I N D E X

SECTION	CONTENTS
A	GETTING STARTED
B	MONITOR
C	BASIC
D	PASCAL (Optional)
E	FORTH (Optional)
F	MICROPROCESSOR BACKGROUND
G	MAD-MICROPROCESSOR ASSEMBLER/ DISASSEMBLER AND PROGRAMMING REFERENCE MANUAL (Optional)
H	THEORY OF OPERATION, CIRCUIT AND PARTS LIST
I	APPENDIX & GLOSSARY OF TERMS

GETTING STARTED

This section helps you to get your Pegasus up and running. It describes aspects of Pegasus operation which you will need to know no matter which language you are using.

If you have not got your Pegasus up and running yet, refer to Appendix One of this section.

At this point you should have a Menu on the Screen like this :-

```
AAMBER Pegasus 6809
Your Name

Basic 1.2
Monitor 2.0

Select one of above:
```

This is always the way the Pegasus powers up. It has scanned through memory and found what programs are present and written these into the menu. If your menu is different from that above it is likely you have different Eproms installed. (If you want to change eproms refer to Appendix 2 at the back of this section). Eproms are the integrated circuits on your board which have labels on them. They contain software such as the Monitor and Basic.

The Monitor Eprom must never be removed. Without it your Pegasus will not function. The other two Eprom sockets may have any of a wide range of languages; a list of some of these follows:

Language	No. of Eproms	
Basic	1 eprom)	
Forth	2 eproms)	
Pascal	2 eproms)	
Word Processor	1 eprom)	Appendix 2
Micro Assembler, Disassembler	1 eprom)	

An Eprom expansion board is available so that all languages can be installed at once. Otherwise you should install the language you wish to use - (refer to Appendix 2).

Any of the programs in your Menu can be activated by typing the first letter of the entry. The program once entered should reply with some form of prompt. The monitor will reply with a > , Basic with ready etc. A list of some of the available programs on eprom and some on cassette which will have Menu entries is given in Appendix 3.

Once you have entered one of these programs the method of returning to the Menu varies and can be found under the documentation for that particular program. However, you can nearly always over-ride this by hitting the Panic Button on the Pegasus Board, to return to the Menu.

When in the Menu all keys not representing a Menu entry will be ignored. In the following discussion about the keyboard it is best to enter Basic by typing B in order to experiment.

The Pegasus Keyboard

The keyboard can generate any character in the Ascii character set. After Power up the upper case alphabetic characters are active and the lower case characters are obtained using the shift key. However, by tapping the caps lock key the lower case letters become the normal characters and the shift key is used to get upper case. Tapping the caps lock key again will take you back to the normal power mode.

The ESC key (Escape) has only one function and that is to halt the output of text to the screen. Tapping it again will resume the output. Alternatively, The Break Key can be typed. In most programs this will also cause a termination of the output as well. Here is a program in Basic to illustrate this. If you are in the Menu and you have Basic installed type B to enter basic and get the "ready" prompt. Now type in the following. You may use the BACK SPACE key if you make an error. Type RETURN when it is all in.

```
10 PRINT "HELLO": GOTO 10
```

If you get an ERR 4 message try again.
Now type RUN. Again if you get an ERR 4 message something is wrong. You should have lots of "HELLO"s appearing at the bottom of the screen and scrolling up. Now type the ESC key. The "HELLO"s will stop. Typing ESC again will allow the output to continue. Type ESC once more to stop output and then type the BREAK key. This will start output but immediately stop the basic program and return you to the basic prompt. This is typical of the way most languages will use the BREAK key.

On the bottom right hand side of the keyboard there are two unmarked keys. They have functions which are used in a typical way in Basic. Firstly tapping the right hand blank key will cause characters typed in to become inverted video on the screen. Tapping it again will revert to ordinary video. This key is called INVFLAG. The left blank key when tapped causes control codes typed in to be displayed as Greek characters, instead of performing their normal functions, (which are described later). Tapping it again reverts to normal mode. This key is called RAWFLAG.

Finally the REPEAT key on the Pegasus is not used. All keys which produce Ascii codes (i.e. all except ESC, BREAK, INVFLAG and RAWFLAG), have an automatic repeat feature. After being held down for approximately one half second the characters will appear repeatedly at the rate of 8 characters per second until the key is released.

Another feature of the Pegasus Keyboard is 2 - key rollover. This means that a key can be pressed before the previous one is released and is essential for fast typing.

Control Codes

The Pegasus Control Codes are invoked by holding the control key down and then typing one of the alphabetic characters. They have various functions which may not appear useful to you at this stage. Basic is again the best way of experimenting with what they do. Most of them are more useful in programs than for use by typing at the keyboard.

CTRL	A	Set inverse video mode
CTRL	B	Clear inverse video mode
CTRL	C	Disable graphics display
CTRL	D	Move cursor right
CTRL	E	Move cursor Up
CTRL	F	Turn cursor On
CTRL	G	Enable Graphics Display
CTRL	H	Same as BACK SPACE
CTRL	I	Same as TAB
CTRL	J	Same as LINEFEED
CTRL	K	Set cursor position
CTRL	L	Form Feed
CTRL	M	Same as RETURN
CTRL	N	Scroll Screen Up
CTRL	O	Turn Cursor Off
CTRL	P	Turn on Clock
CTRL	Q	Turn off Clock
CTRL	S	Move Cursor Left
CTRL	T	Home Cursor
CTRL	U	Clear Line
CTRL	V	Scroll screen down
CTRL	X	Move Cursor down
CTRL	Y	Read Graphic Character
CTRL	[	Program Graphics Character (not avail from keyboard)

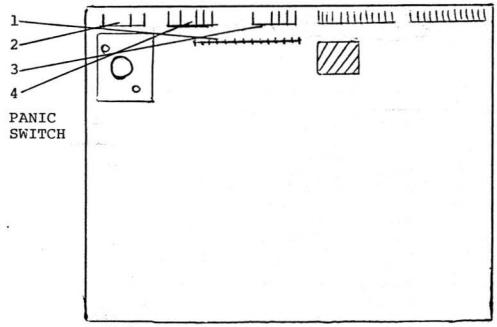
APPENDIX 1

HOW TO START YOUR PEGASUS

Check your equipment:

1 Pegasus
1 Power supply
1 RF Modulator
1 Keyboard
1 TV

Plug positions:



1.	Keyboard plug :	plug in so metal chips face inwards.
2.	Power supply plug :	you can only plug this in one way
3.	Cassette recorder plug :	you can plug this in one way only.
4.	RF modulator plug :	you can plug this in one way only.

NOTE: On cassette recorder interface cable, blue is microphone and yellow is ear.

PLUGGING INTO TELEVISION

1. Turn off television.
2. Remove aerial.
3. Replace with modulator connections.
4. Turn on television and turn to channel 3 or 4.
5. Turn on Pegasus.
6. Tune in television by the use of the fine tuner. If this does not work, try channels 1 to 4 on the television in turn, adjusting first the television fine tuner then by the use of a small flat screwdriver through the hole in the modulator, the modulator tuning. BEWARE, THIS IS SENSITIVE.
7. Look for prompt. and menu.

Aamber Pegasus 6809

B BASIC

M MONITOR

8. To return to the Menu, hit the "Panic" button.

HINTS FOR CASSETTES

Cassettes can be a difficult media to use for data storage, especially to get working on a wide range of quality in equipment and tapes. The following information may be helpful to those experiencing difficulties.

1. Keep recorder away from TV set when saving or loading as it may pick up interference which will degrade performance.

2. Volume setting on playback is usually not critical and should be set between half and full volume. The correct setting for your recorder will be found with experiment. Try three or four positions in this range to get the best results. If you cannot get good loads relatively quickly it is likely something else is wrong. If the load operation never returns with a message it generally means that the volume is so low that it will not trigger the input circuitry. Load Aborted or Checksum errors may mean the volume is either too high or too low.

3. If you suspect the cassette you are trying to load has an inverted recording, see special sheet.

4. Cassettes which have excessive wow may give checksum or load errors, especially if the file was recorded in that condition. Try firmly squeezing the cassette sides and resaving the file. Long cassettes such as C60 or C90's are particularly prone.

5. Sometimes noise on the tape prior to the file can cause a load abort before the file is even reached. If this happens position the tape close to the file before typing load. N.B.: When the Pegasus records a file it first records 10 seconds of silence followed by four seconds of ones, followed by the file data. The cassette can be positioned anywhere before the file data. Generally the Pegasus does not mind starting loading well before the file, even within a previous file. It starts loading data when it finds the first file header.

6. The new leads can be used on old boards by plugging it in upside down.

7. If you still cannot get reliable success, please contact us as we also would like to know why.

CASSETTE RECORDERS

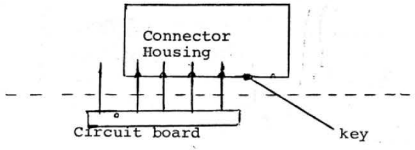
It has come to our attention that some cassette recorders invert the data put into them on record and playback. While this has no effect on programs recorded and played back on the SAME cassette recorder, it can cause problems when tapes are recorded on one machine and played back on another. In our experience this problem does not occur very often, but if it does, the following procedure should be adopted:

If a tape obtained from someone else (THAT HAS NO SECURITY) fails to load on another recorder at the normally good volume setting then you should:

1. Unplug the cassette lead from the Pegasus (and unplus 'MIC' from the recorder.
2. Reinsert it as shown below.
3. Try to load the tape again as normal.

Note: This will in no way improve cassette reliability or help with faulty tapes or recorders - it will only assist with the rare situation where the recorder inverts the data.

4. Once a correct load has been obtained the cassette lead must be replaced in its normal position and the program saved again on another tape. This new tape will then work with that recorder without problems. Save will not work unless the lead is in its normal position.



NOTE: Diagram shows cassette lead plugged in 'upside down and shifted two holes to the right.

Appendix 2 - EPROMS

Eprom need considerable care when handling since they are delicate and expensive devices.

Firstly, always remove the mains power before changing or installing eproms.

Secondly, made absolutely sure that you install the eproms the correct way round. Applying power to an eprom that is plugged in the wrong way round will instantly destroy it.

A screwdriver makes an excellent lever to gently pry the eprom loose without bending any of its pins. When installing eproms you must make sure that all the pins locate correctly into the socket and do not bend out or underneath. All eproms must be aligned the same way - that is with the small notch facing the edge of the board.

The monitor should never be removed and always occupies socket 3. Sockets 1 and 2 may be used for any of the following (Socket 2 is the middle socket). Relocatable means that the eprom will work in either socket 1 or socket 2.

Language	No. of Eproms	Sockets
BASIC	1	Relocatable
FORTH	2	A in socket 1, B in socket 2
PASCAL	2	A in socket 1, B in socket 2
XBASIC	2	A in socket 1, B in socket 2
WORDPROCESSOR	1	Relocatable
MICRO ASSEMBLER DISSEMBLER	1	Socket 1
DEBUG PACKAGE	1	Relocatable

The eprom expansion board removes the need to change eproms and allows all languages to be in the Menu at once. (software selectable)

APPENDIX 3

MENU ALLOCATION

A - Assembler
B - BASIC
C -
D - Disassembler
E -
F - FORTH
G - Galaxy Battle
H -
I - Invaders
J -
K -
L -
M - Monitor
N - Network
O -
P - PASCAL
Q -
R -
S -
T - Tank
U -
V -
W - Word
X - Extended BASIC
Y -
Z -

***** THE MONITOR *****

When your Pegasus powers up, it has amongst its Menu items the "MONITOR". Only a small part of it becomes active when you type "M" and "enter" the monitor. Most of it is actually in constant use by your Pegasus. It is the heart that keeps your Pegasus beating. Some of the functions which the monitor performs are listed below:-

1) Display the Menu and jump to the program selected
2) Scan the keyboard and return Ascii codes to programs
3) Send characters to the screen, and look after the cursor and control codes
4) Perform Video Scanning
5) All cassette loading and saving routines
6) Graphics programming and reading
7) Monitor Commands (e.g. view and change memory)
8) Keeps System Clock
9) Clears all memory on power up and determines memory beginning address
10) Provides routines for outputting and inputting text strings and numbers
11) Vector interupts through ram

You do not need to understand anything about the Monitor in order to use Pascal, Forth or Basic. However, if you are interested in microprocessors at the machine level or how your Pegasus works then the following will be of interest to you.

USER SUBROUTINES

There are several subroutines provided in the Monitor exclusively for use by the USER. Basic for instance uses the routines extensively, and any machine programs you write may use them too. In order that your programs (and ours) are compatible with any version of the monitor, all these routines have pointers to them. The pointers are always at a fixed location even if the routines themselves are not. To use the routines you should use indirect addressing, e.g. JSR (ECHO) which assembles to: (AD 9F F8 ØØ). Appendix 1 contains a description of all such subroutines along with its respective pointer address. By far the biggest of these routines is ECHO. ECHO effectively simulates an output terminal with full cursor facilities, scrolling and many other functions. All the high level languages on the Pegasus use ECHO for output. The next section describes ECHO in more detail. Appendix 2 gives a summary of ECHO's control codes. ECHO's partner, so to speak, is the routine INPUT. By contrast it is very short but it is where the Pegasus spends a lot of its time, waiting for you to type a key.

THE USER SUBROUTINE ECHO

Due to the complexity of this subroutine a full section is devoted to its use. ECHO receives characters in the A accummulator and puts them on the screen at the current cursor location. The cursor is then moved one position. All Ascii characters between $20 and $7F and between $AØ and $FF are treated in this manner. If the Most significant bit is set the character will be displayed with inverted video unless the invert video flag INVERT is set. Characters between $00 and $1F and between $80 and $9F are control characters. They will perform a special function unless a flag called RAWMODE (see monitor system Ram - Appendix 7) is set when they will be displayed as Greek or special characters as per the table in Appendix 4. This makes all the shapes in the character generator ROM available for display through ECHO. However, the RAWMODE flag must be set and cleared directly by USER programs. A suitable way to do this in Basic for instance is to use POKE.

The control codes themselves are summarised in Appendix 2, along with their hexadecimal and decimal equivalents. Control A and B set and clear the invert video flag INVERT. The normal value of this flag is $80. Control C and G take the Pegasus in and out of graphics mode. In other words the normal character generator ROM is substituted for a RAM character generator (if installed). Before this is done, however, it is necessary to program the RAM character generator with the desired character shapes using control [(ESC) See Graphics Section. The use of the ESC code here is not to be confused with the ESC button of the keyboard. When you type ESC at the keyboard it does not generate the ascii ESC code. It just halts ECHO internally. When ESC code is sent to the ECHO subroutine it causes ECHO to use the next 17 bytes sent to it for programming a graphics character.

Control D,E,S and X move the cursor right, up left and down respectively, while control F and O turn it on and off, by clearing or setting a system byte called cursoff. The cursor can be made non-flashing by storing $01 into CURSOFF although no control code will do this.

Control H,J and M are equivalent to BACKSPACE, LINEFEED and RETURN respectively and all perform the expected function. There is no automatic Linefeed on carriage return. The tab character will advance the cursor 6 spaces although USER programs can process them in a more sophistocated manner if required, e.g. the Word Processor.

Control K followed by two further calls to ECHO will set the cursor position to the X and Y co-ordinates received.

Control L performs the form feed function and

Control T performs the cursor home function.

Control N and Control V can be used to scroll the screen up or down without affecting the cursor position.

Control U clears the line that the cursor is currently on.

All cursor movements past the left or right margin of the screen normally result in it re-appearing at the other margin on the next line above or below. This can be prevented by clearing the flag AUTOLN.

Finally, the two control codes P and Q can be used to turn on and off the display of the system clock.

THE SYSTEM MONITOR

Your Pegasus System Monitor resides in a 4K EPROM, starting at address $F000. (See Memory Map.) The system assumes that a minimum of 1K of RAM exists, from $BC00 to $BFFF. In most Pegasi, there will be 4K, from $B000 to $BFFF. When the Pegasus first powers on, a memory test is done that will establish where the system's contiguous RAM resides - contiguous means that there are no gaps in it. (Please note that the Pegasus simply will not work unless there is a system monitor EPROM inserted correctly in its socket.) The beginning and end addresses of user RAM will be derived, and stored in RAM at locations MEMBEG ($BDF2) and MEMEND ($BDF4) respectively. The value in MEMEND will usually be $BD9E, which is the normal end of user RAM (and also the system stack origin), while the value in MEMBEG will vary depending upon the amount of RAM installed, typically being $B000.

VIDEO RAM AND STACK

There are 512 bytes ($0200) reserved for the video display RAM, from $BE00 to $BFFF. Video RAM is organised as 16 lines of 32 characters each, where each character has 8 bits. The top left corner character position equates with $BE00, while bottom right is $BFFF. Characters in this RAM will be displayed on screen as their ASCII equivalents (unless graphics mode is operational), with the MSB indicating reverse video - if 1 (high), then characters will appear as normal, light on a dark background, while a 0 (low) here will cause the character to be inverted. You may modify video RAM directly under program control, but this is not recommended unless you know exactly what you are doing.

The system stack is a collection of bytes that starts at $BD9F and grows downwards towards low memory. The stack is maintained by the stack pointer register, and is used for controlling program flow and subroutine linkage. The amount of memory used by the stack varies, but is typically less than 30 bytes. Note that monkeying with memory near the stack (.....$BD9F) can cause your system to crash.

SYSTEM VARIABLES

There are 96 bytes permanently reserved for special use, occupying addresses from $BDA0 to $BDFF inclusive. These are byte and word variables, that are used by the monitor for various housekeeping functions, many of which are available to the user. Some of these locations are byte flags, or booleans, and may be changed or tested by user programs, while others are reserved for system use, so that changing them could cause unpredictable results to occur. A summary of these variables can be found in Appendix 7.

The system monitor may be entered with 'M' for 'Monitor'. The prompt used for monitor commands is the greater-than symbol, '>'. A summary of monitor commands is given below:

G hhhh
: GO function - machine language programs may be executed simply by typing the desired starting address. If you wish execution to occur, type the '.' key; any other key typed will abort back to the monitor.

L hhhh
: LOAD function - the load operation will read data from the cassette interface into RAM. The data loaded will be a contiguous region of memory, and the load start and end addresses will be displayed, along with the filename that was used when saving the file. A new load start address may be specified to relocate the file on loading.

V
: VERIFY - this is similar to load except that data read from the cassette will not alter the appropriate locations in memory. This is useful for verifying that a program has SAVEd or LOADed successfully.

M hhhh
: MODIFY function - a memory address is specified, and the byte found there will be displayed. You may proceed to the next byte with the '.' key, and move to the previous byte with the ',' key. A new address may be specified by typing the 'M' key. At any stage, the byte at a RAM location may be changed, simply by entering a new, two digit hex value. If the location was not changed, then the contents of the location are redisplayed.

T hh mm ss
: The time may be set using the TIME function. The time is measured in hours, minutes and seconds. Note that ver 1.2 has a 24 hour clock system.

S hhhh hhhh

SAVE - a block of data between the given start and end addresses will be saved on cassette tape. A filename may be specified. The screen will blank for the duration of the SAVE operation.

MENU Items

If you want programs which you have written and have in memory to appear in the Menu when you go back to it, the following convention should be used.
Starting on a 1K boundary assemble the following -

```
        BRA   CONT
        FCB   "Name to appear in Menu", $ØØ
        FDB   $ØØØØ
CONT.
```

When the Pegasus writes the menu it searches all the 1K boundaries for a branch opcode followed by an offset of $20 or less. A Message of up to 29 characters may follow. It must be terminated with three $ØØ bytes.

