The counter continues decrementing internally.

Mode 1 - Programmable One-shot (monostable)

This mode requires a trigger at the counter gate input, so is not available on the controller board.

Mode 2 - Rate Generator

A divide by N counter. The output is low for one input clock period and then high for N clock periods, the cycle repeats until a new mode is selected. If the counter is reloaded between output pulses the current count is not affected but the count after the next input pulse will reflect the new value.

Mode 3 - Square Wave Rate Generator

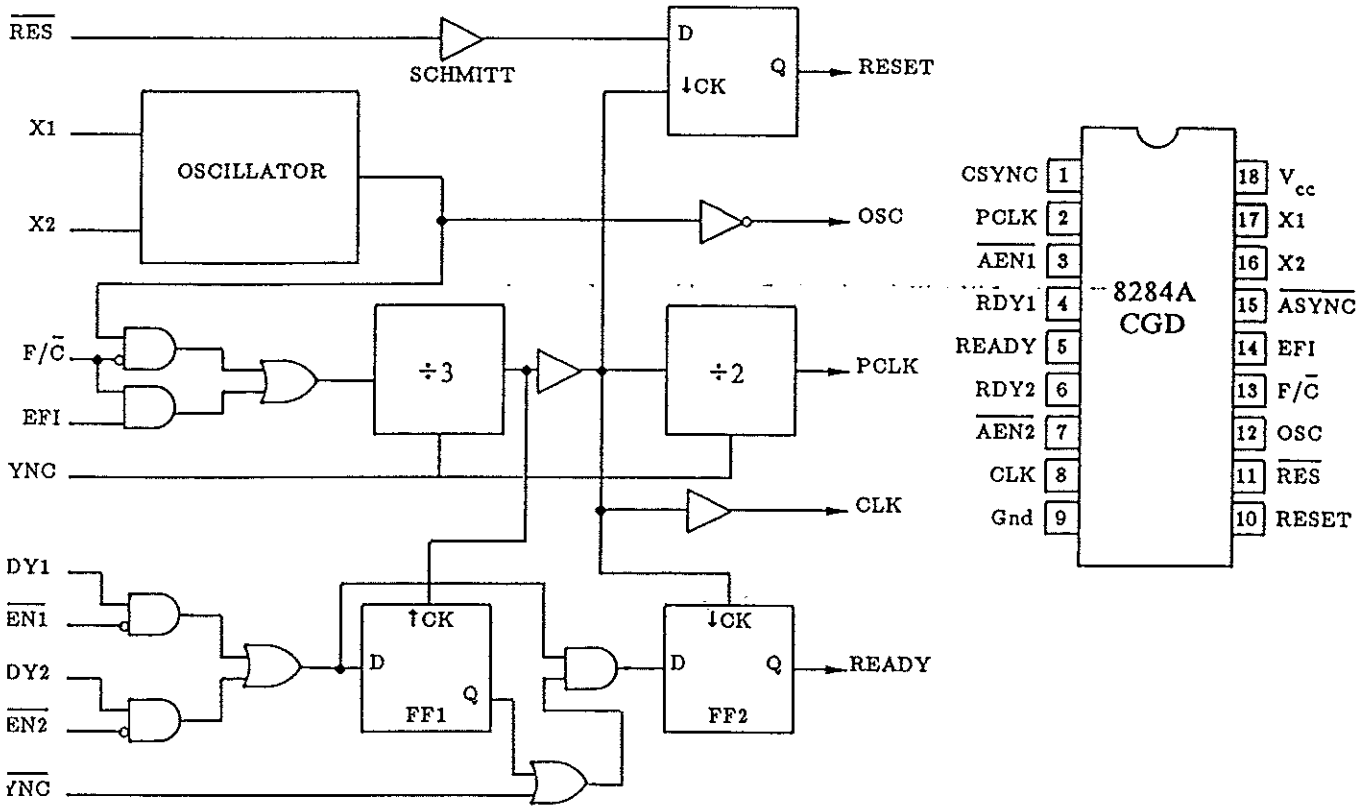
Similar to mode 2, except the output is high for the first half of the count and goes low for the other half. If an ODD numbered count (ie 5) is specified the square wave output will be assymetric (by one input clock cycle).

