LISTING OF MONITOR APPENDIX

***** 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.

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.

GRAFIX

Print CHR(27), CHR(N) (N = character to be changed)
followed by 16 off
Print CHR(?) (? = code for each line)

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

INPUT $F802
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
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
Re-entry point into machine language monitor

CASSOUT $F808
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
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
Character in A reg. is converted from lower case to upper case. Also, '.' becomes '<' and ',' becomes '>'. All registers preserved except A, CC

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

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

GETBYTE $F812
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
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
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
See discussion of Pegasus graphics capabilities. All registers preserved except X.

GETDIG $F81A
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
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
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
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
Entry point back to Menu selection mode in Monitor

HEXTEST $F824
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
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
Equivalent to control-L: clears the video screen. Does not call ECHO. All registers preserved except CC

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

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

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

NUL $F830
Nul function - has no effect

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

WORDOUT FBBE OUTPUT TO ECHO EVERYTHING IN D

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

SYSTEM MONITOR RAM ALLOCATION $BD00 - $BDFF

APPENDIX 4

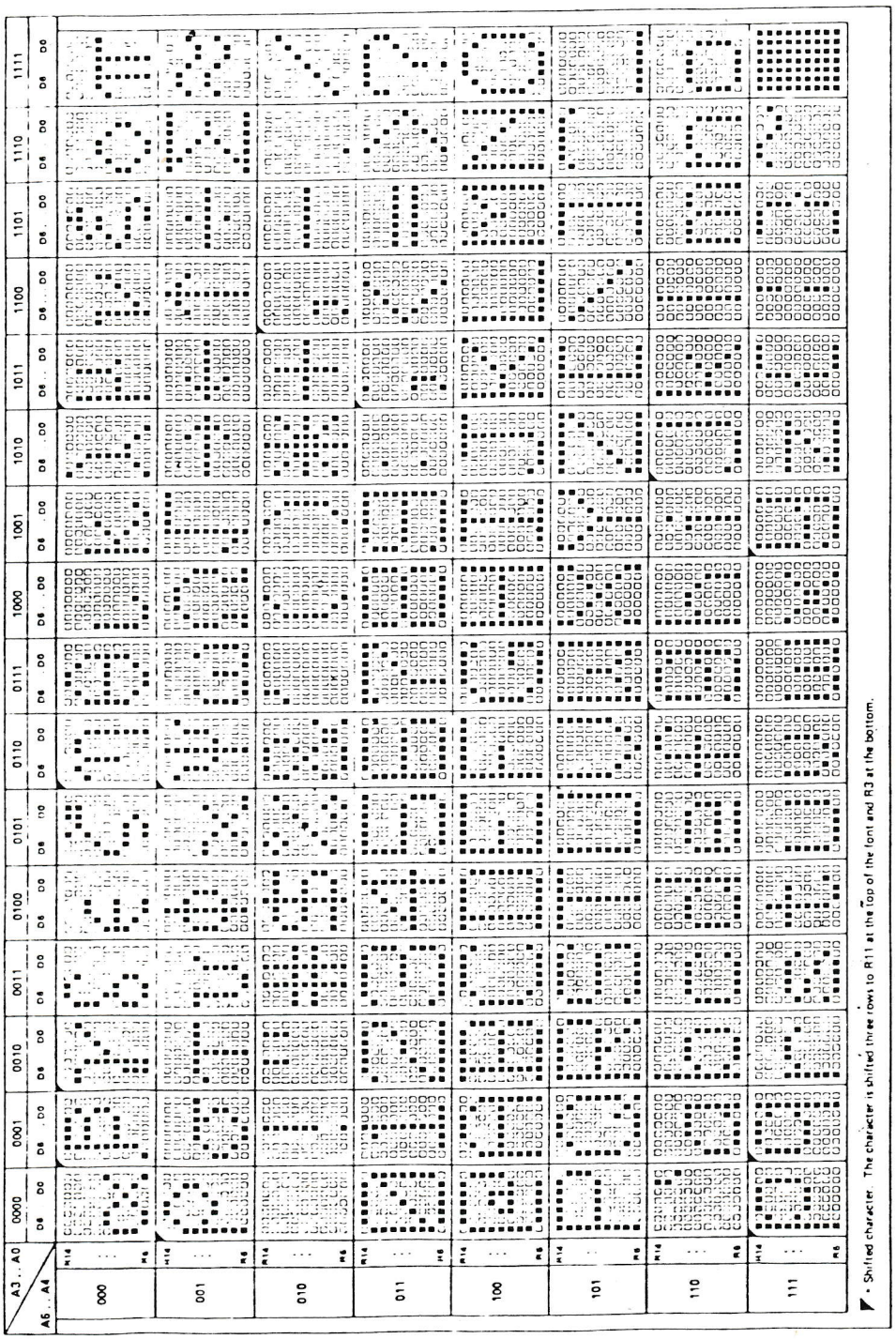


FIGURE 14 – MCM6571 PATTERN

APPENDIX 5

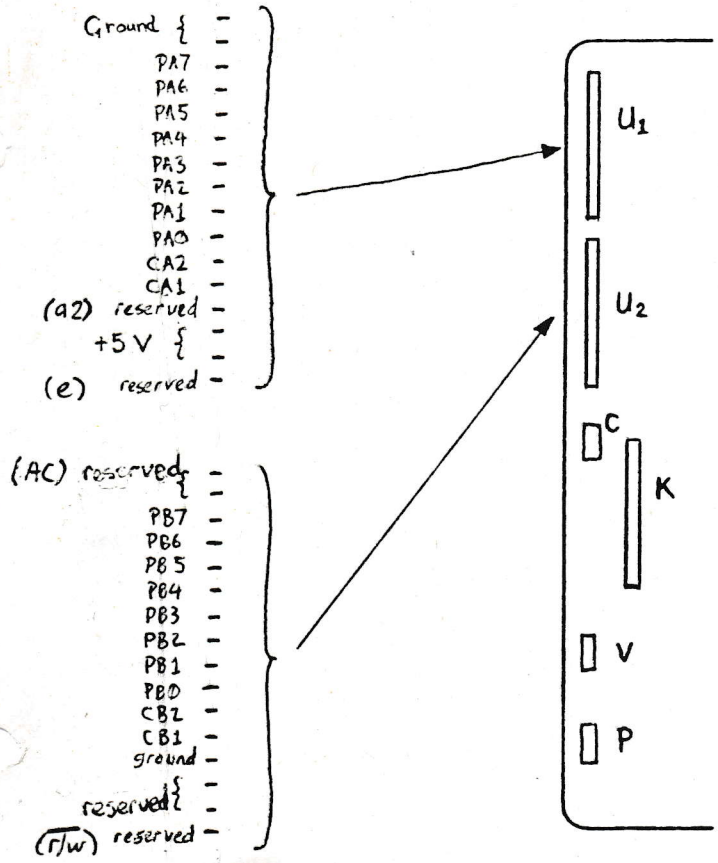
Memory Map for Aamber Pegasus - June 1981

F000..FFFF	System Monitor, in EPROM Socket 1
E818..EFFF	Reserved for future expansion
E810	Port 4 General Purpose user interface *
E808	Port 3 Digital Cassette interface *
E800	Port 2 Printer Port *
	* Requires I/O Expansion Option
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	512 bytes for video RAM *
BDA0..BDFF	96 bytes for System Monitor variables *
B000..BD9F	Standard User RAM, stack = $BD9F *
	* 4K on-board RAM
A000..AFFF	)
9000..9FFF	)
8000..8FFF	)
7000..7FFF	) 36K of address space reserved
6000..6FFF	) for RAM expansion under DAT control
5000..5FFF	)
4000..4FFF	)
3000..3FFF	)
2000..2FFF	)
1000..1FFF	EPROM Socket 2
0000..0FFF	EPROM Socket 3