Appendix 1

USER ACCESSIBLE PEGASUS SYSTEM MONITOR SUBROUTINES

The Aamber Pegasus System Monitor EPROM contains a number of machine language subroutines that may be found to be of use. Each routine is called via a table of addresses, using the 6809 indirect addressing mode. E.g. ECHO is called with AD 9F F8 00.

ECHO $F800 F53A
ASCII character in A reg. is output to video display. All registers are preserved. Can be revectored. See Appendix 7 under ECHOVEC.

INPUT $F802 F549 F54D for BASIC
Waits for key to be typed, returns ASCII value 0..7F in A reg. Character is not echoed. Can be revectored. See Appendix 7 under INPUTVEC. All registers preserved except A,CC

INKEY $F804 F559
Scans keyboard for keystroke. If key found, it will be returned in A reg. with carry clear - if not found, then carry will be set and A will be undefined. Character not echoed. All registers preserved except A,CC

PROMPT $F806 F15C
Re-entry point into machine language monitor

CASSOUT $F808 F8C4
Outputs data to cassette recorder with start and end addresses in locations START and FINISH. (See Appendix 7). Record button on recorder must be going. All registers preserved except A and B

CASSIN $F80A F9B3
Reads the next file from cassette tape, assuming that the play button has been pushed. Leaves load address of file in START and end address in FINISH - See Appendix 7. All registers preserved except A, B and CC

TRIMCASE $F80C F890
Character in A reg. is converted from lower case to upper case. Also, '.' becomes '<' and ',' becomes '>'. All registers preserved except A, CC

HEXOUT $F80E F8A5
Takes number in A reg., displays as two hex digits. All registers preserved except A,CC

HEXDIG $F810 FBC4
Takes digit 0..F in A reg, converts to ASCII '0'..'F'. All registers preserved except A,CC

GETBYTE $F812 FC15
Reads two hex digits from keyboard, echoing if valid, returns 8 bit number in A reg. Carry set if not valid. All registers preserved except A,CC

GETLINE $F814 FC84
Inputs line into buffer pointed to by X reg. on entry. Buffer size given by A reg. Line terminated with RETURN key (echoed as CR,LF pair). Characters are echoed as they are typed, and may be corrected with the BACK SPACE key. The control -U function will clear the buffer. On return, the A and X regs. are preserved, and B contains a byte count of characters typed. The RETURN will not be in buffer. All registers are preserved except B and CC

TEXTOUT $F816 FC4C
Output string pointed to by X reg, terminated with nul ($00). The X reg. is left pointing to the byte after the nul. All registers are preserved except A and CC.

GRAFIX $F818 FF67
See discussion of Pegasus graphics capabilities. All registers preserved except X.

GETDIG $F81A FBE7
Reads and echoes a decimal digit from keyboard, returning in A reg, carry set if non-numeric digit was typed. All register preserved except A,CC

GETNUM $F81C FBFD
Reads and echoes decimal number range 00 to 99 from keyboard, returns in A reg. Non-numeric means carry set. All registers preserved except A,B,CC

CONVERT $F81E FC6A
Takes decimal number, range 00 to 99 in B reg., converts to two ASCII characters in the D reg. (A and B regs.) All registers preserved except A,B,CC

GETWORD $F820 FC3A
Gets four hex digits from keyboard, echoing them. Returns number in X reg. Carry set if non-hex typed. All registers preserved except A,B,X,CC

RETMENU $F822 FOA0
Entry point back to Menu selection mode in Monitor

HEXTEST $F824 FBCD
Tests ASCII char. in A reg. for range '0' to 'F'. If not in range, returns with carry set, else digit returns in A. All registers preserved except A,CC

NEWLINE $F826 FE0F
Causes cursor to move to start of next line. This routine simply calls ECHO, firstly with CR then with LF. All registers preserved except A,CC

FORMFD $F828 FE41
Equivalent to control-L: clears the video screen. Does not call ECHO. All registers preserved except CC

DELINE $F82A FED3
Clears line that cursor is on - same as control-U Does not call ECHO. All registers preserved except CC.

SPACER $F82C FC56
Outputs one space to screen at current cursor position. All registers preserved except CC

HOMEUP $F82E FEC9
Homes cursor to top left corner - same as control-T. All registers preserved except CC

NUL $F830 FC A9
Nul function - has no effect

LINEOUT $F832 FC49
Similar to TEXTOUT, except that a NEWLINE is done first.

MONITOR - USERS NOTE

The initialization done by the monitor at various times has been designed to allow easy recovery from crashed programs.

POWER UP - initializes everything and clears memory

RESET - initializes everything except MEMBEG, MEMEND and PC-REG

RETURN TO MENU OR ENTERING MONITORS - initializes only the system variables which control the keyboard and display. These are MSBINV, RAWFLAG, CAPSLOCK, KEYLOCK, RXREADY, ABORT (break) vector; INVERT, RAWMODE, CURSOFF, PAUSING, LINES, AUTOLN and clears graphics mode and clears the F interrupt mask bit. This gives a standard keyboard and display while leaving all your other variables, vectors and PIA's etc untouched. While Reset is the No. 1 tool for recovering from crashed programs, a separate NMI button is sometimes useful as a way of getting back to the Monitor prompt without initializing everything back. The NMI interrupt is vectored to take you straight into the monitor prompt. A prime exmaple is when you have your own printer drivers installed in RAM and you have revectored the printer routines.

If you do a reset it will change these vectors back to the centronics routines. Your routines would still be safe however because Reset does not change MEMBEG or MEMEND and the stack always starts at MEMEND (Version 2-3).

The NMI interrupt can be used for other purposes because it is vectored through RAM at location $B002

Anytime you are in the monitor you can cause the reset initialization to run by typing "R".

The printer routines can be used from programs as they have vectors pointing to them at $F834 and $F836. The first routine is the initialize printer routine and the second is the output character to printer routine. Both routines save all registers except CC. A carry set condition is returned if the printer is not ready.

You may write your own printer drivers because these routines are vectored through RAM at locations $BDC9 and $BDCB. You may change $BDC9 to point to your printer initializing routine and $BDCB to point to your output character routine. Then you will be able to use all printer commands in XBASIC and those in ECHO for your own printer. Your routines must save all registers. The characters to be printed is in A and you should return a carry set condition if your printer is not ready.

Appendix 2

ECHO CONTROL CODES

Hexadecimal value	Decimal value	Control character	Separate key	Meaning
00	0	@		Null
01	1	A		Set inverse video mode
02	2	B		Clear inverse video mode
03	3	C		Disable Graphics Display
04	4	D		Move cursor right
05	5	E		Move cursor up
06	6	F		Turn cursor on
07	7	G		Enable Graphics Display
08	8	H	BACK SPACE	Back Space
09	9	I	TAB	Horizontal Tab
0A	10	J	LINE FEED	Line Feed
0B	11	K		Set cursor position (followed by X and Y)
0C	12	L		Form feed (clear screen)
OD	13	M	RETURN	Return cursor to left margin
OE	14	N		Scroll screen up
OF	15	O		Turn cursor off
10	16	P		Turn on digital clock
11	17	Q		Turn off digital clock
12	18	R		No function
13	19	S		Move cursor left
14	10	T		Home cursor to top left
15	21	U		Clear line that cursor is on
16	22	V		Scroll screen down
17	23	W		No function
18	24	X		Move cursor down
19	25	Y		Read Graphic character
1B	27	[	ESCAPE	Program graphics character

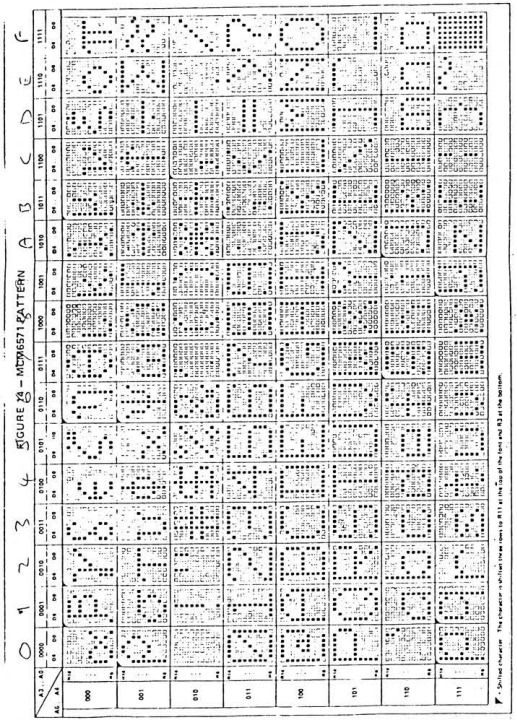
APPENDIX 3

ASCII CHARACTER CODES

1968 ASCII American Standard Code for Information Interchange. Standard No. X3.4 -1968 of the American National Standards Institue

$b_3 b_2 b_1 b_0$		b_6 0	0	0	0	1	1	1	1
		b_5 0	0	1	1	0	0	1	1
		b_4 0	1	0	1	0	1	0	1
		0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	A	LF	SUB	*	:	J	Z	j	z
1 0 1 1	B	VT	ESC	+	;	K	[	k	{
1 1 0 0	C	FF	FS	,	<	L	\	l	|
1 1 0 1	D	CR	GS	-	=	M	]	m	}
1 1 1 0	E	SO	RS	.	>	N	^	n	~
1 1 1 1	F	SI	US	/	?	O	_	o	DEL

APPENDIX 4



APPENDIX 5

Memory Map for Aamber Pegasus - June 1981

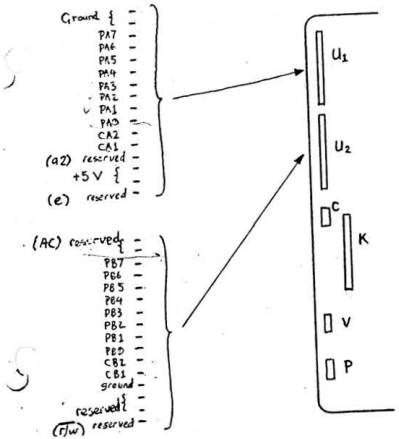
F000..FFFF	System Monitor, in EPROM Socket 1	
E818..EFFF	Reserved for future expansion	
E810	Port 4 General Purpose user interface	Requires I/O Expansion Option
E808	Port 3 Digital Cassette interface	
E800	Port 2 Printer Port	
E400	Port 1 Calculator chip interface - on board PIA	
E600	Port 0 System PIA - reserved for use by Pegasus only	
E200	Programmable Character RAM - not directly addressable	
D000..DFFF	Reserved for	
C000..CFFF	future expansion	
BE00..BFFF	512 bytes for video RAM	4K on-board RAM
BDC0..BDFF	64 bytes for System Monitor variables	
B000..BDBF	Standard User RAM, stack = $BDBF B400 - BBFF = TEMP. CHARS. STORAGE	
A000..AFFF	36K of address space reserved for RAM expansion under DAT control	
9000..9FFF		
8000..8FFF		
7000..7FFF		
6000..6FFF		
5000..5FFF		
4000..4FFF		
3000..3FFF		
2000..2FFF		
1000..1FFF	EPROM Socket 2	
0000..0FFF	EPROM Socket 3	

Room Expansion
5000 - AFFF -
- B00

APPENDIX 6

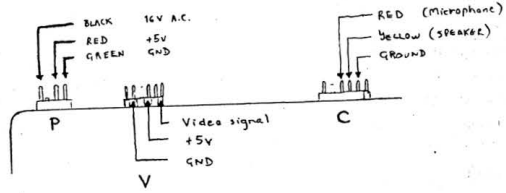
USER P I A CONNECTIONS

(unmarked pins reserved for future use)



Key to Abbreviations:

- U_1 Optional PIA plug 1
- U_2 Optional PIA plug 2
- C Cassette interface plug
- K Keyboard plug
- V Video signal plug
- P Power



Appendix 7

SYSTEM MONITOR 2.0 RAM ALLOCATION

The Aamber Pegasus Monitor reserves 96 bytes of Read/Write memory address range $BDA0 to $BDFF for system housekeeping functions. A summary of useful locations is given below.

$BDC0 INBUFF

The keyboard uses the byte at this location as a single character buffer, before passing the character on to the input subroutines. Whenever a code is generated while the keyboard is enabled, then the character will be deposited in INBUFF, and the RXREADY flag will be set TRUE (non-zero). A sample routine to use the keyboard as an input device is given below:- This routine is taken straight out of the monitor.

```
INPUT    TST RXREADY   .  Check the flag
         BEQ INPUT     .  Loop if FALSE
         LDA INBUFF    .  Get the character
         CLR RXREADY   .  Clear flag for next use
         RTS           .  End of subroutine
```

The keyboard input will not work if FIRQs are disabled. If the character is not read from the buffer by the controlling program before the next key is typed, then it will write over the character that was there. Any ASCII character in INBUFF will be in the range $00..$7F. characters with the MSB set will be implemented for special functions in later Monitor releases.

$BDC1 TICKS

While FIRQs (Fast Interrupt ReQuests) are enabled, an interrupt will be generated every 20 milliseconds, which will be used (among other things) for incrementing the bytes at TICKS. The byte will be cleared every 50 interrupts, providing a mains-derived 1 second clock.

$BDC2 TOCKS

The byte is similar to TICKS, except that it varies from 0 to 255 in 20ms intervals, and is divided by powers of 2 to provide timing synchronisation for system housekeeping functions.

$BDC3 SECONDS

Derived from TICKS, this byte varies from 0 to 59, and provides the seconds value for the system time, which may be set from the monitor using the 'T' function.

$BDC4 MINUTES

This byte contains the minutes part of the system time, varying from 0 to 59 as the time proceeds.

$BDC5 HOURS

This value changes every hour and forms a 24 hour clock that varies from 0 to 23, where 12:00 is midday.

$BDC6 00 PRINTER ON/OFF (02:ON)
Reserved

$BCC7-8 POC

This value is always $55CC. If it is not then a Reset will cause all of memory to be cleared.

$BDC9-C FCF7 FDIE
Reserved
PRINTER VECTORS

$BDCE-F

Program counter used by GO command
CENTRONIC ROUTINE

$BDD0-3 6400 FISC
Reserved NMI VECTOR

$BDD4 INVERT

This byte defines the invert state of characters as they are output to the video screen. When this byte is zero, characters ECHOED will appear dark on a light background; when this byte contains $80, they will appear light on a dark background. INVERT is cleared by the control-A code, and is set by control-B.
Note that its normal state is with invert TRUE ($80).

$BDD5 CLOCK

This byte controls the digital clock display, which will appear at the upper right of the screen. The clock is turned on by typing control-P at the keyboard, while control-Q will turn it off.

$BDD6 CURSOFF

When Minus (MSB set), the cursor is turned off. When zero the cursor is on (flashing) and when plus the cursor is on non flashing. Control 0 and Control F when echoed to the screen will turn the cursor off or on resp. Once turned on the cursor can be made non flashing by storing a $01 in this byte. It is recommended that when updating the video RAM directly, the cursor should be turned off using the control-0 function, and then turned on again using control-0.

$BDF4 MEMEND BD9E

This location is used to store the logical end of the user's read/write memory. The system stack usually originates here. The value stored in MEMEND by the system is $BDBF, but this value may be changed by user programs.

$BDF6 ABORT F0A0

When the BREAK key on the keyboard is depressed, then the routine specified by the contents of this address is called as a subroutine. This is usually used in user programs to set a flag that tells the program to stop execution. The user routine must be very short and must not use direct addressing.

$BDF8 LINES

The number of lines displayed on the screen is controlled by this byte, which must always be in the range of 0 to 16. Due to the nature of software scanned video generation, a reduction in the number of displayed lines will cause a significant increase in execution speed for all programs.

$BDF9 XPOSIT

The horizontal cursor position is maintained in this byte, and should always be in the range of 0 to 31. It is preferred that any changes to XPOSIT or YPOSIT be made through use of the appropriate control codes, rather than through writing directly into them. If you want to write to them directly the cursor must be turned off while you do this.

$BDFA YPOSIT

Vertical cursor position is stored here, which is a number in the range of 0 to 15.

$BDFB Reserved for system 20

$BDFC INPUT VECTOR F543

The sytem keyboard input routine is vectored through this location, and may be re-vectored to a user routine. If these locations are corrupted, then the keyboard will stop working until an NMI occurs.

$SBDE4 FINISH

These two bytes are reserved for storing the cassette file ending address, when saving or after loading

$BDE6 RELOC

This flag specifies, when non-zero, that a file being loaded in is to be relocated to the address that is stored in START.

NB. FF Means relocation. Ø1 here means that a cassette verify is in effect.

$BDE7 AUTOLN FF

When this flag is true, any sideways cursor movement from either edge of the screen will cause the cursor to move to the next line above or below.

$BDE8 RAWFLAG

This flag is toggled by the left-hand unlabelled key. In the monitor this key is used in the GETLINE routine to put control codes into the buffer, and echo them as rawmode characters.

$BDE9 Reserved for system 00

$BDEA IRQVECTOR F57B

Whenever an IRQ is received from pin 3 on the 6809 microprocessor then program execution is vectored to the address stored at this location.

$BDEC SW11 VECTOR F576

The SW1 (SoftWare Interrupt) instruction will cause program execution to transfer to the address stored in these two bytes.

$BDEE SW12 VECTOR F573

The SW12 instruction will vector though here.

$BDFO SW13 VECTOR F570

The SW13 instruction will vector through here.

$BDF2 MEMBEG

When the system is reset, a memory test (non-destructive) is executed that determines the beginning address of Read/Write memory. This address is placed here in MEMBEG.

$BDD7 Reserved by system

$BDD8 PAUSING

When TRUE, this byte will cause output to the screen to be suspended. This function is turned on and off using the ESC (escape key, ASCII 27), and is used for stopping print outs.

$BDD9 RAWMODE

When non-zero, this byte will cause any control codes that are echoed to be printed as a special character, without executing the appropriate control function. This byte must be control directly as there are no control codes to turn it on or off.

$BDDA REREADY

This flag indicates (when TRUE, or non-zero) that a character is ready in the INBUFF character buffer. This flag should be cleared after use.

$BDDB to $BDDD Reserved for system use only.

$BDDE KEYLOCK

The keyboard may be locked out by setting this flag TRUE.

$BDDF CAPSLOCK

This byte will invert the sense of the shift key for the letters A..Z on the keyboard, allowing upper or lower case to be used. This flag is toggled by the CAPS LOCK key towards the left side of the keyboard.

$BDEO Reserved by system

$SBDEI MSBINV

This is a general purpose toggle flag that is set and reset by the unlabelled key at the extreme lower right of the ASCII keyboard. In the Monitor this key is used to invert the most significant bit of characters typed on the keyboard that are read using the GETLINE system subroutine - for instance, Tiny BASIC.

$BDE2 START

Two bytes are reserved here for storing the starting address of files that are saved using the cassette. When files are loaded from cassette, then the starting address is placed here.

$BDFE OUTPUT VECTOR

The system output routine, ECHO, is vectored through this location, and can also be re-vectored if desired. This feature is used for things like outputting information to printers or other devices.

APPENDIX 9 MONITOR 2.3

PRINTER ROUTINES

Monitor 2.3 has printer routines which will drive a centronics parallel standard interface. Extended basic uses these routines directly.

The ECHO routine in the monitor can be configured to use these routines also. This means that you can record everything you do; make Tiny basic listings etc. Three control codes are used to set up this printing mode. Control J (hold both the control and shift keys down and press [2]) is used to enable the printer to print everything sent to ECHO as well as having it appear on the screen. Control ^ sets the printer mode on while disabling the normal screen display. Control _ (underbar) is used to revert back to normal output with printer off. All these codes may be typed in directly from the keyboard or can be sent from USER programs.