Mode 4 - Software Triggered Strobe

The output goes high once the mode is set, and remains high while the counter is decrementing. On count termination, the output will go low for one input clock cycle and then go high again. The output will remain high until a new mode or count is loaded.

Mode 5 - Hardware Triggered Strobe

This mode requires a trigger at the counter gate input, so is not available on the controller board.



Maximum current requirement - 160 mA

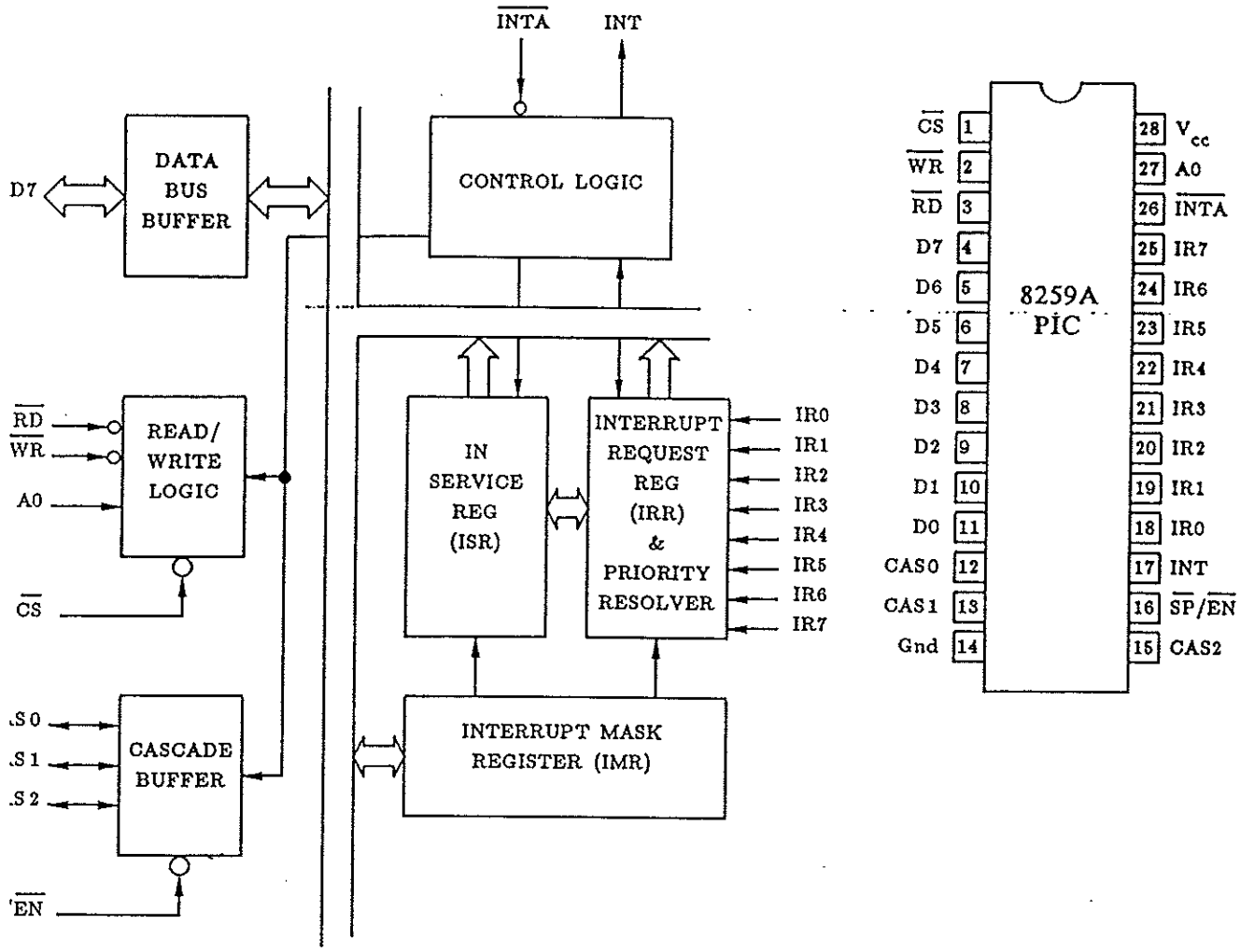
Pin descriptions

The 8284A is being used in its simplest form on the controller board, see the circuit on page 12, so many of the input control lines are not actively utilised or are hard wired to a logic level. These are:

$\overline{AEN1}$, $\overline{AEN2}$	Address Enable inputs
RDY1, RDY2	Bus Ready inputs
$\overline{ASYNC}$	Ready Synchronisation Select for the READY input (It is pulled-up internally)
CSYNC	Clock Synchronisation input, allows multiple 8284's to be used.
F/$\overline{C}$	Frequency/Crystal select determines source of control frequency. '0' uses the external crystal, '1' uses the EFI oscillator input.
EFI	External Frequency Input. Only used when F/C is high.
OSC	Oscillator Output is the TTL level output of the crystal.

The active lines are:

X1, X2	Connections for external series resonant crystal. Crystal frequency is 3 times the desired CPU frequency.
$\overline{RES}$	Reset Input. A low input produces a high RESET output.
RESET	Reset output is active high, synchronised to the falling edge of CLK.
READY	Ready output is active high, synchronised to RDY input.
CLK	Processor Clock output for direct connection to the 8086 CLK input. The frequency is a third that of the crystal with a 33% duty cycle.
PCLK	Peripheral Clock output, half the CLK output with a 50% duty cycle.



Maximum current requirement - 85 mA.

Register Addresses

Register	Activity allowed	A0	Actual Port Address
Initialisation Command Words			
ICW1	Write only	0	10 +
ICW2, 3, 4	Write only	1	12
Operation Command Words			
OCW1	Write only	1	12
OCW2	Write only	0	10 +
OCW3	Write only	0	10 +
Status Registers			
IMR (Interrupt Mask Register)	Read only	1	12
IRR (Interrupt Request Register)	Read only	0	10 *
ISR (In-Service Register)	Read only	0	10 *

* activated and identified using OCW3
\+ ICW1, OCW2 and OCW3 are identified using bits 3 and 4

Function & Pin Descriptions

The 8259A is shown on the circuit diagram on page 18.

	INTERFACE TO 8086 BUS
D0-D7	Data bus connections to low 8-bits of board data bus.
$\overline{RD}$ & $\overline{WR}$	Control lines from the 8086.
$\overline{CS}$	Chip Select, decoded to port address 00010xxx.
A0	Connected to A1 of the board address, selects internal registers.
INT	Interrupt output pin. Connected to 8086 INTR pin. Goes high when a valid interrupt is available.
$\overline{INTA}$	Interrupt Acknowledge input from 8086 used to strobe data from 8259A onto the data bus.
	INTERFACE TO PERIPHERAL CHIPS
IRx	Interrupt Request asynchronous inputs. A peripheral requests an interrupt response by taking its individual line from low to high, see page 4 for priority table.
	INTERFACE TO SLAVE 8259A's (not available on FLIGHT 86 board)
CASx	Cascade lines form a 'private' bus for master slave action.
$\overline{SP}/\overline{EN}$	Slave Program/Enable Buffer. Designates master slave operation.

General Operation

The 8259A PIC operates as an overall manager for the FLIGHT 86 peripheral chip system, allowing them to operate in an interrupt driven mode if required. It accepts requests from the peripheral chips and determines which of the incoming requests has the highest priority. The 8259A then issues the interrupt to the 8086 and supplies a type pointer to the 8086 to enable the correct Interrupt Service Routine to be executed. Because of the confusion of terms, ISR will mean In-Service Register in this 8259A note.

The 8259A can manage eight levels or requests, which can be expanded to 64 levels by using up to seven more slave 8259A's. This is not possible on the FLIGHT 86 controller board. The 8259A is programmed by user software to perform a number of modes of operation. These may be changed dynamically at any time in the program.

Resolving Interrupts

The peripheral interrupt requests are handled by the IRR (Interrupt Request Register) and the ISR (In-Service Register) in cascade. The IRR stores all the interrupt levels of the devices requesting service, and the ISR stores all the interrupt levels that are being serviced.