APPENDIX 6

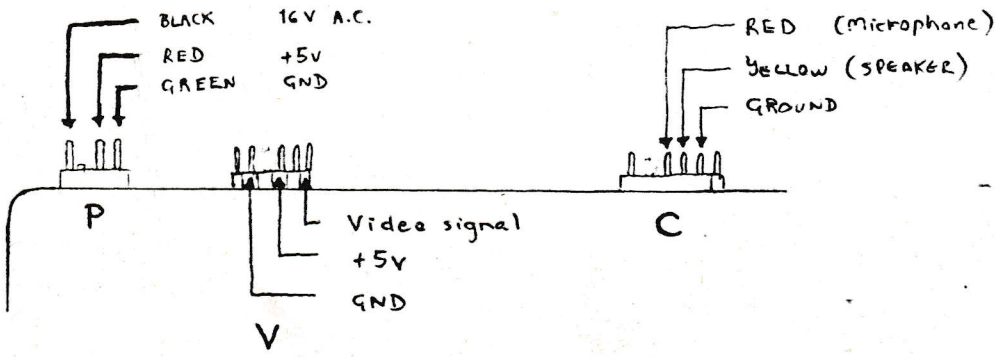
USER P I A CONNECTIONS

(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 PRFL / PRINTER FLAG 00 screen only
01 Print only
02 " & screen.
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
Reserved

$BDCE-F

Program counter used by GO command

$BDD0-3
Reserved

$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

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

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

$BDFC INPUT VECTOR

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

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

$BDEA IRQVECTOR

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 SWl1 VECTOR

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

$BDEE SWl2 VECTOR

The SWl2 instruction will vector though here.

$BDFO SWl3 VECTOR

The SWl3 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.

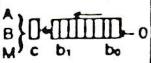
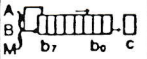
$BDFE OUTPUT VECTOR