A BEGINNERS GUIDE TO

PEGASUS BASIC

INTRODUCTION

This book will teach you how to communicate with your Pegasus Computer. You will learn how to speak its language so that by giving it meaningful instructions you can make it do what you want it to do. That's all programming is, by the way.

There are many computer languages. Your Pegasus understands a language called PEGASUS BASIC which is a simplified form of BASIC (BASIC stands for Beginner's All-purpose Symbolic Instruction Code).

BASIC is perhaps the best language for the beginning micro-computer programmer. It is easily learned (as you will soon see) and programs may be developed quickly. For the more experienced programmer BASIC can form the basis of a system whose sophistication may be indefinitely extended.

So let's get started. Get to know your Pegasus. It can do an infinite number of things for you.

Chapter 1

GETTING STARTED

In this chapter we will introduce you to your Pegasus. You will learn how to use your keyboard and how to control the output display on your TV screen.

Connect your computer by referring to the appropriate section in your Computer Operation Manual.

Switch it on and you will be greeted by the following heading on your television screen:

```
AAMBER Pegasus 6809
Technosys Research Laboratories
Basic      1.0
Monitor    1.0
Select one of the above:
```

Press T and your Pegasus will be 'ready' to go.

Do you see the flashing light? This is called the cursor and it indicates to you where on the screen the characters you type in on the keyboard will be displayed.

Try it. Type the following exactly as shown below:

```
PRINT    "HI, I'M YOUR PEGASUS COMPUTER"
```

When you reach the end of the line on the screen, keep on typing. The last part of the message will appear on the next line automatically. (Notice that the screen can display a maximum of 32 characters.)

Now check your line. Is it alright?

If you made a mistake, no problem. Simply press the BACK SPACE key and you will observe the last character you typed will disappear. Press again, and the next will disappear, and so on .....

This is what you should see on the screen:

```
Ready

PRINT  "HI, I'M YOUR PEGASUS COMP
UTER"
```

Now press RETURN. This key tells the computer that you have finished the line. The computer then proceeds to execute it.

Your screen will then display:

```
Ready

PRINT  "HI, I'M YOUR PEGASUS COMP
UTER"

HI, I'M YOUR PEGASUS COMPUTER

Ready
```

As you can see, the computer has obeyed your command and is 'Ready' for more.

Now type:

```
PRINT "2 * 2"
```

and press return. The computer obeys and prints your message:

```
2 * 2
```

How about some answers! Alright, try it without the quotation marks:

```
PRINT 2 * 2  (RETURN)
```

This time the computer prints something different - the answer to the expression 2* 2

Experiment further by typing the following:

```
PRINT     3+4  (RETURN)
PRINT     "3+4"  (RETURN)
PRINT     "3+4 EQUALS", 3+4  (RETURN)
PRINT     8/2, "IS 8/2" (RETURN)
PRINT     "6/2" (RETURN)
PRINT     6/2  (RETURN)
```


This demonstrates that the computer sees everything you type as either strings or numbers. If it is in quotation marks it is a string. If it is not in quotes, it is a number. The computer sees it exactly as it is. The number might be in the form of a numerical expression (e.g. 3+4) in which case the computer reduces it to a single value.

By now it is likely that the computer has printed some unknown messages on your screen. If it hasn't, type the following, deliberately mispelling the word PRINT.

```
PRIINT "HI"  (RETURN)
```

The computer prints: ERROR #4

This indicates that the computer has detected a syntax error. You will have to type the line again properly.

There are other types of errors, too. Try:

```
PRINT 5/0  (RETURN)
The computer prints:
ERROR #8
```

This indicates an impossible division by zero command.

So whenever PEGASUS BASIC detects an error while executing a line it generates an error message. A listing of error numbers and their corresponding meanings is given in Appendix 1.

CHAPTER 2

NUMBERS, VARIABLES AND EXPRESSIONS

Before we go on it is important that we understand the meanings of numbers, variables and expressions.

NUMBERS

PEGASUS BASIC is an integer BASIC, which means that all numbers in it have no fractional part, e.g. 3.7, 4.02 and 3.1415926 are not integers.

Besides this there are two operating modes - signed and unsigned. When you first switch the machine on the computer automatically goes into the signed mode. In this mode integers in the range of -32768 to 32767 are only allowed.

You can change to the unsigned mode by using the USIG statement.

Type: USIG (RETURN)

In this mode integers in the range 0 to 65535 only are allowed. To return to the signed mode use the SIG statement.

Type: SIG (RETURN)

If you input a number outside the allowed range, or the intermediate or final result to a calculation is outside the allowed range, then an error message will be returned.

VARIABLES

In PEGASUS BASIC a variable is represented by a single capital letter (A to Z) which directly corresponds to a location in the computer memory - we call this the name of the variable. The value of the variable is the number stored there.

For example, assign the variable A the value of 13 and the variable B the value of 7 by typing:

```
LET A =  13 (RETURN)
LET B =   7 (RETURN)
```


As LET is used very often in computer programs, the computer will understand you if you leave out the work 'LET' altogether. From now on that is what we will do.

OK. Now have the computer print out your numbers:

```
PRINT A,",",B
```

Notice the use of commas in the print statement. Your computer will remember your assigned values for A and B as long as it is switched on, or until you decide to change them.
Do this by typing:

```
A = 15 (RETURN)
```

Then, when you ask it to print A it will print 15.

EXPRESSIONS

An expression is a combination of one or more numbers, variables or functions joined by operators. You are probably most familiar with the following mathematical operators.

- `+` addition
- `-` subtraction
- `*` multiplication
- `/` division

Lets say that we want to divide the sum of 9 and 6 by 3. You might write this as:

```
9 + 6 / 3
```

Now try it on your computer.

Type: PRINT 9 + 6 / 3 (RETURN)

Is this the right answer to the problem? No, it isn't! This is because your computer has first worked out 6 divided by 3 (that's 2) and added this to 9 to give 11.

This demonstrates the way that the computer works out arithmetic problems. The computer looks first at the expression and does multiplication and division first. Then it does addition and subtraction.

So, to get the computer to solve the problem differently, you must use parentheses.

Type it as: PRINT (9 + 6) /3 (RETURN)

That's better. The computer solves the expression in parentheses before doing anything else.

What will your computer print in answer to the following problems.

```
PRINT  12 - (6 - 4) /2
PRINT  12 - 6 - 4 /2
PRINT  (12 - 6 - 4) /2
PRINT  (12 - 6) - 4 /2
PRINT  12 - (6 - 4 /2)
```

Check by typing them out.
Now see what happens if you type in:

PRINT (12 - (6 - 4)) /2 (RETURN)

If the computer sees a problem with more than one set of parentheses it solves the inside parentheses first and moves to the outside parentheses.
In other words, it does it like this:

```
(12 - (6 - 4) /2
      └──────────→6 - 4 = 2

(12 - 2) /2
    └──────────→12 - 2 = 10

19/2
  └──────────→10/2 = 5
```

Can you imagine any problem with integer division? What is 13/5?

Try it: PRINT 13/5 (RETURN)

It gives 2 which is the whole number part of the result. If you want the remainder use the MOD operation.

Try it: PRINT 13 MOD 5

It gives 3. You can use MOD just like you use *,/,+, and -.

There are other classes of operators available in PEGASUS BASIC besides the mathematical operators - we'll look into these in later chapters.

As stated in the definintion you can also include variables in expressions.

Try it by typing: PRINT A/3 + B (RETURN)

(Remember A was 15 and B was 7). This feature is particularly useful in programs that we shall soon see.

CHAPTER 3

INTRODUCTION TO PROGRAMMING

Type: NEW (RETURN)

This is just to erase anything that might be in the computer memory.

Now type this line (don't forget the line number, 10):

10 PRINT "HI, I'M YOUR PEGASUS COMPUTER" (RETURN)

Nothing happened, did it? What you have done is to type your first program.

Next type: RUN (RETURN)

And now you have just run it. Type RUN again - and yes it runs again. Add another two lines to the program.

Type: 20 PRINT "GIVE ME A NUMBER" (RETURN)
30 PRINT "AND I WILL DOUBLE IT" (RETURN)

Then type: LIST (RETURN)

Your computer obeys by listing the program. Your screen should look like this:

```
10 PRINT "HI, I'M YOUR PEGASUS CO
MPUTER"

20 PRINT "GIVE ME A NUMBER"
30 PRINT "AND I WILL DOUBLE IT"
```

Don't attempt to type in the number because your computer is not ready for it.

Add the line: 40 INPUT T (RETURN)

Add one more line: 50 PRINT "2 TIMES",T," IS ",2*T

Now list again, and your program should look like this:

```
10 PRINT "HI, I'M YOUR PEGASUS CO
MPUTER"
20 PRINT "GIVE ME A NUMBER"
30 PRINT "AND I WILL DOUBLE IT"
40 INPUT T
50 PRINT "2 TIMES",T," IS ",2*T
```


Now run it. The Input statement prompts you with a question mark. Type in a number (integers only, remember, which the computer will label T) and then (RETURN),

Didn't you do well! This is what you should have got (depending on the number, of course);

```
HI, I'M YOUR PEGASUS COMPUTER
GIVE ME A NUMBER
AND I WILL DOUBLE IT
? 9
2 TIME 9 IS 18
```

Run this program a few more times, inputting different numbers.

Now add another line.

Type: 60 GOTO 10 (RETURN)

And run it.....the program runs over and over again without stopping. That last GOTO statement tells the computer to go back to line 10. This is called a loop, and in this program it will cause it to run perpetually. However, you can get out of it by pressing the BREAK key, then any number and RETURN.

Change line 60 so that it goes to another line number. How do we change a program line? Simply by retyping it, using the same line number.

Type: 60 GOTO 50

Your program listing should then look like:

```
10 PRINT "HI, I'M YOUR PEGASUS CO
MPUTER"
20 PRINT "GIVE ME A NUMBER"
30 PRINT "AND I WILL DOUBLE IT"
40 INPUT T
50 PRINT "2 TIMES ",T," IS ",2*T
60 GOTO 50
```

Run it.....OK, press the BREAK key when you have seen enough. There is a more desirable way of getting out of the loop. Why not get the computer to politely ask if you want to end it?

Change line 60 as follows: 60 PRINT "DO YOU WANT IT DONE AGAIN?"

And add these lines: 70 R = INKEY : IF R = 0 GOTO 70

80 IF R = 89 GOTO 20

Then run the program ....type your number... then type Y and the program loops back again. If you type anything else (e.g. 'N'), the program stops.

This is what the program looks like:

```
10 PRINT "HI, I'M YOUR PEGASUS CO
MPUTER"
20 PRINT "GIVE ME A NUMBER"
30 PRINT "AND I WILL DOUBLE IT"
40 INPUT T
50 PRINT "2 TIMES ",T," IS ",2*T
60 PRINT "DO YOU WANT IT DONE AGAIN?"
70 R = INKEY: IF R = 0 GOTO 70
80 IF R = 89 GOTO 20
```

What are these new lines? Line 60 is simply a printed question. Line 70 is in fact two lines, the two statements being separated by the colon ':'. The first part assigns the ASCII equivalent of the key depressed on the keyboard to the variable R. (ASCII is the American Standards Code for Information Interchange). If no key is pressed then R is assigned 0. The second part of the line tests for this condition and loops back to INKEY if it is true. However, as soon as a key is depressed it gets out of the loop and proceeds to the next line...

Line 80 tells the computer to go to Line 20, IF, (and only if) the Y key (That's ASCII 89) has been depressed. If not the program ends as there are no more lines after this.

This chapter has covered a lot of important concepts of PEGASUS BASIC. Don't worry if some of these things are not absolutely clear at this stage. Experiment with your computer, and above all, enjoy it.

CHAPTER 4

MORE PROGRAMMING

In this chapter we will practise using functions and statements in PEGASUS BASIC.

Type this:

```
10 FOR X = 1 TO 10
20 PRINT "X =", X
30 NEXT X
40 PRINT "FINISHED"
```

Run the program. See how it has printed X for X = 1 TO 10. Now replace line 10 with the following:

```
10 FOR X = 5 TO 8
```

And run again. Let's look at the program listing.

```
10 FOR X = 5 TO 8
20 PRINT "X =", X
30 NEXT X
40 PRINT "FINISHED"
```

It's clear that line 10 determines that starting and ending values of the variable X. Line 30 tells the computer to get the next number - the next X - and to jump back to the line following the FOR ... TO ... line. (i.e. line 20) until it reaches the last number. At this stage it goes straight on to execute the final statement. We can further investigate the path of program execution by using the TRON statement. Try it.

Type: TRON (RETURN)

Now run the program again. This statement has turned on a trace, which provides a line number listing for statements as they are executed. The trace should look like this:

```
<10>  <20>  X = 5
<30>  <20>  X = 6
<30>  <20>  X = 7
<30>  <20>  X = 8
<30>  <40>  FINISHED
```

See how the program keeps jumping from line 30 to line 20 until it eventually goes from line 30 to line 40 and stops. To turn the trace off, type: TROFF (RETURN)

If you like, run your program again to see if the race has gone.

An extra feature of the FOR...TO... statement is that you can specify the actual STEP size. Change line 10 to read:

```
10 FOR X = 2 TO 10 STEP 2
```

And run the program. See how X goes from 2 to 10 in steps of 2. Before, when we didn't specify the step size, it assumed STEP 1. What will happen if line 10 is replaced with:

```
10 FOR X = 3 TO STEP 3
```

Try it... and see that it loops back only for X 10

How about: 10 FOR X = 10 TO 1 STEP - 1

Yes, it counts backwards too.

Now try a new program - that's right, type NEW and (RETURN) - the type:

```
10 FOR X = 1 TO 3
20 PRINT "X =", X
30 FOR Y = 1 TO 2
40 PRINT "Y = ", Y
50 NEXT Y
60 NEXT X
```

Run it ... This is what you should get:

```
X = 1
Y = 1
Y = 2
X = 2
Y = 1
Y = 2
X = 3
Y = 1
Y = 2
```

Notice how it loops within another loop. Programmers call this a 'nested' loop.

Now for something completely different. Type in this new program:

```
10 S = RND /26
20 PRINT "GUESS THE NUMBER"
30 INPUT G
40 IF G = S THEN GOTO 70
```



```
50 PRINT "NO, TRY AGAIN"
60 GOTO 30
70 PRINT "YES, THAT'S IT"
```

And run it... guess numbers between 0 and 9 inclusive (the division by 26 in line 10 gives us this range).

The new statement type encountered here is the IF ... THEN conditional statement. The statement tests the expression G = 5 and IF false will skip immediately to the next line; but IF that statement is true THEN it executes the next statement GOTO 70.

This condition is often the result of a relational operation. In BASIC these are;

= equal to	<> not equal to
< less than	> greater than
<= less than or equal to	>= greaten than or equal to

These are often combined with logical operators AND, OR, NOT to perform quite complex tests, here's an example:

```
600 IF A = 0 OR (C 127 AND D   0)  GOTO 100
```

This will cause a branch to line 100 if A is equal to 0 or if both C is less than 127 and D is not equal to zero.

This type of expression essentially evaluates to 0 for false and -1 for true. Besides being used from true/false evaluation, logical operators can operate on binary numbers.

For example, type: PRINT 6 and 7 (RETURN)

This gives decimal 6 which is 0110 ANDed with 0111

So far we have been looking at relatively short programs. Before long you will be so proficient with your Pegasus that you will be writing quite long and complex programs.

We'll now look at some expressions which will help us keep things in order. Type and run the following program.

```
10 PRINT " EXECTING THE MAIN PROGRAM"
20 GOSUB 400
30 PRING "NOW, BACK IN MAIN PROGRAM"
40 END
```

```
400 PRINT "EXECUTING THE SUBROUTINE"
410 RETURN
```

Line 20 tells the computer to go to the subroutine beginning at line 400. RETURN tells the computer to continue execution with the line following the GOSUB expression. The END expression is necessary to separate the main program from the subroutine.

Subroutines are written for operations that are frequently required. They result in economy of effort when it comes to writing programs.

One final point - you can use the REM statement to place remarks and comments throughout your program. Anything following the REM statement is ignored. These remarks are often placed at different points in a program, particularly at the beginning of subroutines, to explain how unclear or complicated sections of the program work.

Here is a final program that illustrates these points. Try it.

```
10 REM THIS PROGRAM RAISES A
20 REM NUMBER TO AN EXPONENT
30 INPUT "NUMBER"N
40 INPUT "EXPONENT"E
50 GOSUB 1000
60 PRINT:PRINT N, " EXPONENT ",E" IS ",A
70 END
80 REM ------------//------------
1000 REM THIS SUBROUTINE DOES
1010 REM THE ACTUAL EXPONENTIATION
1015 IF E=0 THEN A=1:RETURN
1020 A=1
1030 FOR X-1 TO E
1040 A=A*N
1050 NEXT X
1070 RETURN
```

By now you should feel that you are in complete control of your Pegasus. Try writing some programs of your own.

Good luck, and have fun!

A GENTLE INTRODUCTION TO PEGASUS BASIC

The BASIC Language

BASIC is the most commonly used computer language in the world today. The word BASIC is an acronym, standing for Beginners All-purpose Symbolic Instruction Code.

BASIC is a computer program that was originally developed at Dartmouth College in the U.S. as a means of teaching students the principles of computer fundamentals, as well as making it easier to write computer programs. BASIC itself is usually written in machine code assembler, although higher-level languages have been used.

Bells and Whistles

Hundreds of BASIC interpreters (i.e. programs that will accept and interpret a program written in BASIC) have been written since the first version, and each one is usually unique in its features and limitations. Theoretically anyone with enough knowledge and time can write a BASIC interpreter, although not many people so do. However, when they do, each likes to add a personal touch, in the form of special features, and this is known as adding Bells and Whilstles. (We have not stinted in this tradition). Thus, although BASIC is common, there are many dialects.

Where Do I Start?

At the beginning, of course! We'll look at the idea that a computer program is like a recipe. For example, let's make a milkshake:

```
Fetch container
Fetch milk
Pour milk into container
Fetch flavoured powder
Add powder to milk in container
Pick up container
Shake!
Oops!
Put down container
Clean up mess
Put lid on container
Shake!
Take lid off
Drink milkshake
End of recipe
```

A trivial, almost useless, example. Each line or statement is a command or instruction (apart from 'Oops!' which is a comment or perhaps an invective). The statements were executed sequentially, starting from the top. Note that each statement leaves out a very great amount of detail - like what sort of container was used, where the milk came from, what flavoured powder was used, even whether it was enjoyed or not!

Computer programs are quite like this in their detail - a great deal is implicit or assumed. Computer programs are much simpler, however, in the actions that they describe, in that the tasks a computer performs are (usually) logical and straightforward - unlike the real world of gravity and spilt milk.

Using Numbers

BASIC, like many other computer languages, is designed to work with numbers. Usual operations in BASIC are addition, subtraction, multiplication and division (+,-,*,/). There are two ways that numbers are used in BASIC - constants and variables.

A constant has a value which it keeps for as long as the program runs. Typical constants are 7, 24, 0, -32768, 2000. Variables are symbols for memory cells that may contain numbers. In Pegasus BASIC we use the letters A through to Z to represent these variables. Thus we can refer to a variable in a computer program without having to know its value. When a computer program is first RUN, all variables, A to Z, will have a value of zero.