The highest priority in the IRR is determined by the priority resolver and this is selected and strobed into the ISR during an $\overline{INTA}$ pulse.

The IMR (Interrupt Mask Register) stores the bits that determine the masking of interrupts. The IMR determines which bits in the IRR are allowed through to the ISR. Masking is individual so a masked bit does not affect another level.

The contents of the IRR, ISR and IMR may be read by the 8086 with suitable programming of the OCW3 register.

Interrupt Sequence

1. An IRx (Interrupt Request) line is raised by a peripheral device, setting the corresponding bit in the IRR.
2. If the bit is NOT masked in the IMR and the interrupt is enabled, the 8259A will send an interrupt to the 8086 via the INT line.
3. The 8086 acknowledges the interrupt request with a $\overline{\text{INTA}}$ pulse.
4. Upon receipt of the $\overline{\text{INTA}}$ from the 8086 the 8259A freezes the pending interrupts. The highest pending priority ISR bit is SET and the corresponding IRR bit is RESET. No data is placed on the bus at this stage.
5. The 8086 sends a second $\overline{\text{INTA}}$ pulse, the 8259A treats this as a Read pulse and places an 8-bit pointer onto the data bus.
6. If the 8259A is in the AEOI (Automatic End Of Interrupt) mode then the ISR bit is reset. Otherwise the ISR bit remains set until a suitable EOI command is issued (usually at the end of the Interrupt Service Routine).

If the interrupt was removed before step 4 then the 8259A will issue the lowest level interrupt 7.

The Command Words

ICW's (Initialisation Command Words).
Before the 8259A can operate normally it must be initialised by a sequence of three ICW commands (Could be 2 to 4, but the FLIGHT 86 board will always requires 3 as it uses the 8086 and no slave can be fitted).

OCW's (Operation Command Words).
These set the 8259A to the various possible modes of interrupt, (fully nested, rotating priority, special mask, polled), and can be written at any time.

Reading the Status Registers

The IMR, 8-bit register, contains the interrupt lines mask state (1= masked or inhibited, 0 = not masked or enabled). The IMR may be read at any time from address 12. The bit order is the same as in the OCW1.

The other two registers IRR and ISR can only be read, from address 10, if the RR bits have been pre-programmed using an OCW3. After initialisation the 8259A defaults to read IRR until a suitable OCW3 is received.

The IRR, 8-bit register, contains the levels requesting interrupts, yet to be acknowledged. The highest request is reset when the $\overline{\text{INTA}}$ is received.

IMR does NOT affect these states.

The ISR, 8-bit register, contains the priority levels that are being serviced. The ISR is updated when the EOI or AEOI is received.

Initialisation

An ICW1 has been issued by the 8086 if A0 = 0 and D4 = 1, so OCW2 and OCW3 must ensure D4 = 0. The ICW1 starts the initialisation sequence, which MUST be completed, and causes the following to happen internally:

1. The IRR edge sense circuit is reset, so the IRx inputs must go low to high to generate an interrupt.
2. The IMR is cleared.
3. IR7 input is assigned priority 7.
4. The slave mode address is set to 7 (not used on FLIGHT 86 board).
5. The Special Mask Mode is cleared and the Status read is set to IRR.
6. If IC4 = 0 then ICW4 is zeroed (the 8086 requires IC4 = 1).

ICW1 Format

Action Options	*D7*	*D6*	*D5*	*D4*	*D3* LTIM	*D2* ADI	*D1* SNGL	*D0* IC4
ICW4 not needed (8085) ICW4 needed (8086)								0 1
Cascade 8259s (ICW3 required) Single 8259 (No ICW3)							0 1	
Address Interval 8 bytes 4 bytes (8086)						0 1		
Trigger Mode Edge Level					0 1			
Must be 1 Not used by 8086 systems	x	x	x	1				

ICW2 Format

Action Options	*D7* T7	*D6* T6	*D5* T5	*D4* T4	*D3* T3	*D2*	*D1*	*D0*
Not used by 8086 systems						x	x	x
Interrupt Type Pointer Preset to INT32 by FLIGHT 86 monitor	0	0	1	0	0			