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 8

6809 ADDRESSING MODES

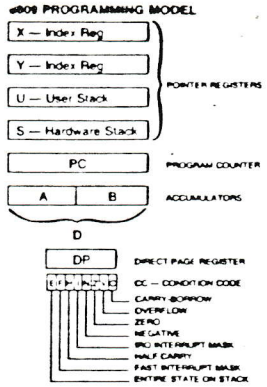
6809 ADDRESSING MODES

INSTRUCTION/ FORMS		INHERENT			DIRECT			EXTENDED			IMMEDIATE			INDEXED[1]			RELATIVE			DESCRIPTION	5 H	3 N	2 Z	1 V	0 C
		OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~[5]	#						
ABX		3A	3	1																B + X → X (UNSIGNED)	•	•	•	•	•
ADC	ADCA				99	4	2	B9	5	3	89	2	2	A9	4+	2+				A + M + C → A	↕	↕	↕	↕	↕
	ADCB				D9	4	2	F9	5	3	C9	2	2	E9	4+	2+				B + M + C → B	↕	↕	↕	↕	↕
ADD	ADDA				9B	4	2	BB	5	3	8B	2	2	AB	4+	2+				A + M → A	↕	↕	↕	↕	↕
	ADDB				DB	4	2	FB	5	3	CB	2	2	EB	4+	2+				B + M → B	↕	↕	↕	↕	↕
	ADDD				D3	6	2	F3	7	3	C3	4	3	E3	6+	2+				D + M M + 1 → D	•	↕	↕	↕	↕
AND	ANDA				94	4	2	B4	5	3	84	2	2	A4	4+	2+				A ∧ M → A	•	↕	↕	0	•
	ANDB				D4	4	2	F4	5	3	C4	2	2	E4	4+	2+				B ∧ M → B	•	↕	↕	0	•
	ANDCC										1C	3	2							CC ∧ IMM → CC	7	7	7	7	7
ASL	ASLA	48	2	1																	↕	↕	↕	↕	↕
	ASLB	58	2	1																	↕	↕	↕	↕	↕
	ASL				08	6	2	78	7	3				68	6+	2+				M c b1 b0	↕	↕	↕	↕	↕
ASR	ASRA	47	2	1																	↕	↕	↕	•	↕
	ASR	57	2	1																	↕	↕	↕	•	↕
	ASR				07	6	2	77	7	3				67	6+	2+				M b7 b0 c	↕	↕	↕	•	↕
BCC	BCC																24	3	2	Branch C = 0	•	•	•	•	•
	LBCC																10 24	5(6)	4	Long Branch C = 0	•	•	•	•	•
BCS	BCS																25	3	2	Branch C = 1	•	•	•	•	•
	LBCS																10 25	5(6)	4	Long Branch C = 1	•	•	•	•	•
BEQ	BEQ																27	3	2	Branch Z = 0	•	•	•	•	•
	LBEQ																10 27	5(6)	4	Long Branch Z = 0	•	•	•	•	•
BGE	BGE																2C	3	2	Branch ≥ Zero	•	•	•	•	•
	LBGE																10 2C	5(6)	4	Long Branch ≥ Zero	•	•	•	•	•
BGT	BGT																2E	3	2	Branch > Zero	•	•	•	•	•
	LBGT																10 2E	5(6)	4	Long Branch > Zero	•	•	•	•	•
BHI	BHI																22	3	2	Branch Higher	•	•	•	•	•
	LBHI																10 22	5(6)	4	Long Branch Higher	•	•	•	•	•
BHS	BHS																24	3	2	Branch Higher or Same	•	•	•	•	•
	LBHS																10 24	5(6)	4	Long Branch Higher or Same	•	•	•	•	•
BIT	BITA				95	4	2	B5	5	3	85	2	2	A5	4+	2+				Bit Test A (M ∧ A)	•	↕	↕	0	•
	BITB				D5	4	2	F5	5	3	C5	2	2	E5	4+	2+				Bit Test B (M ∧ B)	•	↕	↕	0	•
BLE	BLE																2F	3	2	Branch ≤ Zero	•	•	•	•	•
	LBLE																10 2F	5(6)	4	Long Branch ≤ Zero	•	•	•	•	•

INSTRUCTION/ FORMS		INHERENT			DIRECT			EXTENDED			IMMEDIATE			INDEXED[1]			RELATIVE			DESCRIPTION	5 H	3 N	2 Z	1 V	0 C
		OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~[5]	#						
BLO	BLO																25	3	2	Branch Lower	•	•	•	•	•
	LBLO																10 25	5(6)	4	Long Branch Lower	•	•	•	•	•
BLS	BLS																23	3	2	Branch Lower or Same	•	•	•	•	•
	LBLS																10 23	5(6)	4	Long Branch Lower or Same	•	•	•	•	•
BLT	BLT																2D	3	2	Branch < Zero	•	•	•	•	•
	LBLT																10 2D	5(6)	4	Long Branch < Zero	•	•	•	•	•
BMI	BMI																2B	3	2	Branch Minus	•	•	•	•	•
	LBMI																10 2B	5(6)	4	Long Branch Minus	•	•	•	•	•
BNE	BNE																26	3	2	Branch Z ≠ 0	•	•	•	•	•
	LBNE																10 26	5(6)	4	Long Branch Z ≠ 0	•	•	•	•	•
BPL	BPL																2A	3	2	Branch Plus	•	•	•	•	•
	LBPL																10 2A	5(6)	4	Long Branch Plus	•	•	•	•	•
BRA	BRA																20	3	2	Branch Always	•	•	•	•	•
	LBRA																16	5	3	Long Branch Always	•	•	•	•	•
BRN	BRN																21	3	2	Branch Never	•	•	•	•	•
	LBRN																10 21	5	4	Long Branch Never	•	•	•	•	•
BSR	BSR																8D	7	2	Branch to Subroutine	•	•	•	•	•
	LBSR																17	9	3	Long Branch to Subroutine	•	•	•	•	•
BVC	BVC																28	3	2	Branch V = 0	•	•	•	•	•
	LBVC																10 28	5(6)	4	Long Branch V = 0	•	•	•	•	•
BVS	BVS																29	3	2	Branch V = 1	•	•	•	•	•
	LBVS																10 29	5(6)	4	Long Branch V = 1	•	•	•	•	•
CLR	CLRA	4F	2	1																0 → A	•	0	1	0	0
	CLRB	5F	2	1																0 → B	•	0	1	0	0
	CLR				0F	6	2	7F	7	3				6F	6+	2+				0 → M	•	0	1	0	0
CMP	CMPA				91	4	2	B1	5	3	81	2	2	A1	4+	2+				Compare M from A	•	↕	↕	↕	↕
	CMPB				D1	4	2	F1	5	3	C1	2	2	E1	4+	2+				Compare M from B	•	↕	↕	↕	↕
	CMPD				10 93	7	3	10 B3	8	4	10 83	5	4	10 A3	7+	3+				Compare M: M + 1 from D	•	↕	↕	↕	↕
	CMPS				11 9C	7	3	11 BC	8	4	11 8C	5	4	11 AC	7+	3+				Compare M: M + 1 from S	•	↕	↕	↕	↕
	CMPU				11 93	7	3	11 B3	8	4	11 83	5	4	11 A3	7+	3+				Compare M: M + 1 from U	•	↕	↕	↕	↕
	CMPX				9C	6	2	BC	7	3	8C	4	3	AC	6+	2+				Compare M: M + 1 from X	•	↕	↕	↕	↕
	CMPY				10 9C	7	3	10 BC	8	4	10 8C	5	4	10 AC	7+	3+				Compare M: M + 1 from Y	•	↕	↕	↕	↕