Number Size

Pegasus BASIC is an integer BASIC, which means that all numbers in it have no fractional part. E.g. 3.7, 4.02 and 3.1415926 are not integers. Further, the Pegasus has 16 bit signed two's complement and 16 bit unsigned numbers, which means that for signed numbers you are limited to -32768 to 32767, while unsigned integers have a range of 0 to 65535. Any outside this range will cause an error.

Number Representation

Numbers are stored internally in binary, but to make it easier for people to handle them, we have provided two forms of integer format: numbers may be output (printed) in decimal or hexadecimal (base 16). For instance, if variable A contains 19, then we can print the two forms thus:


```
  PRINT A," ",HEX(A)
```

which will print out:

```
  19 13
```

For inputting numbers, they must always be in decimal, but may be signed or unsigned. Hexadecimal numbers may be used directly in a program by preceeding them with a dollar sign ($), e.g.:

```
  PRING $13
```

will print:

```
  19
```

on your television screen. Both signed and unsigned numbers may be used and may be selected with two statements, SIG and USIG, which stand for signed and unsigned. Signed numbers have a range of -32768 to +32767, while unsigned are in the range of 0 to 65535. Note that an unsigned number greater than 32767 will be printed as a negative number if the program switches back to the signed mode.

Arithmetic

In BASIC,arithmetic may be done with expressions. An expression is a group of tokens, each of which has a definite value associated with it, that is built up using a set of possible operators, and is solved as an algebraeic expression that returns a single numeric value. Now that we have confused you, let's clear it up with some examples:

`A*3+7*R`	
`(3+Q)-(21/L+(8*I))`	Note that the parentheses must match
`2+2`	
`1`	yes, a number is an expression too
`$4F OR 51`	note the Boolean operator
`ABS(-R)`	functions are expressions too

A variable or constant by itself may also be considered an expression, and expressions may consist of other expressions, as long as they are logically organised, and the number of left and right parentheses match correctly. Unlike some BASICs, nearly any complexity of expressions can be used.

Operators

Constants, functions, variables and expressions may be related by operators to form a new expression. All the operators work with 16 bit integers, and return 16 bit integers as results.

+	simple addition
-	subtraction
*	multiplication
/	divison
MOD	modulus, same as taking remainder after a division instead of the quotient. e.g. 7 MOD 6 yeilds 1.
+	unary plus, e.g. +7 by itself
-	unary minus, e.g. - 12
NOT	returns one's complement, e.g. NOT $F012 returns $0FED.
AND	logical AND, may also be used as Boolean connector
OR	logical OR, similar to AND

Note that expressions are no good unless you do something with them, using one of the statements available. The simplest statement to use is the assignment statement, LET. This is used for assigning values to variables, e.g.

```
LET Q=I+9          The '=' means 'is assigned'
L=17*(8+T) MOD 15  The LET is optional
I=I+1
```

The last statement is of particular interest since it illustrates how a variable is fetched, incremented, and then stored back in the same memory cell again. The '=' sign does not mean 'equals', but means is assigned the value of '. Note that 3=A or 3=7 are illegal and will give an error message. Spaces may be used freely in expressions, however, they may not be imbedded inside function or statement names.

A quick way of using your Pegasus for math is to use the PRINT statement in conjunction with an expression. Remember that the question mark, '?', is shorthand for PRINT. For instance, ? 7*8 gives 56. When expressions are evaluated, they are executed in an order defined by the operator precedence. This means that values that are conjoined by certain operators will be executed before others in an expression. The precedence order is as follows:

1st	constants, variables
2nd	functions (includes special @ function)
3rd	unary - or +, NOT
4th	operators * / MOD AND
5th	operators + - OR

The order of evaluation may be changed by using parentheses. Some examples are given for your enjoyment:


```
3+4 * 2+5 resolves to 16
(3+4) * (2+5)evaluates to 49
```

A special class of operator, the relational operator, is covered in the section on Booleans.

6th	relational operators (lowest precedence)

Booleans

A Boolean expression is similar to an arithmetic expression, apart from the use of the realtional operators. Any relation evaluates to 0 for false and non-zero for true. The most usual non-zero value found will be -1 (hex FFFF). The relational operators are:

=	equality
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
<>	not equal to

Boolean expressions may be mixed with arithmetic expressions, leading to results like:

```
A=B=C+1
```

Boolean expressions may be used with the IF statement, e.g.

```
IF Q=7 THEN END
IF T THEN GOTO L
```

Here, L is treated as an unsigned line number that the program will GOTO if T is non-zero.

Statements and the Editor

Program lines in BASIC are usally organised in a strictly sequential manner, using line numbers in the range of 1 to 65535. A program will consist of a series of lines, where each line consists of one or more statements (separated by the colon, ':') and is executed sequentially, except where a special statement will change the flow of program logic. Here is a sample program that will print out the integers between 1 and 10.

```
10  I=0 : REM I IS ASSIGNED A VALUE OF ZERO
20  I=I+1 : REM I IS INCREMENTED
30  PRINT I : REM PRINT OUT THE VALUE CONTAINED
    IN I
40  IF I=10 THEN STOP : REM STOP WHEN I REACHES
    10
50  GOTO 20
```

Follow the program through by hand, or better still, try it on your Pegasus. When typing the program in, terminate each line with the RETURN key, and correct any typing mistakes by using the BACK SPACE key. If you notice a mistake on a line that you have already left, simply re-type the correct version (with the same line number) and the old line will be automatically replaced. To remove a line entirely type the line number by itself, followed by the RETURN key.

Experiment with your own programs to print out different sorts of number sequences, until you are fully satisfied with the material covered so far. If you have trouble stopping a program once you have started it, tap the break key.

Summary of Statements

PRINT expressions, string constants

This statement will evaluate and print results of expressions, as well as printing string constants.

A string constant is a collection of characters delimited by double quotes, e.g.

"FRED NURKE WAS HERE"

"THAT'S all FOLKS"

Expressions will be evaluated, and the results printed, with no leading or trailing spaces. String constants and expressions MUST be separated by commas. Upon completion of the print statement, the cursor will move to the beginning of the next line, unless the PRINT statement is terminated with a comma. The cursor may be positioned to anywhere on the screen at any stage in the PRINT by using the form [x,y]. For example, PRINT [10,2], "HELLO", will move the cursor to column 10, line 2, and print "HELLO", leaving the cursor immediately after the 'O'. The vertical position 'y' is optional, but if it is included then it must be separated from the 'x' column position by a comma. There are three functions that may only be used with the PRINT statement, since all of them produce some sort of output, without returning a value. These output functions are detailed below:

CHR (expression)

This will output the ASCII character that is represented by the result of the expression. The result is forced into the range of 0 to 255 (decimal), or $00 to $FF (hex). If the number is greater than 127, then the character will be inverted. Note that characters in the range of 0 to 31 and 128 to 159 will not print, but will cause one of the control functions to be executed.

HEX (expression)

The expression is evaluated, range 0 to 255, and the appropriate hex number is output, range $00 to $FF.

RAW (expression)

This function is very similar to CHR, except that values in the range of 0 to 31 and 128 to 159 will have a special character output, without executing the appropriate control function.

Note that all functions that require an expression in brackets, must not have a space before the left parentheses. (The RAW function is associated with the RAWON and RAWOFF statements, covered later in this document.) Examples of their use are given below, for you to try on your Pegasus.

```
PRINT "Print a hex number:",HEX(19), CHR(10),
      RAW(0)
PRINT CHR($46),CHR($52),RAW($45),CHR($44)
PRINT "There are ",Q," beans in the box."
PRINT CHR(12)  :  REM Clear screen
PRINT RAW(12)  :  REM Output Greek letter 'nu'
```


LIST starting line, ending line

Program lines may be listed out, either as individual lines, subranges of lines, or the entire program. The expressions are both optional, and are unsigned numbers always. If a line is specified that is not in the program, then the nearest one to it will be used. This is the only case in which such leniency is tolerated. If you try to force BASIC to use the 'nearest' line number in other statements, then a small quantity of plastic explosives attached to your Pegasus will be detonated, removing your typing fingers.
YOU HAVE BEEN WARNED.
Note that the starting and ending lines may be expressions, and the LIST statement may be part of a BASIC program.

RUN expression

The RUN statement will initialize all variables, then start program execution at the line number specified. If no number is specified, the program will start at the beginning.

INPUT "string constant" input list

This statement, unlike many others, can only be executed with a line number as part of a program. Its purpose is to request numbers from the user for input to the program. The string constant (if specified) will be printed out as a prompt to the user before input is requested, and must not be followed by a comma. When each input expression (yes, expressions can be input) is typed, it must be terminated with a RETURN key. Only one string constant may be specified, and if used it must be immediately after the INPUT.

FOR variable = start value TO end value STEP step-size. This is the standard BASIC looping statement. This will cause all statements between the FOR and its appropriate NEXT to be executed repeatedly until the variable's value reaches or exceeds the end value. Note that the step size may be positive or negative. If the step is not given, it will default to one.

NEXT variable name
Terminating statement for FOR loops.

GOTO expression
The expression will be evaluated to an unsigned 16 bit integer, and if a line is found with a matching line number, then that line will be executed next.

GOSUB expression
The expression will be evaluated, and the subroutine which starts with the matching line number will be called, returning to after the GOSUB statement when it reaches and executes the RETURN statement. GOSUBs may be nested to any depth, depending upon free ram space for the stack.

RETURN

This statement indicates the logical end of a BASIC subroutine.

EXIT

The EXIT statement will return you back to the Pegasus Menu selection mode.

NEW

This statement will zero all variables, as well as deleting all program lines.

STOP

The STOP statement will cause program execution to terminate, returning to the line edit mode. Execution may be continued with the CONT statement, as long as the program has not been changed. Any other immediate mode statement may be executed however.

END

The END is similar to the STOP statement, except that the CONT statement will not continue program execution after an END

CONT

The CONT will cause program execution to continue, as defined by the STOP and END statements.

REM

Any user remarks may appear after this statement, since they will be ignored by the BASIC interpreter. The REMark is terminated by the end of line or a colon.

LET variable name = expression
The assignment operation assigns the value of an expression to the named variable. Only variables and the special function '@' may be used on the left side of the '=' sign.

IF expression THEN statement or expression
The IF statement will evaluate the first expression, and if it is zero, then the remainder of the statement will be skipped, going to the next line. Upon a true state, then the part after the THEN will be executed if it is a statement, or if it is an expression, then it will be evaluated, and a GOTO will be executed.

TRON

This statement will bring the trace mode into effect, whereby each line number will be printed out as the line is executed, following the flow of program execution as the RUN proceeds.

TROFF

This statement will turn the trace mode off.

SIG

This forces the system to accept and print only signed numbers, in the range of -32768 to +32767. A point to note here is this example:
PRINT HEX($B010/256) will yield B1, instead of the expected value of B0. This is because although the hex number is unsigned, SIGned mode is in effect, and must be disabled using USIG before the correct result may be achieved.

USIG

The system can accept unsigned integers, in the range of 0 to 65535, for input, output, and arithmetic expressions. Note that this mode is checked when determining whether the result of an expression is outside its range.

SAVE

BASIC programs are saved on cassette tape, with a filename that you may specify (8 characters only). The BLUE tagged lead goes into the MIC jack, while the YELLOW lead goes into the EAR jack.

LOAD

Previously SAVEd programs may be loaded from cassette tape. The filename and load area will be printed.

POKE expression , expression
The first value resolves as an unsigned 16 bit address, which gives the location to poke the second value into.

LINES expression
This statement controls the number of lines displayed on the screen. The expression must resolve to a number in the range of 1 to 16, or an error will stop execution of the program. Reducing the number of lines displayed has the result of speeding up program execution proportionally.

RAWON

This statement will turn on the RAWMODE flag. This means that any control code that is echoed to the screen through the normal PRINT routine, or through typing in lines, will cause a special character from the character generator ROM to be printed, without executing the control function.

RAWOFF

Turns RAWMODE off.

CLS

This statement, when executed, will clear the video display screen, and move the cursor to the top left hand corner.

BASIC Functions

Pegasus BASIC has a number of functions, each of which may be used in expressions, (apart from the ones specified in the PRINT statement description). Note that there must be no spaces between the function name and the left parenthesis.

ABS(expression)
: Takes absolute value of the argument.

PEEK(expression)
: Returns byte at address given by expression.

FREE
: Returns number of free bytes available in system RAM.

RND
: Returns pseudo-random number in the range 0 to 255.

USR(expression)
: Calls a machine language routine subroutine in memory at the address specified by the expression - the function value returned reflects the state of the X register, and may be data or an address pointing at more data.

@(expression)
: Special function that implements a one dimensional integer array that utilises all available RAM space. The function may be used anywhere that a variable name is used, including in assignment statements. Unlike variables, the @ array is not cleared by the NEW statement. The array index is

unsigned, starts at zero, and its size is FREE/2-1

INKEY

This function will scan the keyboard to see if a key has been pressed - if one has, then its ASCII value in the range of 1 to 127 will be returned, else if no key has been pressed then zero will be returned.

Information for Experts

The variables, since they are in fixed locations in RAM,(in the 4K system only), can be accessed by machine code subroutines by referencing directly their addresses. There are 26 variables, and they start at $B03C.

Nearly all tokens in BASIC have a shorthand form, for instances, '?' means PRINT, and 'e' means PEEK(. Try finding out what the rest are - this information will be published in the newsletter.

When a program is executing on the Pegasus, it may be stopped in its tracks by using the BREAK key on the keyboard. This is functionally equivalent to the program encountering a STOP statement.

Output that is being sent to the screen may be paused by use of the ESCAPE key, on the upper left of the keyboard. Tapping once will stop, tapping again will start.

If inverse video characters are required inside strings, they may be effected by tapping the blank key on the extreme lower right of the keyboard. Tapping the key again will remove the inverted state. Note that only characters inside double quotes will remain inverted. Inverted characters may also be generated by setting the most significant bit of the byte for the ASCII character (ASCII equivalent greater than 128).

RAWMODE may be set and resent by tapping the blank key on lower right of keyboard, second one in. When in effect, any control characters typed will appear as a special printing character, without the appropriate control function being executed.

Note that the RETURN key, being a control code, will not work as it should until the RAWMODE flag is turned off by tapping the second blank key again. This feature allows you to insert control codes into strings, and then the output of those control codes as characters or functions may be governed by use of the BASIC statements RAWON and RAWOFF.

BASIC may be re-entered from the monitor after using the PANIC button by jumping to $0B offset from its start.

Basic Error Numbers and Messages

Whenever a syntax or eduction error occurs, then an error number will be printed out, each number matching to one of the error messages given below:

(1) Out of Memory
This means that there is insufficient RAM space between the program end and the stack to perform the last operation.

(2) Invalid Line Number
A line number was specified that either does not exist, or is illegal

(3) Next Without For
A NEXT statement was found without the appropriate FOR statement

(4) Syntax Error
This is a general error that occurs whenever there is incorrect syntax in a program line

(5) Return Without Gosub
A RETURN statement was executed, but the system did not find a GOSUB to return to

(6) Immediate Mode Illegal
A statement was executed in immediate mode that is illegal for that mode

(7) Overflow Error
The results of an arithmetic operation exceed the current range specified

(8) Divide by Zero
An attempt was made to divide a number by zero.

(9) Screen length Error
The LINES statement must have an argument in the range of 1 to 16 only.

BASIC STRING INPUT/OUTPUT

Subroutine 1000 stores a string of 32 characters beginning at $B8ØØ + 32 * N, where N is an integer from 0 to 31 if 1K of RAM is used from $B8ØØ to $BBFF.

Line 1020 — I is the memory location for string element of string number N

Line 1030 — Inkey checks for a key pressed

Line 1040 — backspaces and deletes previous character from RAM

Line 1050 — returns from subroutine

Line 1060 — prints the character and stores it in location S + I

Line 1070 — puts a null at the end of the string

Subroutine 2000 outputs the string for a given N

Line 2010 — -PEEK's the element of the string

2020 — -checks for the end null

2030 — -prints the string element and increment S for the next one.

To input or output a string, specify the string number and then use the appropriate subroutine. The input routine will wait for Keyboard input, ending with a RETURN.

```
10 REM SAMPLE STRING I/O
20 REM FOR BASIC
30
50 REM INPUT STRING
100 N=1:GOSUB1000:END
150 REM OUTPUT STRING
200 N=1:GOSUB2000:END
500
1000 REM STRING INPUT
1020 I=$B800+32*N:S=1
1030 K=INKEY:IF K=0 THEN 1030
1040 IF K=8 THEN PRINTCHR(8),:S=S-1:POKE(S+I),0:GOTO1030
1050 IF K=13 THEN RETURN
1060 PRINTCHR(K),:POKE(S+I),K:IF S<33 THEN S=S+1:GOTO1030
1070 POKE(S+I),0:RETURN
1100
2000 REM STRING OUTPUT
2010 I=$B800+32*N:FOR S=1 TO 32:C=PEEK(S+I)
2020 IF C=0 THEN S=32:NEXT S:RETURN
2030 PRINTCHR(C),:NEXT S:RETURN
9999 END
```

The following pages begin an introduction to the language FORTH . This section of the manual should familiarise you with stack operations, simple integer calculations and elementary programming development. For more comprehensive programming, the magazines or books in the reference list should be consulted.

Using FORTH:
1) Switch your computer on
2) You should see a display including the words FORTH 1.1
3) Press F
4) The message Pegasus 6809 FORTH should appear
5) Now you are ready to begin reading the FORTH manual.

To EXIT FORTH
1) Type MON (press RETURN)
2) You are now back to the Select mode

TO REENTER FORTH
1) You should be in the Select mode
2) Type M
   G 0011. (the . should return you to FORTH)
3) ONLY use this method if you have previously been in FORTH since switch on

PANIC!
1) If for some reason (usually an illegal loop operation) the computer 'hangs-up' and will not respond, all is not lost. Press the NMI button inside the computer and then reenter FORTH as above.

INDEX

INTRODUCTION	E - 2.0
STACK OPERATIONS (Cassette program)	E - 7.50
ARITHMETIC OPERATIONS	E - 8.0
PROGRAMMING ADVICE	E -10.0
SAVE & LOAD	E -10.1
FORTH WORD REFERENCE	E -11.0
SCREEN OPERATIONS	E -14.0
ASCII CHARACTERS	E -15.0
DICTIONARY WORDS	E -22.5
FORTH EDITOR	E -34.0

PEGASUS FORTH

FORTH is quite different in structure from languages such as Basic or Pascal. FORTH begins with a set of defined WORDS in a DICTIONARY. Creating a program involves using these WORDS to define new WORDS which may be linked together to perform some task. For example, to display a chart of numbers on the screen, WORDS would be defined to print headings, to calculate results, and to print these results in columns. The word CHART would use all these words in its definition to perform the total task.

The FORTH HANDY REFERENCE and the FORTH DICTIONARY WORDS sections list and explain the uses of the FORTH words available when the computer is put into the FORTH language.

FORTH REFERENCES

BYTE	August 1980
Dr. DOBB'S JOURNAL	January 1981
USING FORTH	FORTH, Inc. 2309 Pacific Coast Highway, Hermosa Beach, California, 90254
FORTH DIMENSIONS	Forth Interest Group P.O. Box 1105 San Carlos, California, 94070

A Gentle Introduction to Pegasus Forth

Forth as a language is very different from most other computer languages, such as Basic or Pascal. It requires a structured approach, (having no GOTO statement), yet has all the convenience of an interactive interpreter. An expert's definition of Forth might be:

threaded, extensible, interactive, tree-structured, self-implementing, interpretive language.