ICW3 Format *NOTE: NOT used on FLIGHT 86 board*

Action Options	*D7* S7	*D6* S6	*D5* S5	*D4* S4	*D3* S3	*D2* S2	*D1* S1	*D0* S0
Slave device position used	x	x	x	x	x	x	x	x

ICW4 Format

Action Options	*D7*	*D6*	*D5*	*D4* SFNM	*D3* BUF	*D2* M/S	*D1* AE01	*D0* uPM
Microprocessor Mode 8085 8086								0 1
End of Interrupt Normal Single 8259 Automatic							0 1	
Mode Non buffered (FLIGHT 86 board) Buffered Slave Buffered Master					0 1 1	x 0 1		
Not SFNM (FLIGHT 86 board) Special Fully Nested Mode				0 1				
Always zero	0	0	0					

Once the ICW sequence is complete the 8259A is ready to accept interrupts on the IRx lines. The default conditions may be overridden by use of the OCW's.

Operation Commands

These commands may be sent at any time, after initialisation, to dynamically alter the 8259A operation.

OCW1 Format - sets and clears the mask conditions in the IMR (Interrupt Mask Register). 1 to mask or inhibit the channel, 0 to unmask or enable the channel.

Action Options	*D7* M7	*D6* M6	*D5* M5	*D4* M4	*D3* M3	*D2* M2	*D1* M1	*D0* M0
Mask channel	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

OCW2 Format - controls the Rotate and end of interrupt modes.

Action Options	*D7* R	*D6* SL	*D5* EOI	*D4*	*D3*	*D2* L2	*D1* L1	*D0* L0
Interrupt level to be used IR0						0	0	0
IR1						0	0	1
IR2						0	1	0
IR3						0	1	1
IR4						1	0	0
IR5						1	0	1
IR6						1	1	0
IR7						1	1	1
Always 0 identifies OCW2					0			
Always 0 identifies OCW				0				
End of Interrupt Non specific EOI	0	0	1					
Specific EOI *	0	1	1					
Rotation on non-specific EOI	1	0	1					
in automatic EOI mode set	1	0	0					
EOI mode clear	0	0	0					
Rotate on specific EOI *	1	1	1					
Set priority to lowest *	1	1	0					
No operation	0	1	0			* uses L2 to L0		

OCW3 Format - controls the special mask mode and status register usage.

Action Options	*D7*	*D6* ESMM	*D5* SMM	*D4*	*D3*	*D2* P	*D1* RR	*D0* RIS
No action							0	x
Enable Read IRR							1	0
ISR							1	1
Poll Mode Not polled						0		
polled *						1		
Always 1 identifies OCW3					1			
Always 0 identifies OCW				0				
No action		0	x					
Reset special mask		1	0					
Set special mask		1	1					
Always zero	0							

* polling overrides the status read (RR must be 1 to be active)

Fully nested mode

The interrupt requests are ordered as IR0 (highest) through IR7. The highest priority pending interrupt is used when the $\overline{\text{INTA}}$ signal is received. The corresponding bit is set in the ISR, and this remains set until an EOI is received. While this bit is set all interrupts of the same or lower priority are inhibited. Priorities may be changed using the rotating priority mode.

AEOI, Automatic End of Interrupt

If bit 1, the AEOI bit, of ICW4 is set, the appropriate bit in the ISR will be reset automatically following the falling edge of the second $\overline{\text{INTA}}$ pulse.

EOI, Normal End of Interrupt

If the AOEI bit is not set, a specific command must be issued to the 8259A before returning from the interrupt service routine. The 8259A can determine the bit to be reset if it is being used in a fully nested mode. A Non-specific EOI may be issued, with OCW2, this resets the highest bit set in the ISR. If an ISR bit is masked by an IMR bit, it will NOT be cleared by a Non-specific EOI command.

If the 8259A is being used in a mode that has disturbed the priorities, then a specific EOI must be used, with OCW2. The values in L0 to L2 determine the position of the bit to be reset.