INSTRUCTION/ FORMS		INHERENT OP	~	#	DIRECT OP	~	#	EXTENDED OP	~	#	IMMEDIATE OP	~	#	INDEXED[1] OP	~	#	RELATIVE OP	~[5]	#	DESCRIPTION	5 H	3 N	2 Z	1 V	0 C
COM	COMA	43	2	1																$\bar{A} \rightarrow A$	•	↕	↕	0	1
	COMB	53	2	1																$\bar{B} \rightarrow B$	•	↕	↕	0	1
	COM				03	6	2	73	7	3				63	6+	2+				$\bar{M} \rightarrow M$	•	↕	↕	0	1
CWAI		3C	20	2							3C	20	2							CC ∧ IMM →CC Wait for Interrupt					7
DAA		19	2	1																Decimal Adjust A	•	↕	↕	0	↕
DEC	DECA	4A	2	1																A − 1 → A	•	↕	↕	↕	•
	DECB	5A	2	1																B − 1 → B	•	↕	↕	↕	•
	DEC				0A	6	2	7A	7	3				6A	6+	2+				M − 1 → M	•	↕	↕	↕	•
EOR	EORA				98	4	2	B8	5	3	88	2	2	A8	4+	2+				A ∨ M → A	•	↕	↕	0	•
	EORB				D8	4	2	F8	5	3	C8	2	2	E8	4+	2+				B ∨ M → B	•	↕	↕	0	•
EXG	R1, R2	1E	7	2																R1 ↔ R2[3]	•	•	•	•	•
INC	INCA	4C	2	1																A + 1 → A	•	↕	↕	↕	•
	INCB	5C	2	1																B + 1 → B	•	↕	↕	↕	•
	INC				0C	6	2	7C	7	3				6C	6+	2+				M + 1 → M	•	↕	↕	↕	•
JMP					0E	3	2	7E	4	3				6E	3+	2+				EA[3] → PC	•	•	•	•	•
JSR					9D	7	2	BD	8	3				AD	7+	2+				Jump to Subroutine	•	•	•	•	•
LD	LDA				96	4	2	B6	5	3	86	2	2	A6	4+	2+				M → A	•	↕	↕	0	•
	LDB				D6	4	2	F6	5	3	C6	2	2	E6	4+	2+				M → B	•	↕	↕	0	•
	LDD				DC	5	2	FC	6	3	CC	3	3	EC	5+	2+				M: M + 1 → D	•	↕	↕	0	•
	LDS				10 DE	6	3	10 FE	7	4	10 CE	4	4	10 EE	6+	3+				M: M + 1 → S	•	↕	↕	0	•
	LDU				DE	5	2	FE	6	3	CE	3	3	EE	5+	2+				M: M + 1 → U	•	↕	↕	0	•
	LDX				9E	5	2	BE	6	3	8E	3	3	AE	5+	2+				M: M + 1 → X	•	↕	↕	0	•
	LDY				10 9E	6	3	10 BE	7	4	10 8E	4	4	10 AE	6+	3+				M: M + 1 → Y	•	↕	↕	0	•
LEA	LEAS													32	4+	2+				EA[3] → S	•	•	•	•	•
	LEAU													33	4+	2+				EA[3] → U	•	•	•	•	•
	LEAX													30	4+	2+				EA[3] → X	•	•	↕	•	•
	LEAY													31	4+	2+				EA[3] → Y	•	•	↕	•	•
LSL	LSLA	48	2	1																A, B, M: C b7 b6 b5 b4 b3 b2 b1 b0 ← 0	•	↕	↕	↕	↕
	LSLB	58	2	1																	•	↕	↕	↕	↕
	LSL				08	6	2	78	7	3				68	6+	2+					•	↕	↕	↕	↕
LSR	LSRA	44	2	1																A, B, M: 0 → b7 b6 b5 b4 b3 b2 b1 b0 → C	•	0	↕	•	↕
	LSRB	54	2	1																	•	0	↕	•	↕
	LSR				04	6	2	74	7	3				64	6+	2+					•	0	↕	•	↕
MUL		3D	11	1																A × B → D (Unsigned)	•	•	↕	•	↕
NEG	NEGA	40	2	1																$\bar{A}$ + 1 → A	↕	↕	↕	↕	↕
	NEGB	50	2	1																$\bar{B}$ + 1 → B	↕	↕	↕	↕	↕
	NEG				00	6	2	70	7	3				60	6+	2+				$\bar{M}$ + 1 → M	↕	↕	↕	↕	↕
NOP		12	2	1																No Operation	•	•	•	•	•
OR	ORA				9A	4	2	BA	5	3	8A	2	2	AA	4+	2+				A ∨ M → A	•	↕	↕	0	•
	ORB				DA	4	2	FA	5	3	CA	2	2	EA	4+	2+				B ∨ M → B	•	↕	↕	0	•
	ORCC										1A	3	2							CC ∨ IMM → CC					7
PSH	PSHS										34	5+[4]	2							Push Registers on S Stack	•	•	•	•	•
	PSHU										36	5+[4]	2							Push Registers on U Stack	•	•	•	•	•