Stack Concepts

Forth is a stack language. A stack can be defined as a collection of data items, usually byte (8 bit) or work (16 bit) quantities, which are pointed to by a register known as a stack pointer, and are arranged so that the last item placed on to the stack is the first to be removed. Stacks usually grow from high memory addresses down to low memory addresses, and are contiguous.

Any form of data may be placed on a stack, including number, characters, or addresses which point to data. If you have used a calculator with Reverse Polish Notation (RPN, e.g. Hewlett Packard), you will be at home with Forth. Such a calculator operates something like Forth in its use of a stack, and post-fix notation (or RPN), to evaluate expressions.

For instance, to evaluate (3*4)+(6*2), where the asterisk '*' is the standard computer symbol for multiplication, we would enter into the computer:

```
3 4 * 6 2 * + .          (press RETURN)
```

with a space separating each symbol, or word.

Evaluating from left to right: 3 is stacked, then 4, then the multiplication operator multiplies the two numbers, removing them from the stack, and leaving 12 on the top. Then 6 and 2 are stacked and multiplied, leaving another 12 on top of the 12 already there. The two products are then added, leaving 24 on the stack, which is then removed from the stack and printed with the '.' operator.

After the RETURN key is typed, the computer will respond

```
24 OK
```

It is a convention with Forth that when the top item of the stack is operated upon, then it is destroyed.

Forth has two stacks - the variable S stack, which is used for passing data to operations, and the return stack, R, which is reserved for passing return addresses, as well as maintaining parameters for loop variables and controlling program flow. The S and R stacks occupy different regions of memory, although data may be passed between them.

When typing in numbers or Forth words to the Forth input interpreter, separated by spaces, each data item is placed onto the S stack, starting from left to right.

The S and R stacks are primarily used for internal communications within the Forth system. Most Forth operations communicate only through a stack.

Dictionary

The aspect of Forth which gives it its great power and extensibility is the Dictionary, which contains the definitions of the vocabulary items. Nearly all of Forth consists of Dictionary entries. Each entry is either machine code, or a variable length list of addresses pointing to other Dictionary entries. The Dictionary is extensible, growing upwards towards high memory. In the Pegasus, most of the Dictionary resides in EPROM, with any extensions occuring in RAM.

The user may extend the Dictionary by defining new entries at the terminal. New entries may consist of standard words, or previously defined user words. New entries may also have the same name as previous entries, in which case they are considered to redefine the previous word. For instance, the addition operator, +, may be redefined as multiplication, *, leading to results like

```
3 4 + . 12 OK
```

Programming

Now we come to writing the program itself. Firstly, the problem is broken down into all its constituent sub-tasks through a process called 'step-wise refinement'. Each subtask is then defined as a Forth word, in terms of words previously defined. Then build modules upwards by defining more words, until finally the last word that you define is the name of the program itself.

As an example, let's write a small program to evaluate the polynomial

$$3x^2 + 4x - 7$$

for x=2 and x=7
We type in the following:

```
: POLY DUP DUP *  3  * SWAP  4 * + 7 - . ;
```

The colon word ':' means that a definition is to follow - the definition is terminated by the semicolon word ';'.
POLY is the name of the Forth word that we are defining - any name of up to 31 characters is acceptable here, and it can include any character except the space or RETURN, including control codes or special punctuation characters.

DUP is a Forth word that duplicates the top stack, and leaves the result on the stack. Executed twice, this makes two copies of the number on the top of the S stack.

The * operator is used to multiply these two copies, destroying them and leaving the result on top. The result is then multiplied by 3, leaving a new result, then SWAPped with the original number, x, which is multiplied by 4. Then the two terms are added, 7 is subtracted, and the result printed.

The program is now written, so let's run it:

```
2 POLY (CR) 13 OK
7 POLY (CR) 168 OK
```

It's as simple as that!

POLY is now a Forth word, that will be there until we turn off the Pegasus, or until we tell it to

```
FORGET POLY (CR)
```

in which case all definitions subsequent to and including POLY will be destroyed (unless they are saved on cassette first).

Forth Program to generate first n Fibonacci Numbers

```
: FIBO 1 + 0 SWAP 1 SWAP 1
DO CR DUP . SWAP OVER +
LOOP ;
```

To run program, simply enter number of terms to generate, and invoke, e.g. for 10 terms, enter

```
10 FIBO        (press RETURN)
```

FORTH

Please note that the following pages; E - 7.1 to E - 7.5 inclusive will only apply when used with the cassette "Forth Stack Operations" which can be obtained from your dealer.

STACK OPERATIONS

(Please see next page for loading cassette instructions and program commands).

This section describes, with the aid of a FORTH cassette program, some of the operations available in handling numbers on the FORTH variable S Stack. The S Stack is used during calculations; the return stack R (used for loops and program flow) will not be discussed in this section.

VARIABLE STACK

This Stack is a storage place for numbers used during a FORTH program. When a number is entered, it is placed on the top of the stack. When another number is entered, the new number is placed on the top after the previous numbers are pushed down.

CASSETTE PROGRAM: STACK OPERATIONS

Only the top 5 locations of the stack will be shown. The total number of locations available depends on the computer memory size.
Load the FORTH program "STACK OPERATIONS"
Type in BEGIN STACK (then press RETURN)
When OK and the flashing cursor appear, the program is ready to enter numbers.

Type 5 (press RETURN)
Observe how 5 is placed on the stack
(press CTRL and U at the same time to clear the cursor's line)
Enter these numbers in the same way:

6 10 2

The stack should now show

2
10
6
5
0

You are now ready to experiment with the words in the Operation List. Only the upper case letters should be typed (e.g. DUP, ROT etc).

Loading Cassette Program:

1) Power up PEGASUS
2) Type F to select FORTH 1.1
3) Type LOAD ,press PLAY on your recorder
   press RETURN on the keyboard
4) When the program has loaded successfully, the display should show FSTACK LOADED . Turn off your recorder.

Starting the Program:

Type BEGIN STACK then press RETURN

To Clear the Stack Display:

Type CLEAR (then press RETURN) Zeros will be displayed on the stack.

To Redisplay the Stack

If some error has caused the display to scroll up, then type WST (press RETURN) to display the stack again, with the latest values on the stack.

To EXIT the Program:

1) If you wish to permanently leave the program, press the computer's NMI button. (located inside the computer box)
2) If you wish to use FORTH type CLS FORTH (then press RETURN)

   To begin the stack program again, type BEGIN STACK (press RETURN)

NOTE: THE NEGATIVE SIGN - IS AN UNSHIFTED KEY; the shifted key is NOT a negative sign or minus but an underline.

The following is only a suggested exercise before experimenting yourself with the stack.

Type in one line at a time and observe each operation (Remember to delete the cursor's line, use CTRL U)

DROP	2 removed
DUP	10 copies
DROP	top 10 removed
SWAP	10 and 6 change places
OVER	10 copied and put on top
ROT	Third number put on top
+	Top two numbers added
SWAP	
-	Top number subtracted from second
*	Top two numbers multiplied
2	2 entered
/	Second number divided by top

Enter numbers and use operations in any order you wish until you understand fully each operation.

NOTE:
Numbers for -20 to 20 are entered automatically (For numbers outside this range, please type E after each number e.g. 105 E 61 E ONLY during this cassette program)
You may use 2 or more operations on the same line (e.g. 1 2 3 + DUP)

Type F to speed up the display, S to slow down the display Multiple F's or S's may be used, each separated by a space.

You are now ready to attempt these exercises.
(Answers on next page)

1) Enter 1 2 3 4 5 onto the stack

Which operation would you use to get

(a)	4	then (b)	3	then (c)	3	then (d)	6
	5		4		3		4
	3		5		4		5
	2		2		5		2
	1		1		2		1

2) Enter 1 2 3 4 5 again

Which two operations would you use to get

(a)	4	then (b)	4	then (c)	3
	5		5		9
	5		3		3
	3		2		2
	2		1		1

3) Use the stack to do these calculations (enter the numbers in the correct order and then do the operation)

(a) 6 + 2
(b) 6 - 2
(c) 6 * 2
(d) 6 - 2 (use / for division)

4) Use the stack to find (do multiplication before addition)

(a) 5 * 2 + 3
(b) 5 + 2 * 3

ANSWERS

1. (a) SWAP (b) ROT (c) DUP (d) +

2. (a) DUP ROT (b) ROT DROP (c) + OVER

3. (a) 6 2 + (b) 6 2 - (c) 6 2 * (d) 6 2 /

4. (a) 3 5 2 * + (b) 5 2 3 * +

Arithmetic Operations

In normal operation, this version of FORTH uses 16-bit signed arithmetic which allows integers from -32768 to +32767 to be used. You can develop methods to work with numbers outside this range, but only this range will be assumed for now.

CALCULATIONS

Reverse Polish Notation (RPN) is used in calculations. For Example:

```
6 + 5 * 2 is entered as   5 2 * 6 +
(Note:  * is the symbol for multiplication)
(       * is done before +  )
```

To print the answer, (which is stored on top of the stack) the FORTH word . is used (i.e. a full stop)

Type in 5 2 * 6 + .
The answer 16 should be printed in the same line followed by OK.

Exercises: (The answers are on the following page)

Change these operations to RPN; then use FORTH to print the answers.

(a) 7 * 2 + 3 + 4
(b) 7 * 2 + 3 * 4
(c) 7 * (2 + 3) * 4
(d) 72 - 9 (use / on the keyboard for division)
(e) (64 + 6) - 7

ANSWERS

(a) 7 2 * 3 + 4 + .

(b) 7 2 * 3 4 * + .

(c) 2 3 + 7 * 4 * . (+ adds the 2 and 3)
(first * multiplies the answer by 7)
(second * then multiplies by 4)

(d) 72 9 / .

(e) 64 6 + 7 / .

Note: Other answers are possible. For example (a) could be done
3 4 7 2 * + + .
In this case, all numbers were put on the stack, and then the operations were performed.

Similarly: (c) 7 4 2 3 + * * .
(e) 7 64 6 + / .

PROGRAMMING ADVICE

* A list of FORTH WORDS is given in the FORTH HANDY REFERENCE pages.

* A new WORD may be defined using previously defined words by using : to begin a definition (leave a space after the colon)
  ; to end the definition

  EXAMPLE: : CK 16 EMIT ; to display the clock
  CK is the new word to be defined.
  To use the word, type CK

* A WORD can be any string of up to 31 characters bounded by spaces.
* Keep WORDS simple in operation. Test each new word after you define it. Then build further words from these simpler words until the full TASK is developed.
* All FORTH WORDS must be bounded by spaces.
  For numbers 5 2 6 enters three numbers
  5 26 enters two numbers (5 and 26)

SOME FORTH MESSAGE STATEMENTS

WORD NOT FOUND	You have not defined a word used or you have misspelt the word
REDEF: ISN'T UNIQUE	Just a warning that a word has been defined before. You may continue with the new definition.
CONDITIONALS NOT PAIRED	You forgot one of the words that are used to end DO , IF, or BEGIN structures.

CASSETTE SAVE & LOAD

To SAVE the FORTH words you have defined -

1. Check that the recorder is properly attached to your PEGASUS
2. Type SAVE (press RETURN)
3. The screen will display FILENAME?
4. Type in the name of your program (8 characters maximum) DO NOT PRESS RETURN YET
5. Press RECORD & PLAY on you recorder and then press RETURN on the keyboard.
6. The screen display will be blank during the SAVE operation.
7. When the display returns, turn off the recorder. Your FORTH words are now saved.

To LOAD a FORTH cassette -

1. Check that the recorder is properly attached to your PEGASUS and that the volume control is at a suitable level. (you may have to make some initial adjustments for a proper LOAD, but once this level has been determined, keep the level at that point)
2. Type LOAD DO NOT PRESS RETURN
3. Depress PLAY on your recorder and then press RETURN on the keyboard.
4. The screen will be blank during the load operation.
5. When the display returns, your program should be loaded. Turn off your recorder.
6. If the program did not load, then begin at 1 above. The most probable cause for a faulty load is an incorrect volume level.

FORTH HANDY REFERENCE

Operand key:

n, n1	16-bit signed numbers
d, d1	32-bit signed numbers
u	16-bit unsigned number
addr	address
b	8-bit byte
c	7-bit ascii character value
f	boolean flag

STACK MANIPULATION

DUP	(n - n n)	Duplicate top of stack
DROP	(n -)	Throw away top of stack
SWAP	(n1 n2 - n2 n1)	Reverse top two stack items
OVER	(n1 n2 - n1 n2 n1)	Make copy of second item on top
ROT	(n1 n2 n3 - n2 n3 n1)	Rotate third item on top
-DUP	(n - n ?)	Duplicate only if non-zero
>R	(n -)	Move top item to "return stack"for temporary storage (use caution)
R>	(- n)	Retrieve item from return stack
R	(- n)	Copy top of return stack onto stack

NUMBER BASES

DECIMAL	(-)	Set decimal base
HEX	(-)	Set hexadecimal base
BASE	(- addr)	System variable containing number base

ARITHMETIC AND LOGICAL

+	(n1 n2 - sum)	Add
D+	(d1 d2 - sum)	Add double-precision numbers
-	(n1 n2 - diff)	Subtract (n1-n2)
*	(n1 n2 - prod)	Multiply
/	(n1 n2 - quot)	Divide (n1/n2)
MOD	(n1 n2 - rem)	Modulo (i.e. remainder from division)
/MOD	(n1 n2 - rem quot)	Divide, giving remainder and quotient
*/MOD	(n1 n2 n3 - rem quot)	Multiply, then divide (n1*n2/n3), with double-precision intermediate

FORTH HANDY REFERENCE

Stack inputs and outputs are shown; top of stack on right.
This card follows usage of the Forth Interest Group (S.F. Bay Area); usage aligned with the *Forth 78* International Standard.

For more info: Forth Interest Group
P.O. Box 1105
San Carlos, CA 94070.

Operand key:

n, n1, ...	16-bit signed numbers
d, d1, ...	32-bit signed numbers
u	16-bit unsigned number
addr	address
b	8-bit byte
c	7-bit ascii character value
f	boolean flag

STACK MANIPULATION

DUP	(n — n n)	Duplicate top of stack.
DROP	(n —)	Throw away top of stack.
SWAP	(n1 n2 — n2 n1)	Reverse top two stack items.
OVER	(n1 n2 — n1 n2 n1)	Make copy of second item on top.
ROT	(n1 n2 n3 — n2 n3 n1)	Rotate third item to top.
?DUP	(n — n ?)	Duplicate only if non-zero.
>R	(n —)	Move top item to "return stack" for temporary storage (use caution).
R>	(— n)	Retrieve item from return stack.
R	(— n)	Copy top of return stack onto stack.

NUMBER BASES

DECIMAL	(—)	Set decimal base.
HEX	(—)	Set hexadecimal base.
BASE	(— addr)	System variable containing number base.

ARITHMETIC AND LOGICAL

+	(n1 n2 — sum)	Add.
D+	(d1 d2 — sum)	Add double-precision numbers.
-	(n1 n2 — diff)	Subtract (n1–n2).
*	(n1 n2 — prod)	Multiply.
/	(n1 n2 — quot)	Divide (n1/n2).
MOD	(n1 n2 — rem)	Modulo (i.e. remainder from division).
/MOD	(n1 n2 — rem quot)	Divide, giving remainder and quotient.
*/MOD	(n1 n2 n3 — rem quot)	Multiply, then divide (n1*n2/n3), with double-precision intermediate.
*/	(n1 n2 n3 — quot)	Like */MOD, but give quotient only.
MAX	(n1 n2 — max)	Maximum.
MIN	(n1 n2 — min)	Minimum.
ABS	(n — absolute)	Absolute value.
DABS	(d — absolute)	Absolute value of double-precision number.
MINUS	(n — -n)	Change sign.
DMINUS	(d — -d)	Change sign of double-precision number.
AND	(n1 n2 — and)	Logical AND (bitwise).
OR	(n1 n2 — or)	Logical OR (bitwise).
XOR	(n1 n2 — xor)	Logical exclusive OR (bitwise).

COMPARISON

<	(n1 n2 — f)	True if n1 less than n2.
>	(n1 n2 — f)	True if n1 greater than n2.
=	(n1 n2 — f)	True if top two numbers are equal.
0<	(n — f)	True if top number negative.
0=	(n — f)	True if top number zero (i.e., reverses truth value).

MEMORY

@	(addr — n)	Replace word address by contents.
!	(n addr —)	Store second word at address on top.
C@	(addr — b)	Fetch one byte only.
C!	(b addr —)	Store one byte only.
?	(addr —)	Print contents of address.
+!	(n addr —)	Add second number on stack to contents of address on top.
CMOVE	(from to u —)	Move u bytes in memory.
FILL	(addr u b —)	Fill u bytes in memory with b, beginning at address.
ERASE	(addr u —)	Fill u bytes in memory with zeroes, beginning at address.
BLANKS	(addr u —)	Fill u bytes in memory with blanks, beginning at address.

CONTROL STRUCTURES

DO ... LOOP	do: (end+1 start —)	Set up loop, given index range.
I	(— index)	Place current index value on stack.
LEAVE	(—)	Terminate loop at next LOOP or +LOOP.
DO ... +LOOP	do: (end+1 start —) +loop: (n —)	Like DO ... LOOP, but adds stack value (instead of always "1") to index.
IF ... (true) ... ENDIF	if: (f —)	If top of stack true (non-zero), execute. [Note: *Forth 78* uses IF ... THEN.]
IF ... (true) ... ELSE ... (false) ... ENDIF	if: (f —)	Same, but if false, execute ELSE clause. [Note: *Forth 78* uses IF ... ELSE ... THEN.]
BEGIN ... UNTIL	until: (f —)	Loop back to BEGIN until true at UNTIL. [Note: *Forth 78* uses BEGIN ... END.]
BEGIN ... WHILE ... REPEAT	while: (f —)	Loop while true at WHILE; REPEAT loops unconditionally to BEGIN [Note: *Forth 78* uses BEGIN ... IF ... AGAIN.]