Rotating Priority modes

If equal priority is required for interrupting devices, the last device to be serviced may be set to the lowest priority. This is achieved by rotating the priorities in the ISR when the EOI is issued. Automatic or specific rotation may be used with the OCW2.

Specific priorities may be set by programming the lowest priority in OCW2.

Special mask mode

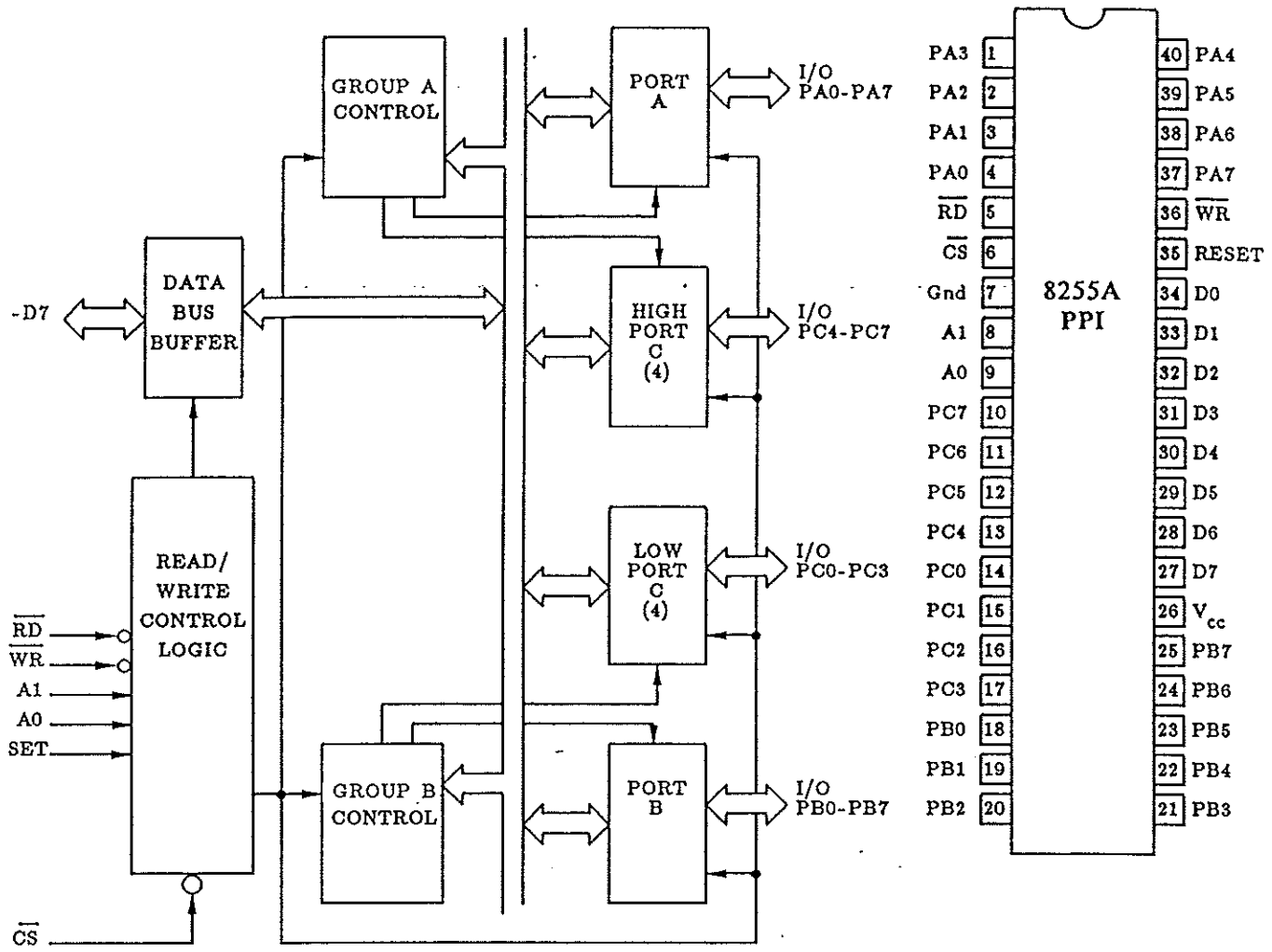
The special mask mode enables lower level interrupts to be used even though an EOI has not been issued for the current interrupt. If the SMM bit and ESMM bits are set in OCW3 then any interrupts may be selectively enabled by loading the mask register.