INSTRUCTION/ FORMS		INHERENT OP	~	#	DIRECT OP	~	#	EXTENDED OP	~	#	IMMEDIATE OP	~	#	INDEXED[1] OP	~	#	RELATIVE OP	~[5]	#	DESCRIPTION	5 H	3 N	2 Z	1 V	0 C
PUL	PULS										35	5+[4]	2							Pull Registers from S Stack	•	•	•	•	•
	PULU										37	5+[4]	2							Pull Registers from U Stack	•	•	•	•	•
ROL	ROLA	49	2	1																A, B, M: C b7 b6 b5 b4 b3 b2 b1 b0 ←	•	↕	↕	↕	↕
	ROLB	59	2	1																	•	↕	↕	↕	↕
	ROL				09	6	2	79	7	3				69	6+	2+					•	↕	↕	↕	↕
ROR	RORA	46	2	1																A, B, M: → b7 b6 b5 b4 b3 b2 b1 b0 → C	•	↕	↕	•	↕
	RORB	56	2	1																	•	↕	↕	•	↕
	ROR				06	6	2	76	7	3				66	6+	2+					•	↕	↕	•	↕
RTI		3B	6/15	1																Return From Interrupt					7
RTS		39	5	1																Return From Subroutine	•	•	•	•	•
SBC	SBCA				92	4	2	B2	5	3	82	2	2	A2	4+	2+				A − M − C → A	↕	↕	↕	↕	↕
	SBCB				D2	4	2	F2	5	3	C2	2	2	E2	4+	2+				B − M − C → B	↕	↕	↕	↕	↕
SEX		1D	2	1																Sign Extend B into A	•	↕	↕	0	•
ST	STA				97	4	2	B7	5	3				A7	4+	2+				A → M	•	↕	↕	0	•
	STB				D7	4	2	F7	5	3				E7	4+	2+				B → M	•	↕	↕	0	•
	STD				DD	5	2	FD	6	3				ED	5+	2+				D → M: M + 1	•	↕	↕	0	•
	STS				10 DF	6	3	10 FF	7	4				10 EF	6+	3+				S → M: M + 1	•	↕	↕	0	•
	STU				DF	5	2	FF	6	3				EF	5+	2+				U → M: M + 1	•	↕	↕	0	•
	STX				9F	5	2	BF	6	3				AF	5+	2+				X → M: M + 1	•	↕	↕	0	•
	STY				10 9F	6	3	10 BF	7	4				10 AF	6+	3+				Y → M: M + 1	•	↕	↕	0	•
SUB	SUBA				90	4	2	B0	5	3	80	2	2	A0	4+	2+				A − M → A	↕	↕	↕	↕	↕
	SUBB				D0	4	2	F0	5	3	C0	2	2	E0	4+	2+				B − M → B	↕	↕	↕	↕	↕
	SUBD				93	6	2	B3	7	3	83	4	3	A3	6+	2+				D − M: M + 1 → D	•	↕	↕	↕	↕
SWI	SWI[6]	3F	19	1																Software Interrupt 1	•	•	•	•	•
	SWI2[6]	10 3F	20	2																Software Interrupt 2	•	•	•	•	•
	SWI3[6]	11 3F	20	2																Software Interrupt 3	•	•	•	•	•
SYNC		13	≥2	1																Synchronize to Interrupt	•	•	•	•	•
TFR	R1, R2	1F	7	2																R1 → R2[3]	•	•	•	•	•
TST	TSTA	4D	2	1																Test A	•	↕	↕	0	•
	TSTB	5D	2	1																Test B	•	↕	↕	0	•
	TST				0D	6	2	7D	7	3				6D	6+	2+				Test M	•	↕	↕	0	•