./	(n1 n2 n3 - quot)	Like ./MOD but give quotient only
MAX	(n1 n2 - max)	Maximum
MIN	(n1 n2 - min)	Minimum
ABS	(n - absolute)	Absolute value
DABS	(d - absolute)	Absolute value of double-precision number
MINUS	(n - -n)	Change sign
DMINUS	(d - -d)	Change sign of double-precision number
AND	(n1 n2 - and)	Logical AND (bitwise)
OR	(n1 n2 - or)	Logical OR (bitwise)
XOR	(n1 n2 - xor)	Logical exclusive OR (bitwise)

COMPARISON

<	(n1 n2 - f)	True if n1 less than n2
>	(n1 n2 - f)	True if n1 greater than n2
=	(n1 n2 - f)	True if top two numbers are equal
0<	(n - f)	True if top number negative
0=	(n - f)	True if top number zero (i.e., reverses truth value)

MEMORY

@	(addr - n)	Replace word address by contents
!	(n addr -)	Store second word at address on top
C@	(addr - b)	Fetch one byte only
C!	(b addr -)	Store one byte only
?	(addr -)	Print contents of address
+!	(n addr -)	Add second number on stack to contents of address on top
CMOVE	(from to U -)	Move u bytes in memory
FILL	(addr u b -)	Fill u bytes in memory with b beginning at address
ERASE	(addr u -)	Fill u bytes in memory with zeroes, beginning at address
BLANKS	(addr u -)	Fill u bytes in memory with blanks, beginning at address

CONTROL STRUCTURES

DO...LOOP	do: (end+1 start -)	Set up loop, given index range
!	(- index)	Place current index value on stack
LEAVE	(-)	Terminate loop at next LOOP or +LOOP
DO...+LOOP	do: (end+1 start -) +loop: (n -)	LikeDO...LOOP, but adds stack value (instead of always '1') to index
IF...(true) ...ENDIF		If top of stack true (non-zero) execute (Note: Forth 78 uses IF...THEN)
IF...(true) ...ELSE ...(false) ...ENDIF		Same but if false, execute ELSE clause. (Note: Forth 78 uses IF...ELSE...THEN)
BEGIN...UNTIL		Loop back to BEGIN until true at UNTIL (Note: Forth 78 uses BEGIN... END)
BEGIN...WHILE ...REPEAT		Loop while true at WHILE, REPEAT loops unconditionally to BEGIN (Note: Forth 78 uses BEGIN...IF ...AGAIN)

TERMINAL INPUT-OUTPUT

.	(n -)	Print number
.R	(n fieldwidth -)	Print number, right justified in field
D.	(d -)	Print double-precision number
D.R	(d fieldwidth -)	Print double-precision number, right-justified in field
CR	(-)	Do a carriage return
SPACE	(-)	Type one space
SPACES	(n -)	Type n spaces
."	(-)	Print message (terminated by ")
DUMP	(addr u -)	Dump u words starting at address
TYPE	(addr u -)	Type string of u characters starting at address
COUNT	(addr - addr+1 u)	Change length-byte string to TYPE form
KEY	(- c)	Read key, put ascii value on stack
EMIT	(c -)	Type ascii value from stack
EXPECT	(addr n -)	Read n characters (or until carriage return) from input to address

INPUT-OUTPUT FORMATTING

NUMBER	(addr - d)	Convert string at address to double precision number
<*	(-)	Start output string
*	(d - d)	Convert next digit of double precision number and add character to output string
*S	(d - 0 0)	Convert all significant digits of double prevision number to output string
SIGN	(n d - d)	Insert sign of n into output string
*	(d - addr u)	Terminate output string (ready for TYPE)
HOLD	(c -)	Insert ascii character into output string

DEFINING WORDS

: xxx	(-)	Begin colon definition of xxx
;	(-)	End colon definition
VARIABLE xxx	(n -) xxx: (- addr)	Create a variable named xxx with initial value n; returns address when executed
CONSTANT xxx	(n -) xxx:(- n)	Create a constant named xxx with value n; returns value when executed
BUILDS... DOES	does: (- addr)	Used to create a new defining word, with execution-time routine for this data type in higher level Forth

VOCABULARIES

CONTEXT	(- addr)	Returns address of pointer to context vocabulary (searched 1st)
CURRENT	(- addr)	Returns address of pointer to current vocabulary (where new definitions are put)
FORTH	(-)	Main Forth vocabulary (execution of FORTH sets CONTEXT vocabulary)
DEFINITIONS	(-)	Sets CURRENT vocabulary to CONTEXT
VOCABULARY xxx	(-)	Create new vocabulary named xxx
VLIST	(-)	Print names of all words in CONTEXT vocabulary

TERMINAL INPUT-OUTPUT

.	(n –)	Print number.
.R	(n fieldwidth –)	Print number, right-justified in field.
D.	(d –)	Print double-precision number.
D.R	(d fieldwidth –)	Print double-precision number, right-justified in field
CR	(–)	Do a carriage return.
SPACE	(–)	Type one space.
SPACES	(n –)	Type n spaces.
."	(–)	Print message (terminated by ").
DUMP	(addr u –)	Dump u words starting at address.
TYPE	(addr u –)	Type string of u characters starting at address.
COUNT	(addr – addr+1 u)	Change length-byte string to TYPE form.
KEY	(– c)	Read key, put ascii value on stack.
EMIT	(c –)	Type ascii value from stack.
EXPECT	(addr n –)	Read n characters (or until carriage return) from input to address.

INPUT-OUTPUT FORMATTING

NUMBER	(addr – d)	Convert string at address to double-precision number.
<#	(–)	Start output string.
#	(d – d)	Convert next digit of double-precision number and add character to output string.
#S	(d – 0 0)	Convert all significant digits of double-precision number to output string.
SIGN	(n d – d)	Insert sign of n into output string.
#>	(d – addr u)	Terminate output string (ready for TYPE).
HOLD	(c –)	Insert ascii character into output string.

DISK HANDLING (To be available later)

LIST	(screen –)	List a disk screen.
LOAD	(screen –)	Load disk screen (compile or execute).
BLOCK	(block – addr)	Read disk block to memory address.
B/BUF	(– n)	System constant giving disk block size in bytes.
BLK	(– addr)	System variable containing current block number.
SCR	(– addr)	System variable containing current screen number.
UPDATE	(–)	Mark last buffer accessed as updated.
FLUSH	(–)	Write all updated buffers to disk.
EMPTY-BUFFERS	(–)	Erase all buffers.

DEFINING WORDS

: xxx	(–)	Begin colon definition of xxx.
;	(–)	End colon definition.
VARIABLE xxx	(n –) xxx: (– addr)	Create a variable named xxx with initial value n; returns address when executed.
CONSTANT xxx	(n –) xxx: (– n)	Create a constant named xxx with value n; returns value when executed.
CODE xxx	(–)	Begin definition of assembly-language primitive operation named xxx.
:CODE	(–)	Used to create a new defining word, with execution-time "code routine" for this data type in assembly.
<BUILDS ... DOES>	does: (– addr)	Used to create a new defining word, with execution-time routine for this data type in higher-level Forth.

VOCABULARIES

CONTEXT	(– addr)	Returns address of pointer to context vocabulary (searched first).
CURRENT	(– addr)	Returns address of pointer to current vocabulary (where new definitions are put).
FORTH	(–)	Main Forth vocabulary (execution of FORTH sets CONTEXT vocabulary).
DEFINITIONS	(–)	Sets CURRENT vocabulary to CONTEXT.
VOCABULARY xxx	(–)	Create new vocabulary named xxx.
VLIST	(–)	Print names of all words in CONTEXT vocabulary.

MISCELLANEOUS AND SYSTEM

(	(–)	Begin comment, terminated by right paren on same line; space after (.
FORGET xxx	(–)	Forget all definitions back to and including xxx.
ABORT	(–)	Error termination of operation.
' xxx	(– addr)	Find the address of xxx in the dictionary; if used in definition, compile address.
HERE	(– addr)	Returns address of next unused byte in the dictionary.
PAD	(– addr)	Returns address of scratch area (usually 68 bytes beyond HERE).
IN	(– addr)	System variable containing offset into input buffer;
SP@	(– addr)	Returns address of top stack item.
ALLOT	(n –)	Leave a gap of n bytes in the dictionary.
,	(n –)	Compile a number into the dictionary.

MISCELLANEOUS AND SYSTEM

(	(-)	Begin comment, terminated by right paren on same line; sapce after (.
FORGET xxx	(-)	Forget all definitions back to and including xxx
ABORT	(-)	Error termination of operation
xxx	(- addr)	Find the address of xxx in the dictionary; if used in definition, compile address
HERE	(- addr)	Returns address of next unused byte in the dictionary
PAD	(- addr)	Returns address of scratch area (usually 68 bytes beyond HERE).
IN	(- addr)	System variable containing offset into input buffer
SP@	(- addr)	Returns address of top stack item
ALLOT	(n -)	Leave a gap of n bytes in the dictionary
.	(n -)	Compile a number into the dictionary

New FORTH Words: (available from FORTH dictionary)

CLS	- screen clear
CS	- turn cursor on
CSX	- turn cursor off
IV	- turn inverse video on
IVX	- turn inverse video off
U.	- unsigned number print (range 0 to 65535)
U.R.	- right justified unsigned print

Available with FORTH 1.2

CK	- turn clock on
CKX	- turn clock off
H	- enter hours
M	- enter minutes
S	- enter seconds (enter as 6 H 24 M 0 S)
PC	- switch to programmable character RAM
PCX	- switch to normal characters
DFORTH	- prints the definition of a user FORTH word (see E - 39.0for use and caution notes)

See also E - 34.0 to E - 38.0 for EDITOR words available

FORTH PROGRAMMING: Screen Operations

Although you must understand how the stack operates to do calculations, you can do much more with FORTH than imitate a programmable calculator. This section will show you how to define words to output text, numbers and some graphics onto the screen.

The FORTH word EMIT will output the ASCII number at the top of the stack to the screen. If the number is from 32 to 127 EMIT will put a letter, digit or other character onto the screen. Try typing in these commands to see how EMIT may be used.

```
65 EMIT  (press RETURN after each line)
75 EMIT
49 EMIT
42 EMIT
```

Look at the table ASCII characters. You should find the characters A K 1 * beside the numbers used above.

Use the word EMIT to put these characters on the screen:

P P) #

If you use a number less than 32, one of the screen functions will be activated instead. Some of the functions are:

Clear screen	12 EMIT		
Inverse video on	1 EMIT	Inverse video of	2 EMIT
Cursor on	6 EMIT	Cursor off	15 EMIT
Clock display on	16 EMIT	Clock display off	17 EMIT
Set cursor position	11 EMIT		
Delete current line	21 EMIT		

Try some of the above functions by using EMIT.

You are now ready to define some of your own FORTH words.

The colon : is used to begin a definition. IT MUST BE FOLLOWED BY A SPACE!
A semi-colon ; ends the definition.

The words to clear the screen, and turn the cursor and inverse video on and off have been included in the FORTH dictionary. Their symbols are CLS CS CSX IV IVX
(see E - 13.0)

ASCII CHARACTER TABLE

The members from 0 to 31 are reserved for special screen function controls.

CHARACTER	ASCII (decimal)	CHARACTER	ASCII (decimal)	CHARACTER	ASCII (decimal)
Space	32	@	64		96
!	33	A	65	a	97
"	34	B	66	b	98
#	35	C	67	c	99
$	36	D	68	d	100
%	37	E	69	e	101
&	38	F	70	f	102
'	39	G	71	g	103
(	40	H	72	h	104
)	41	I	73	i	105
*	42	J	74	j	106
+	43	K	75	k	107
,	44	L	76	l	108
-	45	M	77	m	109
.	46	N	78	n	110
/	47	O	79	o	111
0	48	P	80	p	112
1	49	Q	81	q	113
2	50	R	82	r	114
3	51	S	83	s	115
4	52	T	84	t	116
5	53	U	85	u	117
6	54	V	86	v	118
7	55	W	87	w	119
8	56	X	88	x	120
9	57	Y	89	y	121
:	58	Z	90	z	122
;	59	[	91	{	123
<	60	\	92	|	124
=	61	]	93	}	125
>	62	^	94	~	126
?	63	_	95	▯	127

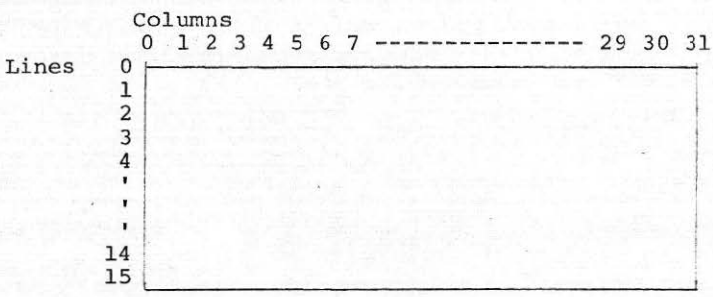
Note the difference between the digit 0 and the letter O

Define these words by typing in the following-

```
:   CK   16 EMIT  ;   -  displays the clock
:   CKX  17 EMIT  ;   -  turn display off
```

To use these words, type CK (press RETURN)
or CKX

SCREEN DISPLAY - The screen has 16 lines, 32 columns wide. The figure below shows how these are numbered.



If you want to put the cursor at column 20 of line 5 you must use EMIT three times.

11 EMIT	20 EMIT	5 EMIT
prepares to set position	column number	line number

All this must be done even before you can put anything there. To put the letter 'A' (ASCII 65) at this position, type in:

```
11 EMIT 20 EMIT 5 EMIT  65 EMIT
```

sets position — prints A

A word CPOS has been included in the FORTH dictionary to set the cursor position. This word is defined as -

```
:  CPOS 11 EMIT SWAP EMIT EMIT ;
```

Now to print A in column 20, lines 5, type:

```
20 5 CPOS 65 EMIT ( a bit easier?)
```

HOW AND WHY CPOS WORKS:

CPOS expects two numbers to be on the stack, the column number and the line number.

	STACK		OPERATION
Step 1	5 20 x x		- line number entered last - column number entered first
Step 2	11 5 20 x		CPOS 11 puts 11 on stack
Step 3	5 20 x x	11 ⟶	EMIT - first EMIT uses 11 to prepare to set the position
Step 4	20 5 x x		SWAP - we need column number first
Step 5	5 x x x	20 ⟶	EMIT - column number now used
Step 6	x x x x	5 ⟶	EMIT - line number now used

The stack has used the two numbers 20 and 5 to set the cursor position and it is now ready for further entries.

Now, use CLS and then CPOS four times to put four x's, one below the other, starting at column 25 on line 4.

You still have to type a lot to just to get one letter onto the screen. If you want several words, or a line of characters, there are other ways instead of using EMIT.

PRINTING TEXT

." is the FORTH word to print text. THERE IS NO SPACE BETWEEN . AND " BUT THERE IS AFTER "

" is the FORTH word to end the text printing.

Example : W1 ." PEGASUS" ; is the definition of W1 to print PEGASUS

Now to print PEGASUS starting at column 20, line 5, type

```
20 5 CPOS W1 CR     (the CR is used to begin a
                     new line after printing)
```

Let's define another new word to combine these words.

```
: PRINT CPOS W1 CR ;
```

This new word will expect two numbers, the column number and the line number.

To use, type 20 5 PRINT

EXERCISES (clear the screen after each exercise)

1. Define a word PRINT1 to print PEGASUS in inverse video. You may use IV and IVX to turn the inverse video on before printing and off afterwards.

2. The FORTH word SPACES will put blanks on the screen.

   IV 9 SPACES IVX will put 9 inverse blanks (white blocks) on the screen.
   Use this information to define a word 9SP to put 9 white blocks beginning at a specified column and line.

3. Define a word TASK to print PEGASUS near the middle of the screen, in normal video, with a white border on top, bottom, and sides.
   Break the task into several parts; define words and test them for each part. Then combine these words into the one word TASK which will do the entire operation.

(Answers are on the following page)

POSSIBLE ANSWERS

1. : PRINT1 CPOS IV WI IVX CR ; to use type 20 5 PRINT1

2. : 9SP CPOS IV 9 SPACES IVX CR ; to use type
20 5 9SP

3. You should break this task into several parts.

(a) : TB 10 6 9SP ; - top border

(b) Define another word

: 1SP CPOS IV 1 SPACES IVX CR ; NOTE - SPACES does not change

Use 1SP to provide the left and right borders.

: LB 10 7 1SP ; - left border

: RB 18 7 1SP ; - right border

(c) : PG 11 7 PRINT ; - prints PEGASUS

(d) : BB 10 8 9SP ; - bottom border

Now put all these parts together

: TASK CLS TB LB PG RB BB ;

To use, type TASK

COMBINING MATHS AND TEXT PRINTING

This section will describe a task using defined words for

(a) finding the cube of a number

(b) calculating cubes of 10 numbers

(c) printing headings and putting results in chart form

(a) If you want to cube a number, you need 2 duplicate numbers to multiply with the original

```
: CUBE DUP DUP * * ;
```

Type 5 CUBE . (the . is needed to print the result)

Check carefully how the definition of CUBE works.

(NOTE: The stack can only store numbers from -32768 to 32767 32 CUBE will cause stack to overflow)

(b) We must use a DO ------ LOOP to do a calculation more than once. Let's define a word to print the numbers from 1 to 10.

```
: COUNT 11 1 DO I . CR LOOP ;
```

Type CLS COUNT to test.

How COUNT works

```
11 1    - loop starts at 1 but stops at 10 , NOT 11
DO      - Start of Loop
I .     - prints this number
CR      - begins new line
LOOP    - end of loop
```

When the list was printed, the 0 of 10 was not in the one's column. Redefine COUNT using 6 .R instead of . and try COUNT now. (6 .R - sets up a block of 6 spaces with the one's column on right.

EXERCISE

If we want to print a list of cubes, we can use a loop similar to COUNT.

```
: 10 CUBES 11 1 DO I CUBE 6 .R CR LOOP ;
```

The only difference is that I is cubed before it is printed.

Next combine these ideas into one loop that will print number and its cube.

```
: CUBECHART 11 1 DO 4 SPACES I 6 .R 10 SPACES I CUBE
  6 .R CR LOOP ;
```

4 SPACES provides spaces on the left margin of the chart;
10 SPACES gives a gap between the numbers and their cubes.

(c) Finally, we need a word to print a heading for each column. You can do this yourself.

EXERCISE

Use what you learned about text printing and CUBECHART to do the following.

1. Clear the screen
2. Print on line 0 CUBES FROM 1 to 10
3. Print on line 2 in inverse video
   NUMBER CUBE
4. Print columns for number and its cube

POSSIBLE ANSWERS

```
: L0 ."  CUBES FROM 1 to 10" ;
: W1 ."  NUMBER" ;
: W2 ."  CUBE" ;
: L2 3 SPACES IV W1 IVX 13 SPACES IV W2 IVX ;
: CHART CLS L0   CR CR L2 CR CUBECHART ;
```

Your answer may have different methods of printing line 0 and 2.
If it works, it's right!

GUIDE TO DICTIONARY WORDS:

(see E - 11.0 to E - 12.0 for summary)

DEFINING WORDS	E - 23.0
MEMORY	E - 24.0
NUMBER BASES	E - 26.0
COMPARISON	E - 27.0
CONTROL STRUCTURES	E - 28.0
INPUT-OUTPUT	E - 31.0
MISCELLANEOUS	E - 33.0
DFORTH	E - 39.0
EDITOR	E - 34.0

FORTH DICTIONARY WORDS

DEFINING WORDS

(a) New words are defined with a COLON DEFINITION.

: begins the definition

; ends the definition

Example: : CK 16 EMIT ;

(b) Variables may be stored for later use.

0 VARIABLE NUM	defines NUM with an initial value 0
25 NUM !	changes value to 25 (! is pronounced "store")
NUM @	puts the value of NUM onto the stack (@ is pronounced "fetch")
NUM ?	prints the value of NUM

(NOTE- ? is the same as @ .)

(c) Constants may be defined for later use.

50 CONSTANT N	defines N with the initial value 50
The value of N cannot be changed once it is defined.	
N	puts the value of N onto the stack
N .	prints the value of N

NOTE: Observe the differences in putting the values of variables and constants onto the stack and for printing their values.

	VARIABLE	CONSTANT
Definition	0 VARIABLE NUM	50 CONSTANT N
Change value	25 NUM !	fixed value
Value onto stack	NUM @	N
Print Value	NUM @ .	N .
	or NUM ?	

MEMORY UTILISATION WORDS

Each address location in the computer can store 1 byte of 8 bits. One byte can represent numbers from 0 to 255; thus ASCII characters may be stored as 1 byte.
1 cell is 2 bytes for a total of 16 bits.
One cell can represent numbers from 0 to 65535 or
numbers from -32768 to 32767
FORTH uses this last range of numbers

IN MOST CASES: 1 byte is used for ASCII characters
1 cell is used for numbers

(a) @ (pronounced "fetch")

@ replaces a memory address with the number in the cell starting at this address.