Polled mode.

The 8259A may be 'polled' if the 8086 interrupts are disabled, by setting bit 2, the P bit, in OCW3. Any read from the 8259A will result in D7 being set if a device is requesting service, and bits D2 to D0 represent the binary code of the highest priority level requesting service.

Edge and Level triggered modes.

Programmed by using bit 3, the LTIM bit, in ICW1. In the edge triggered mode the IRx input can remain high without triggering another interrupt request. If the level mode is used the high on the IRx input MUST be removed before the EOI signal is issued. In BOTH modes the IRx input MUST remain high until after the falling edge of the first $\overline{\text{INTA}}$. If the IRx signal is removed prior to this, the priority will default to IR7. To differentiate between a 'genuine' and an 'enforced' IR7, the IR7 interrupt service routine should examine the ISR. The IR7 bit will be set for a genuine IR7 interrupt request.



Maximum current requirement - 120 mA.

Register Addresses

Register	Activity allowed	A1	A0	Actual Port Addresses U10	U9
Port A	Read/Write	0	0	00	01
Port B	Read/Write	0	1	02	03
Port C	Read/Write	1	0	04	05
Control (D7 defines use)	Write only	1	1	06	07

Function & Pin Descriptions

The 8255A is shown on the circuit diagram on page 18.

INTERFACE TO 8086 BUS

D0-D7 Data bus connections to low 8-bits of board data bus.

$\overline{RD}$ & $\overline{WR}$ Control lines from the 8086.

$\overline{CS}$ Chip Select, decoded to port address 00000xxx.

A1 & A0 Connected to A2 and A1 of the board address, selects internal registers.

RESET Reset, a high on this pin clears the control register and all three ports are set to input mode, ie tristate. After a reset the 8255A will remain in the input mode. Connected to 8086 RESET line.

INTERFACE TO OUTSIDE WORLD

GROUP A

PA0-PA7 Port A, 8-bit programmable input/output port.

PC4-PC7 High nibble of Port C, 4-bit programmable input/output port, and may be used as controller for the A port.

GROUP B

PB0-PB7 Port B, 8-bit programmable input/output port.

PC0-PC3 Low nibble of Port C, 4-bit programmable input/output port, and may be used as controller for the B port.

General Operation

The 8255A can operate in three modes:

Mode 0 Basic Input/Output
Mode 1 Strobed Input/Output
Mode 2 Bi-directional bus (not available on FLIGHT 86 board)

The modes for Ports A and B may be separately defined, and Port C is assigned between them to act as a control port if desired. Whenever a mode is changed ALL the output registers will be reset. The control word defines the activity of the individual ports.

Control Word Formats

		D7	*D6*	*D5*	*D4*	*D3*	*D2*	*D1*	*D0*
MODE DEFINITION FORMAT		1							
GROUP A Port A	Mode 0		0	0					
	Mode 1		0	1					
	Mode 2		1	x					
	Direction Output				0				
	Input				1				
Port C	High Direction Output					0			
	Input					1			
GROUP B Port B	Mode 0						0		
	Mode 1						1		
	Direction Output							0	
	Input							1	
Port C	Low Direction Output								0
	Input								1
PORT C BIT SET/RESET FORMAT		0							
Don't care			x	x	x				
Select	Bit 0					0	0	0	
	Bit 1					0	0	1	
	Bit 2					0	1	0	
	Bit 3					0	1	1	
	Bit 4					1	0	0	
	Bit 5					1	0	1	
	Bit 6					1	1	0	
	Bit 7					1	1	1	
Set/Reset	Reset								0
	Set								1

The Port C Bit Set/Reset format is shown on page 101, for this to take effect D7 must be reset. This mode gives individual bit set/reset facilities for Port C. This is particularly useful when Port C is used to control either Port A or Port B. These lines may also be used to control the interrupt lines, reset the bit to disable, or set the bit to enable the interrupt.