INDEXED ADDRESSING MODES

TYPE	FORMS	NON INDIRECT Assembler Form	NON INDIRECT Post-Byte OP Code	NON INDIRECT + ~	NON INDIRECT + #	INDIRECT Assembler Form	INDIRECT Post-Byte OP Code	INDIRECT + ~	INDIRECT + #
CONSTANT OFFSET FROM R	NO OFFSET	, R	1RR00100	0	0	[, R]	1RR10100	3	0
	5 BIT OFFSET	n, R	0RRnnnnn	1	0	defaults to 8-bit			
	8 BIT OFFSET	n, R	1RR01000	1	1	[n, R]	1RR11000	4	1
	16 BIT OFFSET	n, R	1RR01001	4	2	[n, R]	1RR11001	7	2
ACCUMULATOR OFFSET FROM R	A—REGISTER OFFSET	A, R	1RR00110	1	0	[A, R]	1RR10110	4	0
	B—REGISTER OFFSET	B, R	1RR00101	1	0	[B, R]	1RR10101	4	0
	D—REGISTER OFFSET	D, R	1RR01011	4	0	[D, R]	1RR11011	7	0
AUTO INCREMENT/DECREMENT R	INCREMENT BY 1	, R+	1RR00000	2	0	not allowed			
	INCREMENT BY 2	, R+ +	1RR00001	3	0	[, R+ +]	1RR10001	6	0
	DECREMENT BY 1	, -R	1RR00010	2	0	not allowed			
	DECREMENT BY 2	, - - R	1RR00011	3	0	[, - - R]	1RR10011	6	0
CONSTANT OFFSET FROM PC	8 BIT OFFSET	n, PCR	1XX01100	1	1	[n, PCR]	1XX11100	4	1
	16 BIT OFFSET	n, PCR	1XX01101	5	2	[n, PCR]	1XX11101	8	2
EXTENDED INDIRECT	16 BIT ADDRESS	—	—	-	-	[n]	10011111	5	2