C@ replaces an address with the contents in the byte at this address

Examples: NUM @ NUM was defined as a location for a variable
NUM puts the address on the stack.
@ replaces this address with the value of the variable

HEX BF00 C@ C@ replaces the hex address BF00 with the value in this address.
This value is now on the stack.

(b) ! (pronounced "store")

! and C! work in the same way as @ and C@ but contents are stored into cells or bytes.

Examples: 25 NUM !

HEX 41 BF00 C! (puts 41 (hex) into BF00)

(c) CMOVE is used to move bytes fron one address to another.

The screen addresses are from:

BE00 to BFFF in HEX

48640 to 49151 in DECIMAL

HEX BE00 BF00 20 CMOVE moves 20 (hex) bytes starting at BE00 to 20 (hex) bytes starting at BF00 .

In DECIMAL the operation becomes:

DECIMAL 48640 48896 32 CMOVE

(e) FILL ERASE BLANKS

A comment is required about ASCII characters shown on the screen. The ASCII code for A is 65 (decimal) If 65 is stored into the screen memory address, an INVERSE A will appear.

If 193 (i.e. 65 + 128) is stored into this screen memory address, then a normal A will appear

In HEX the comparable values are

41 inverse A

C1 normal A (C1 = 41 + 80)

FILL This word is used to fill an area of memory with a single character of your choice. For example

HEX BF00 1 41 FILL 1 location at BF00 is filled with an inverse A

HEX BF00 OD C1 FILL OD (hex) locations are filled with a normal A

In DECIMAL, these operations become:

DECIMAL 48896 1 65 FILL

DECIMAL 48896 13 193 FILL

ERASE and BLANKS These words fill an area of memory with nulls (i.e. ASCII 0) or blanks (ASCII 32 in decimal, 20 in hex)

In HEX

BF00 40 ERASE is the same as BF00 40 0 FILL

BF00 40 BLANKS is the same as BF00 40 20 FILL

NUMBER BASES

Calculations may be done in any number base.

DECIMAL	sets all operations to base 10
HEX	base 16
n BASE !	base n, where n is 2, 3, 4, etc.

Number Base Conversions:

To change 20 (decimal) to other bases, try this;

```
0 VARIABLE NUM          define NUM with an initial value 0
DECIMAL 20 NUM !        stores number in base 10
HEX NUM ?               prints the value in base 16
2 BASE ! NUM ?          prints the value in base  2
```

To change 1E (hex) to other bases;

```
0 VARIABLE NUM
HEX 1E NUM !
DECIMAL NUM ?
8 BASE ! NUM ?
2 BASE ! NUM ?
```

COMPARISON

There are 5 conditional tests available.

<	less than
>	greater than
=	equal
0<	less than zero
0=	equal to zero

Each test destroys the number or numbers compared on the stack and places a value on the stack -

1 if the condition is TRUE

0 if the condition is FALSE

Examples: (only the relevent one or two numbers on top of the stack are shown)

STACK BEFORE	COMPARISON	STACK AFTER
3 2	< (is 2<3 ?)	1 (true)
2 3	< (is 3<2 ?)	0 (false)
5 8	> (is 8>5 ?)	1 (true)
5 6	= (is 6 = 5 ?)	0 (false)
1	0< (is 1 less than zero?)	0 (false)

CONTROL STRUCTURES

(1) DO - - - - - - LOOP

DO takes 2 values from the stack
- the initial index (on top of the stack)
- the final index PLUS 1 (next on the stack)

EXAMPLE 5 1 DO - - - LOOP
The count starts at 1 and ends at 4 NOT 5
A word defined as
: CNT 5 1 DO I . LOOP ;
will count from 1 to 4
where I places the current index of the loop on the stack and . prints this index.
Define CNT and then type CNT to use.

(2) DO - - - - - +LOOP

The word +LOOP allows the index to change by any amount.

EXAMPLE : CNT2 10 2 DO I . 2 +LOOP ;
CNT2 will count from 2 to 8 by two's; the number 2 just before the +LOOP gives the interval of the loop.

(3) (a) IF --- ELSE --- THEN
(b) BEGIN --- UNTIL
(c) BEGIN --- WHILE --- REPEAT

All these structures can use the comparison words
< > = 0< 0=
to produce a true or false value on the stack. If a comparison is false, a zero is put on the stack; if true, a one goes on the stack.

NOTE: The comparison words destroy the values being compared and replace them by a number to indicate true or false.

EXAMPLES

(a) The IF --- ELSE --- THEN structures allows a choice of two branches to be taken.
As a simple example, we shall define a word TEST to print whether the top value on the stack is positive or negative, and then print the absolute value.

```
: TEST DUP 0< IF ." NEGATIVE " ELSE ." POSITIVE "
                THEN ABS 2 SPACES . ;
```

where DUP 0< makes a copy of the number and tests if the copy is less than zero

IF ." NEGATIVE " prints NEGATIVE if true

ELSE ." POSITIVE " prints POSITIVE if false

THEN ABS 2 SPACES . prints the absolute value of number

To use, type in a number, a space, and then TEST

The ELSE statement is optional in the structure and may be omitted if not required.

(b) BEGIN --- UNTIL

BEGIN marks the start of a sequence that may repeatedly be executed UNTIL a conditional test is false

EXAMPLE:

```
: INPUT CR BEGIN KEY DUP EMIT CR 32 =
                UNTIL ." SPACE" CR ;
```

where KEY puts the ASCII value of a key pressed onto the stack

DUP EMIT CR makes a copy and prints the ASCII character onto the screen followed by a carriage return

32 = tests if the ASCII value is 32 (for a space)

UNTIL the structure goes back to BEGIN until a space is entered, then it prints SPACE and stops

To use type INPUT (then press RETURN). Press any keys on the keyboard and they will be printed until you press the space bar.

(c) BEGIN --- WHILE --- REPEAT

This structure repeats while a condition test is TRUE.

EXAMPLE

```
: INPUT2 CR BEING KEY DUP 32 > WHILE EMIT
                CR REPEAT ." STOP " ;
```

This example is similar to the previous one except the character is printed while its ASCII value is greater than 32. If it is not greater, then STOP is printed. Try using INPUT2 the same way as INPUT.

INPUT-OUTPUT

String input: A string may be stored using the word EXPECT

0 VARIABLE STR 20 ALLOT	reserves 20 spaces at the address STR
STR 20 BLANKS	clears these spaces
STR 20 EXPECT (press RETURN)	waits for up to 20 characters to be typed in. Pressing RETURN will stop the input.

String output:

CR STR 20 -TRAILING TYPE	prints out the string stored at STR -TRAILING deletes any following blanks

Number Formatting:

You may wish to display numbers in certain formats, such as 12:24 for time displays.

Before a number may be formatted, it must be in double precision form and the sign of the number must be stored separately. This is easily done by using DUP ABS 0 before the formatting work <#

DUP	makes a signed copy
ABS	makes an unsigned number for formatting
0	provides a dummy high order part for the double prevision mode

EXAMPLES

```
: N DUP ABS 0 <# 41 HOLD #S SIGN 40 HOLD #>
TYPE ;
```

N will take a number on top of the stack and enclose it in brackets.

```
-100 N          will print        (-100)
```

How N works:

DUP ABS 0	prepares the single precision number for conversion
<#	begins the conversion
41 HOLD	since the conversion starts on the RIGHT the) bracket is required first
S	all significant digits are converted
SIGN	the sign is placed on the left if negative
40 HOLD	the (bracket is placed on the left
#>	the conversion ends
TYPE	the formatted number is printed

```
: TIME DUP ABS 0   #  #  #  58 HOLD  #S SIGN #   TYPE ;
```

2345 TIME will print 23.45

<#	begins the conversion
#	the rightmost number is converted
#	the next number is converted
58 HOLD	: is inserted
#S	the remaining numbers are converted
SIGN	the sign is placed on the left
#> TYPE	conversion ends and the number is printed

Miscellaneous Words

VLIST	prints the words in the dictionary, starting with the most recent definition
FORGET	deletes definitions up to and including the definition named

Advanced topics such as VOCABULARIES, CODE definitions and the construction < BUILDS --- DOES> will be treated in future manual updates and newsletters.

EDITOR WORDS

The lower case letters before a definition represent an address or number to be entered before the word is used. Address locations are usually given in hex; type HEX before the number is entered. When you wish to return to base 10, type DECIMAL

addr TEXTSTART - allows the user to define the memory location for the start of the text.
e.g. HEX B800 TEXTSTART

n BLOCKS - defines blocks of 128_{10} bytes beginning with the location of TEXTSTART.
e.g. 8 BLOCKS uses memory from B800 to BBFF

TEXT? - prints the start address of text and the number of blocks assigned

TEXTCLEAR - fills all blocks with blanks; this word is used when setting up the EDITOR

n B/CLR - fills block n with blanks

n B/VIEW - displays the contents of block n near the centre of the screen ready for entering or altering text

n B/REV - displays the contents of block n near the top of the screen for review only. NO editing is done in this part of the screen.

n B/COMP - compiles the text of block n

n B/INS - inserts a blank block at n and moves the following blocks up one; the last block must be clear or an error message will occur.

e.g. before 1 2 3 4 5 6 7 8 (8 is blank)
2 B/INS 1 2 3 4 5 6 7 8 (2 is blank)

n B/DEL - deletes block n and moves the following blocks down one

e.g. before 1 2 3 4 5 6 7 8
3 B/DEL 1 2 3 4 5 6 7 8 (8 now blank)

{ - begins entry or alteration of text in a block

NOTE: this work should be preceeded by n B/VIEW so that you are editing the correct block.
e.g. 3 B/VIEW {

NOTE: The cursor may be moved within the bock using:

CTRL E	- up
CTRL X	- down
CTRL D	- right
CTRL S	- left (avoid using BACKSPACE)
RETURN	- new line

} - ends the entry or editing of text in a block

NB - views next page

LB - views previous page

TEXTCOMP - compiles all text from block 1 onwards. Each block will be displayed as it is compiled. An error in a definition will stop the compilation. Remember to FORGET any definitions you are recompiling or you will get a REDEF --- NOT UNIQUE message.

TEXTSAVE - saves all blocks on cassette tape; a file name is requested

TEXTLOAD - loads text from cassette tape into the memory specified by TEXTSTART

USING THE EDITOR

For a system with 4K of RAM a suitable location to store text would be from B800 to BBFF - 8 blocks. The compiled FORTH definitions will be stored from the beginning of the Dictionary space upwards; the average program should easily fit in this space up to B7FF. The RAM above BC00 is used for the stack and system parameters.

To begin, use TEXTSTART BLOCKS TEXTCLEAR

For example
```
HEX  B800  TEXTSTART  8  BLOCKS
DECIMAL                to return to base 10
TEXTCLEAR              to clear all blocks
1  B/VIEW              to view block 1
{                      to begin editing
```
Now enter text, using the CTRL keys and RETURN to move the cursor. The word } ends the entry or alteration of text.

Block 1 will now contain any text you have entered. Definitions may be from 1 to 4 lines long. Begin a new line for each new definition. If you are starting a new definition, be certain you will have enough space or else begin the next block. The compiling ignores any blank lines. Blank lines are often useful for inserting changes later.
To display block 1 at the top of the screen, type 1 B/REV Now you can type 2 B/VIEW { to enter text to block 2 while still displaying block 1. You may display any block for revision while you are working on another block. Continue entering text until you are ready to test your definitions. You may compile and test a block at a time, beginning with block 1 using B/COMP or you may compile the entire text with TEXTCOMP. If you wish to make alterations, clear the screen CLS and use B/VIEW and { to edit any block. Then FORGET back to the first definition you changed and recompile that block and those following.

SAVING and LOADING TEXT

To save FORTH text, use TEXTSAVE A file name will be requested. The file name can be 8 characters long, but it is suggested to use .F as the last two characters to specify FORTH text. e.g. FILENAME? PROGR1.F

To save a compiled FORTH program, use SAVE as explained in the FORTH manual. Use .C for the end of the filename. It is always advisable to save text so you can make later revisions. If you only save the compiled program, any alterations may mean extensive retyping.
One option is to save the TEXT on one side of a cassette, and the compiled version on the other. The compiled version will load faster and free the text space for other text or programs.

To load text, use TEXTLOAD , to load the compiled version use LOAD.

DFORTH

If you define words without the use of the FORTH 1.2 text EDITOR, you lose the text for the definition when it scrolls off the screen. Quite often it is necessary to recall these definitions. This recall may be done using DFORTH.

e.g. DFORTH PROG will give the text of your definition for the word PROG

HOWEVER, there are certain limitations:

1) definitions should be kept simple. For example you should not combine IF ELSE THEN
or BEGIN END
or BEGIN IF AGAIN
or DO LOOP in one definition.
Instead, use a separate word for each structure required and then combine these new words into an additional word.

e.g. : INPUT BEGIN KEY DUP EMIT 32 = UNTIL ;
: PHRASE 5 1 DO INPUT LOOP CR ;

PHRASE will input 4 words from the keyboard each separated by a space.

2) definition containing ." " to print characters may take several seconds to DFORTH (be patient). Again, keep this construction separate from the others in 1) above.

3) certain synonyms are possible for words

IF ELSE THEN reappears as	IF ELSE ENDIF
BEGIN UNTIL	BEGIN END
BEGIN IF AGAIN	BEGIN WHILE REPEAT

If DFORTH should get 'hung up' ,possibly because of a faulty definition, or a word that is too complex, refer to E - 1.0 under the heading PANIC!

THEORY OF OPERATION - TECHNICAL SUMMARY

Page numbers refer to the circuit diagrams.

Page.1

An AC supply of between 6 and 15 volts is rectified and clamped to between 0 and 5 volts by the diode resistor combination. The 74LS14 Schmitt inverter provides rectangular pulses from this at 50Hz.

A falling edge then triggers both halves of a 74LS123 monostable producing a 1ms low going $\overline{\text{vsync}}$ pulse from one and a 2ms low going pulse from the other. These pulses are synchronized to $\overline{\text{hsync}}$ by one half of the 74LS74 ensuring that $\overline{\text{vsync}}$ and $\overline{\text{hsync}}$ are locked together. The rising edge of this second pulse clocks the other half of the 74LS74 to produce an active high output delayed by 1ms from the termination of $\overline{\text{vsync}}$. This in turn is gated with $\overline{\text{hsync}}$ by a 74LS32 OR gate to produce $\overline{\text{firq}}$ interrupt pulses to the processor once every $\overline{\text{hsync}}$ pulse while the output of the 74LS74 remains high. This is cleared under software control at the completion of each video frame via the clr bit of the 6821 PIA.

The $\overline{\text{hsync}}$ pulse is derived from the master microprocessor clock (e) which is divided by 64 by two 74LS93 counters. Their outputs are gated by a 74LS21 and 74LS00 to produce an 8us $\overline{\text{hsync}}$ pulse every 64us as required for video synchronization.

Page.2

The 6809 microprocessor has a 4MHz crystal providing a master clock (e) of 1MHz. Reset is accomplished at power on by an RC delay network and two 74LS14 Schmitt inverters. The non-maskable interrupt is utilized as an abort feature by coupling it to the on board PANIC push button. This button is debounced by an RC network and two more Schmitt inverters.

The main address and data buses are buffered by a 74LS245 and two 74LS241's to provide expansion capability via the SS-50 bus edge connector.

Page.3

The system chip selects are provided by three 74LS139's and some additional gating. Initially the top two address lines, a14 and a15 are decoded into four 16k blocks which are further decoded by the other gates. The bottom 16k block is decoded by another half of a 74LS139 and a12 and a13 to provide four 4k blocks. The bottom two of these ($\overline{\text{rom2}}$ and $\overline{\text{rom3}}$) may be used for two optional roms (see memory map). Note that the position of these two roms is selectable by jumpers on board.

The top 16k block is further decoded by half a 74LS139 and a12 and a13 to provide 4k block selects for $\overline{\text{rom1}}$, $\overline{\text{rom2}}$, and $\overline{\text{rom3}}$. $\overline{\text{Rom1}}$ is the system monitor rom and occupies the top 4k of the memory map, $\overline{\text{rom2}}$ and $\overline{\text{rom3}}$ are the optional roms described above. The middle 4k of this 16k block is used for I/O operations and is further divided by another half 74LS139 to give chip selects for $\overline{\text{pia1}}$, $\overline{\text{pia2}}$ and $\overline{\text{char ram}}$.

Note that these only occupy half of the 4k I/O block, the other half being reserved for I/O on additional boards. The next 16k block down is only partially utilized. The top 4k of it is decoded by half a 74LS139 to give four chip selects for $\overline{\text{ram1}}$ through $\overline{\text{ram4}}$ (1k each). These are the onboard ram chips, and the upper half of $\overline{\text{ram1}}$ is used for video character storage. The rest of the circuitry, the 74LS21, 74LS08 and 74LS00 are used to provide a $\overline{\text{sel}}$ strobe when none of the onboard devices are selected.

Page.4

$\overline{\text{Rom1}}$, $\overline{\text{rom2}}$, and $\overline{\text{rom3}}$ select 2532 EPROMS as described above. The 6821 PIA is dedicated to system usage. Its lines are used to control the keyboard, cassette and video circuitry. Eight lines of port A are used for two purposes. During video scan they control which row of each character is being displayed and during keyboard scan they set up which row of the key matrix is being scanned. The remaining two lines of port A are used for the cassette interface. One provides serial data out via an RC network to the recorders microphone input and the other receives data in from the recorders earphone output. This data is first clipped by diodes and then squared by the 74LS14 and 74LS04.

Three lines of port B are used to provide the video control signals page, char and blank. A fourth control pulse, clr, is provided by the pia's handshaking output. The remaining four port B data lines are configured as inputs from the keyboard columns. These are also fed to the 74LS20 so that any

low transition (keypressed) will result in a pia handshaking interrupt.

Page.5

The second (optional)pia is completely available to the user and all port lines are brought out to an edge connector.
The expandable on board ram consists of six 2114's selected by $\overline{\text{ram2}}$ through $\overline{\text{ram4}}$.

Page.6

The 74LS20 and the 74LS08 gating provides a select pulse to the 2114 ram chips if one of three conditions is met. Firstly if page is low (signifying an access to the character generator ram), secondly in $\overline{\text{ram1}}$ is low (signifying a normal processor ram access) and thirdly if any access to $(\text{F600} - \text{F7FF})_{16}$ occurs (signifying a video scan cycle). The read/write to this ram is organized so that only reads occur during page access via the 74LS00 gating (see software theory of operation).
The 74LS245 gates the raml access in the normal fashion. During video access data from the ram is fed to the programmable character generator ram and the rom character generator (6671O). These provide the necessary video data and character row selects are derived from the pia as previously described.
On the programmable character ram, bit seven of the input data selects between the ram chips (via a 74LS86) and the r/w is

gated via a 74LS32 so that it is always high (read) except for when the char ram select is active.