Note: all these mask flip-flops are reset during mode selection or a RESET. PC6 and PC7 are not used on the FLIGHT 86 board.

Mode 0 Operation

Mode 0 configuration gives the simplest form of I/O possible, where no 'handshaking' is required. Data is simply read from or written to the specified port. The 16 possible control words are shown below.

Mode 0 Port definition

Note D7, D6, D5 and D2 are fixed.

GROUP A		GROUP B							
PORT A	PORT C high	PORT B	PORT C low	*D7*	*D6*	*D5*	*D4*	*D3*	*D2*	*D1*	*D0*
OUT	OUT	OUT	OUT	1	0	0	0	0	0	0	0
OUT	OUT	OUT	IN	1	0	0	0	0	0	0	1
OUT	OUT	IN	OUT	1	0	0	0	0	0	1	0
OUT	OUT	IN	IN	1	0	0	0	0	0	1	1
OUT	IN	OUT	OUT	1	0	0	0	1	0	0	0
OUT	IN	OUT	IN	1	0	0	0	1	0	0	1
OUT	IN	IN	OUT	1	0	0	0	1	0	1	0
OUT	IN	IN	IN	1	0	0	0	1	0	1	1
IN	OUT	OUT	OUT	1	0	0	1	0	0	0	0
IN	OUT	OUT	IN	1	0	0	1	0	0	0	1
IN	OUT	IN	OUT	1	0	0	1	0	0	1	0
IN	OUT	IN	IN	1	0	0	1	0	0	1	1
IN	IN	OUT	OUT	1	0	0	1	1	0	0	0
IN	IN	OUT	IN	1	0	0	1	1	0	0	1
IN	IN	IN	OUT	1	0	0	1	1	0	1	0
IN	IN	IN	IN	1	0	0	1	1	0	1	1

Mode 1 operation

Mode 1 gives the facility of strobed input/output. Thus 'handshaking' signals may be used for the transfer of data. The nibble of Port C assigned to either the Port A or Port B is used for the handshake control.

Strobed Input Control Signals

The control word for Port A strobed input is 1011xxxx and for Port B strobed input is 1xxxx11x. They may both be defined strobed input with 1011x11x.

The Port C pin definitions are usually renamed for this mode:

PORT A	PORT B	Function
PC4 = $\overline{\text{STBa}}$	PC2 = $\overline{\text{STBb}}$	Strobe, input
PC5 = IBFa	PC1 = IBFb	Input Buffer full, output
PC3 = INTRa	PC0 = INTRb	Interrupt request, output

When the strobe input goes low the port data is loaded into the latch.

The buffer full output goes high to indicate the latch has been loaded. It does not go low again until the data in the latch is read.

The interrupt request output, may be used to interrupt the 8086, via the 8259A. It goes high, provided STB and IBF are both high and the interrupt enable flag is set. The interrupt enable for Port A is controlled by the set/reset of Port C4, Port B is controlled by the set/reset of Port C2,

Strobed Output Control Signals

The control word for Port B strobed output is 1xxxx10x. You cannot use Port A in the strobed output mode, on the FLIGHT 86 board, as PC6 & PC7 are not available.

The Port C pin definitions are usually renamed for this mode:

PORT A	PORT B	Function
PC7 = $\overline{\text{OBFa}}$	PC1 = $\overline{\text{OBFb}}$	Output Buffer full, output
PC6 = $\overline{\text{ACKa}}$	PC2 = $\overline{\text{ACKb}}$	Acknowledge, input
PC3 = INTRa	PC0 = INTRb	Interrupt request, output

The output buffer full goes low to indicate the valid data has been written to the port by the 8086 and now contains valid data. It does not go high again until the external device acknowledges this.

A low on the acknowledge input indicates the external device has accepted the data on the port. When this line goes high the 8086 can write another byte to the port.

The interrupt request output, may be used to interrupt the 8086, via the 8259A. It goes high, provided $\overline{\text{ACK}}$ and $\overline{\text{OBF}}$ are both high and the interrupt enable flag is set. Port B is controlled by the set/reset of Port C2.

Mode 2 operation

This mode of operation is not available on the FLIGHT 86 board.

INDEX