R = X, Y, U, or S — RR: 00 = X, 10 = U, 01 = Y, 11 = S
X = DON'T CARE

NOTES:

1. This column gives a base cycle and byte count. To obtain total count, add the values obtained from the INDEXED ADDRESSING MODES table.
2. R1 and R2 may be any pair of 8 bit or any pair of 16 bit registers.
   The 8 bit registers are: A, B, CC, DP
   The 16 bit registers are: X, Y, U, S, D, PC
3. EA is the effective address.
4. The PSH and PUL instructions require 5 cycles plus 1 cycle for each byte pushed or pulled.
5. 5(6) means: 5 cycles if branch not taken, 6 cycles if taken.
6. SWI sets I and F bits. SWI2 and SWI3 do not affect I and F.
7. Conditions Codes set as a direct result of the instruction.
8. Value of half-carry flag is undefined.
9. Special Case — Carry set if b7 is SET.

LEGEND:

OP	Operation Code (Hexadecimal)	Z	Zero (byte)
~	Number of MPU Cycles	V	Overflow, 2's complement
#	Number of Program Bytes	C	Carry from bit 7
+	Arithmetic Plus	I	Test and set if true, cleared otherwise
-	Arithmetic Minus	•	Not Affected
•	Multiply	CC	Condition Code Register
$\overline{M}$	Complement of M	:	Concatenation
→	Transfer Into	V	Logical or
H	Half-carry (from bit 3)	∧	Logical and
N	Negative (sign bit)	⊻	Logical Exclusive or

Simple Conditional Branches

Condition	Instruction	Complement
Equal	BEQ	BNE
Minus	BMI	BPL
C-Bit Set	BCS	BCC
V-Bit Set	BVS	BVC

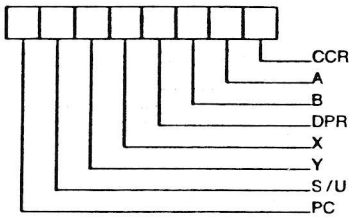
Signed Conditional Branches

Condition	Instruction	Complement
R > M	BGT	BLE
R ≥ M	BGE	BLT
R = M	BEQ	BNE
R ≤ M	BLE	BGT
R < M	BLT	BGE

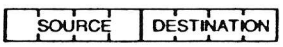
Unsigned Conditional Branches

Condition	Instruction	Complement
R > M	BHI	BLS
R ≥ M	BHS	BLO
R = M	BEQ	BNE
R ≤ M	BLS	BHI
R < M	BLO	BHS

PUSH/PULL POST BYTE



TRANSFER/EXCHANGE POST BYTE



REGISTER FIELD

0000 = D (A:B)	1000 = A
0001 = X	1001 = B
0010 = Y	1010 = CCR
0011 = U	1011 = DPR
0100 = S	
0101 = PC	

6809 STACKING ORDER

PULL ORDER
↓
CC
A
B
DP
X Hi
X Lo
Y Hi
Y Lo
U/S Hi
U/S Lo
PC Hi
PC Lo
↑
PUSH ORDER

INCREASING MEMORY
↓

6809 VECTORS

FFFE	Restart
FFFC	NMI
FFFA	SWI
FFF8	IRQ
FFF6	FIRQ
FFF4	SWI2
FFF2	SWI3
FFF0	Reserved

$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.