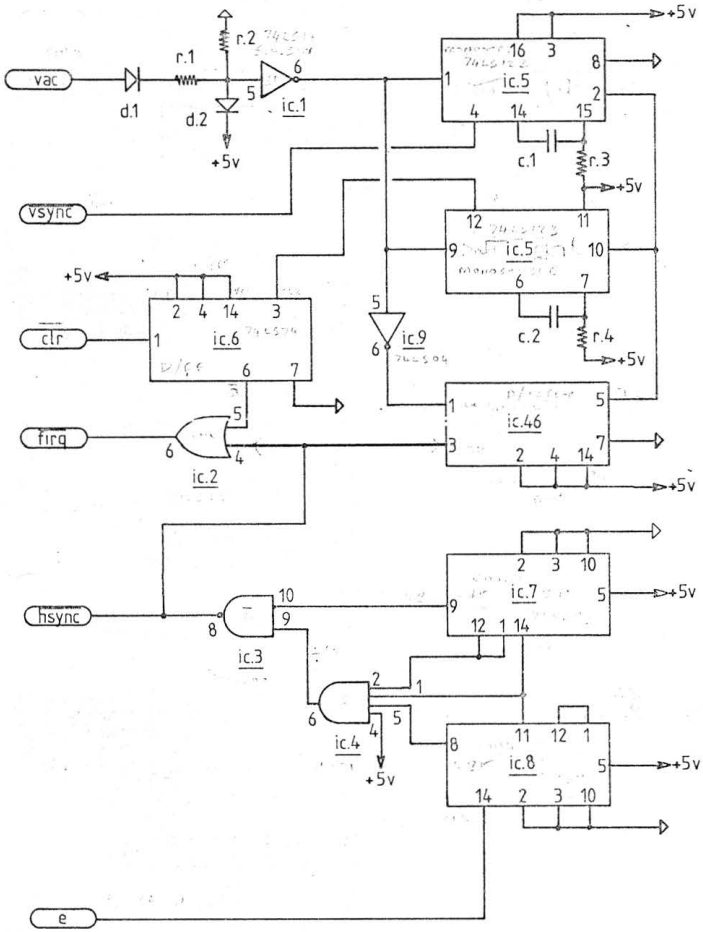
Page.7

The outputs from the two character generators are selected by the char line from the pia via two 74LS157 multiplexers. The selected output (char rom if char is low, char ram if char is high) is then loaded into a 74LS165 shift register. The clock timing for the shift register is generated by the 74LS04 and the series of RC networks, synchronized to the master clock (e). The 74LS245 data buffer provides microprocessor access to the programmable character generator ram.

Page.8

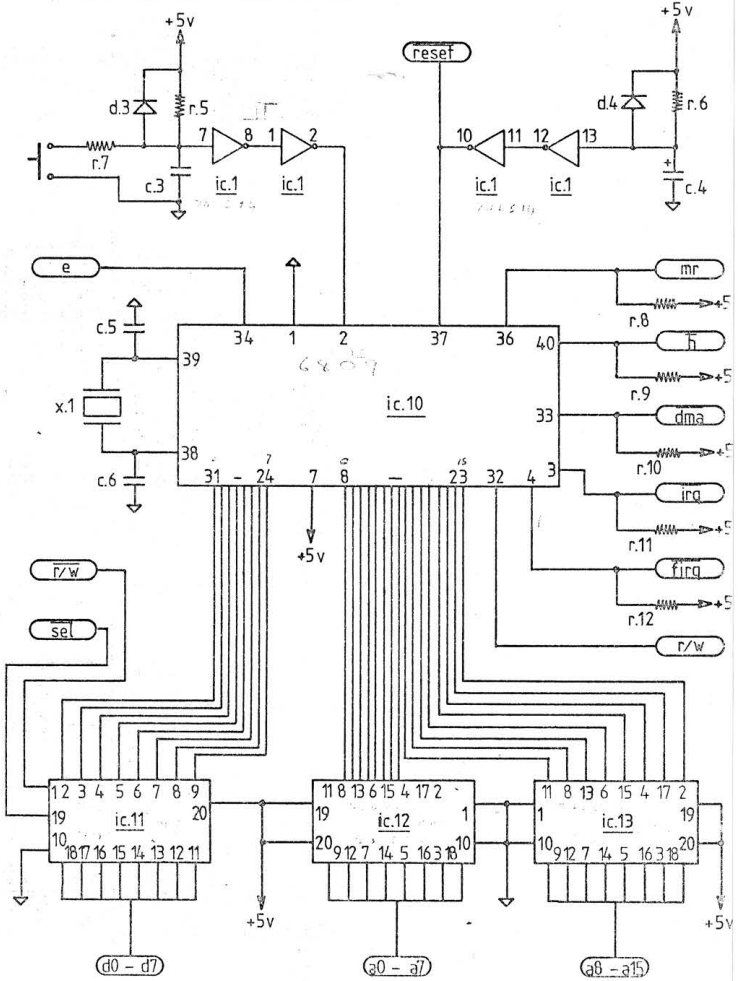
The 74LS157 multiplexes the eight columns of the keyboard (each with a pullup resistor) to four lines fed to the key inputs of the system pia. This is controlled by the pia's asc line.
The most significant data bit from the video ram is fed to half the 74LS74 which is clocked by the shift register load pulse. This synchronizes it with each new byte of video data. The output from this flip flop is then used to selectively invert video data via a 74LS86. This provides individual polarity control of the on screen characters.
Another 74LS32 is used to control blanking of video data via the blank signal from the system pia.Video data and sync are combined by a 4066 to produce composite video output.

Aamber Pegasus



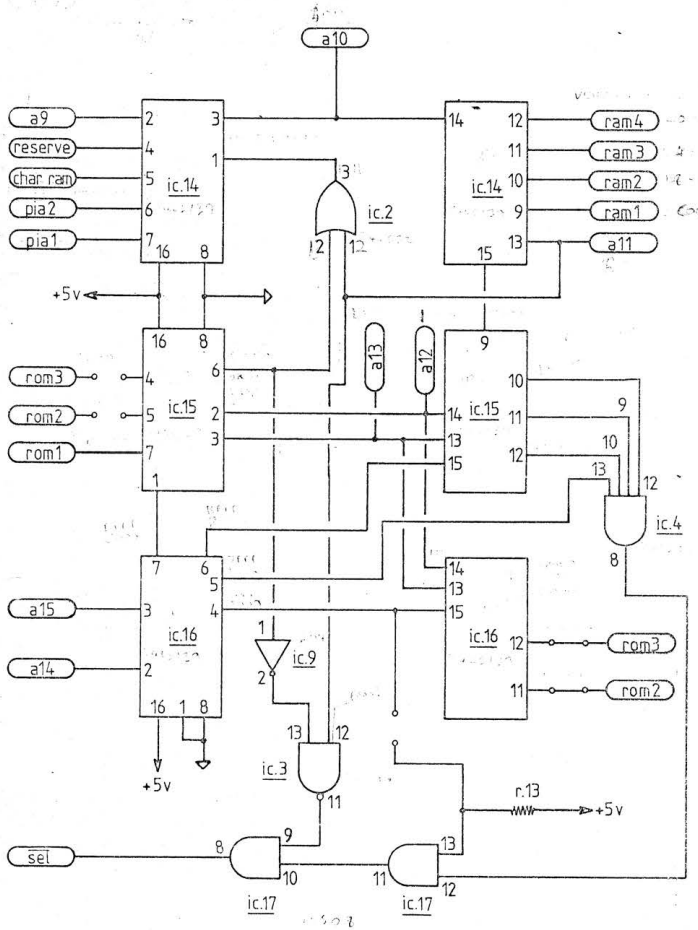
Aamber Pegasus

SJH 1/81 V1.0



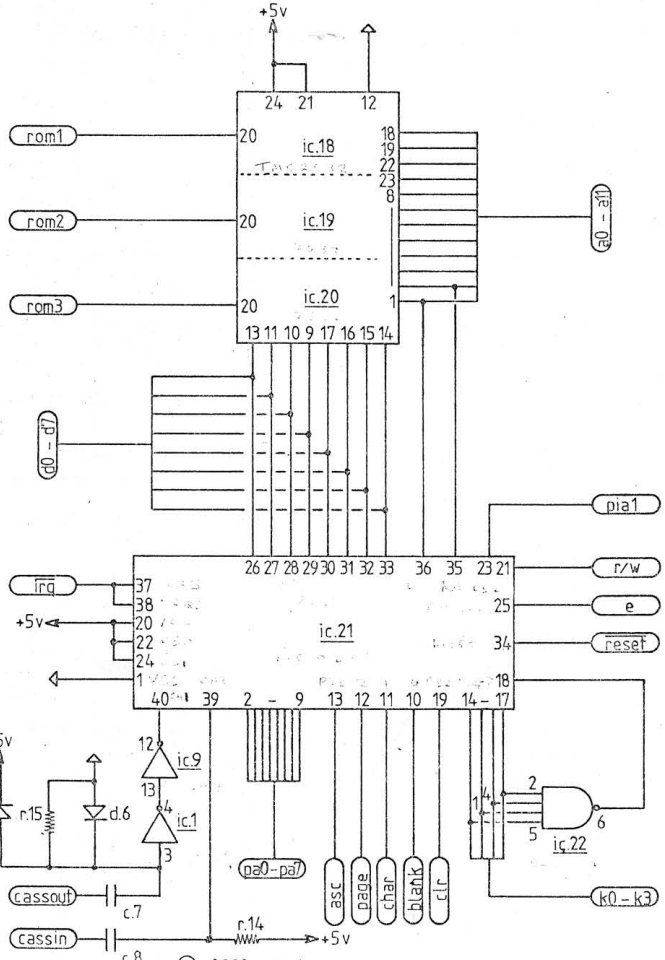
Aamber Pegasus

SJH 1/81 V1.0

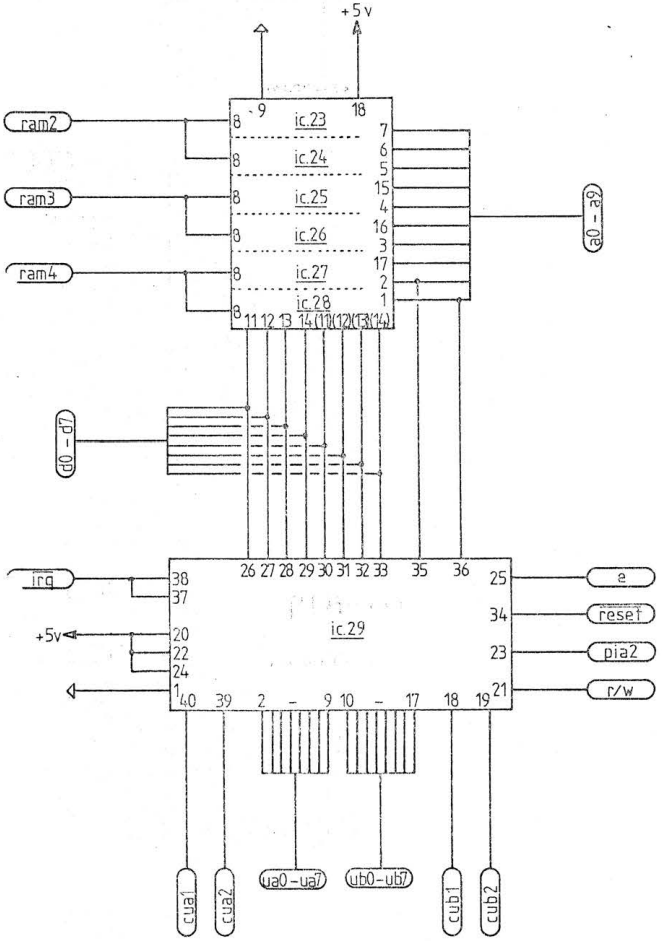


3-08

Aamber Pegasus

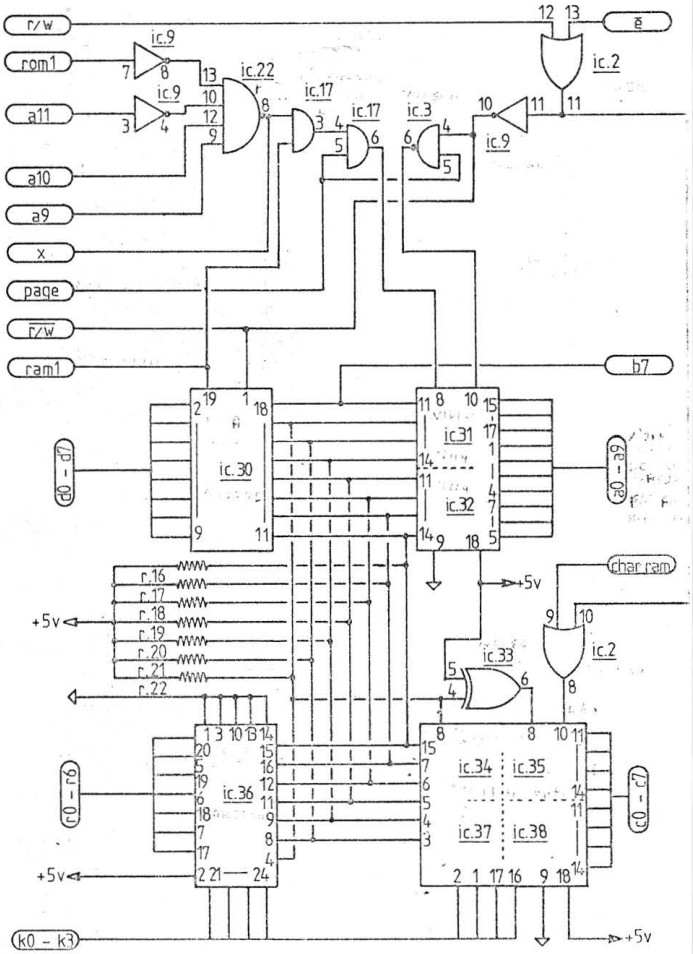


Aamber Pegasus



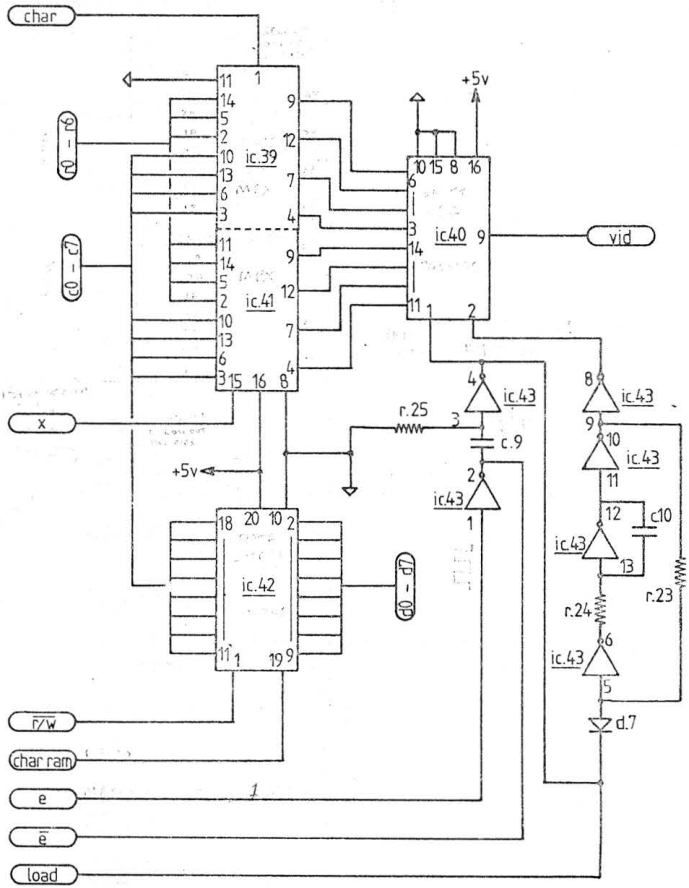
Aamber Pegasus

SJH 1/81 V1.0

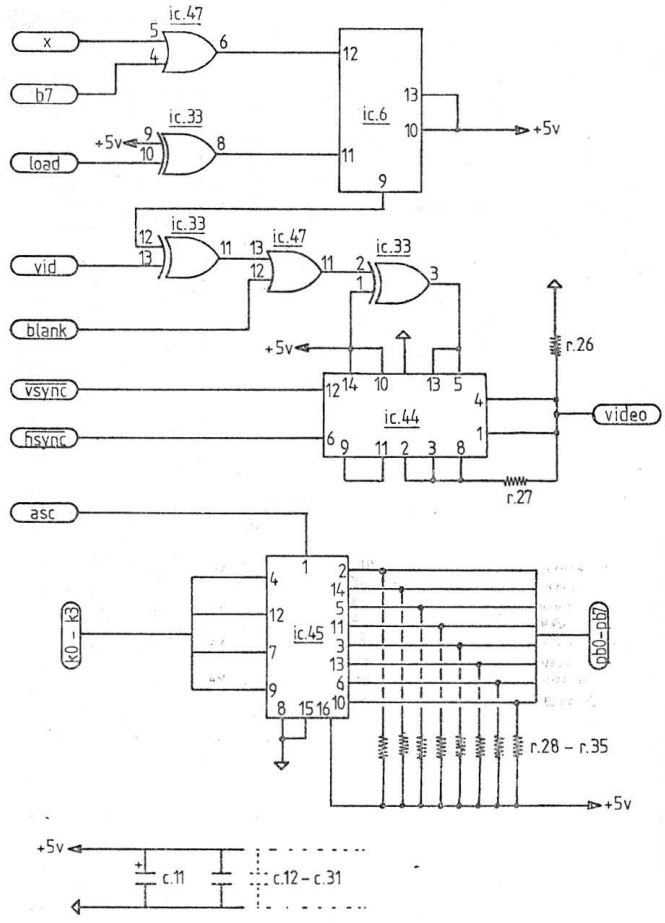


Aamber Pegasus

SJH 1/81 V1.0



Aamber Pegasus



Aamber Pegasus

Parts List:

IC.1	74LS14		IC.24	MC2114-30	*
IC.2	74LS32		IC.25	MC2114-30	*
IC.3	74LS00		IC.26	MC2114-30	*
IC.4	74LS21		IC.27	MC2114-30	*
IC.5	74LS123		IC.28	MC2114-30	*
IC.6	74LS74		IC.29	MC6821	*
IC.7	74LS93		IC.30	74LS245	
IC.8	74LS93		IC.31	MC2114-30	
IC.9	74LS04		IC.32	MC2114-30	
IC.10	MC6809		IC.33	74LS86	
IC.11	74LS245	*	IC.34	MC2114-30	*
IC.12	74LS241	*	IC.35	MC2114-30	*
IC.13	74LS241	*	IC.36	MC66710	
IC.14	74LS139		IC.37	MC2114-30	*
IC.15	74LS139		IC.38	MC2114-30	*
IC.16	74LS139		IC.39	74LS157	
IC.17	74LS08		IC.40	74LS165	
IC.18	TMS2532		IC.41	74LS157	
IC.19	TMS2532	*	IC.42	74LS245	*
IC.20	TMS2532	*	IC.43	74LS04	
IC.21	MC6821		IC.44	MC14066	
IC.22	74LS20		IC.45	74LS157	
IC.23	MC2114-30	*	IC.46	74LS74	

* denotes optional components

Aamber Pegasus

Parts List:

R.1	220	R.14	470
R.2	220	R.15	10k
R.3	27k	R.16 - R.22	1k
R.4	8.2k	R.23	1k
R.5	100k	R.24	33
R.6	1m	R.25	470
R.7	1k	R.26	390
R.8 - R.12	1k	R.27	2.7k
R.13	1k	R.28 - R.35	1k

all resistors ¼w carbon and all values in ohms

D.1 - D.7 IN4148

X.1 4.0 MHz Crystal

C.1	150nf mylar	C.7	20nf disc ceramic
C.2	330nf mylar	C.8	100nf disc ceramic
C.3	100nf disc ceramic	C.9	47pf disc ceramic
C.4	1uf 35v tantalum	C.10	1nf disc ceramic
C.5	27pf disc ceramic	C.11	68uf 16v tantalum
C.6	27pf disc ceramic	C.12 - C.31	0.1uf disc ceramic