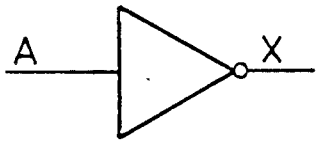
S/36

CAMBRIDGE UNIVERSITY
ENGINEERING DEPARTMENT

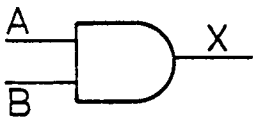
MICROPROCESSOR DATA

1986

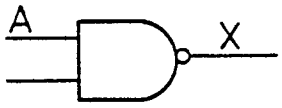
List of Contents



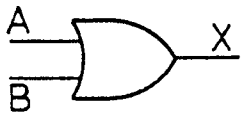
COMPLEMENT: $X = \overline{A}$



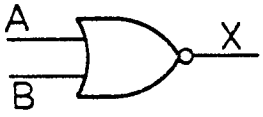
AND: $X = A.B$
$= A \wedge B$



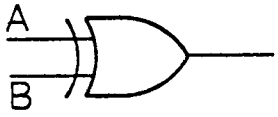
NAND: $X = \overline{A \cdot B}$



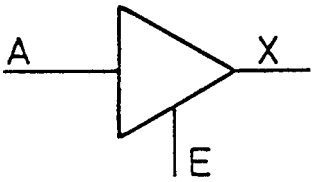
OR: $X = A + B$
$= A \vee B$



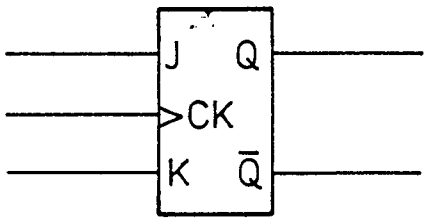
NOR: $X = \overline{A+B}$



EXCLUSIVE OR:
$X = A.\overline{B} + \overline{A}.B$
$= A \oplus B$
$= A \not\equiv B$

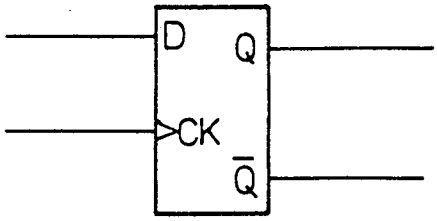


3-STATE BUFFER: E = 1 ; X = A
E = 0 ; OUTPUT DISABLED
(HIGH IMPEDANCE)



J – K FLIP-FLOP

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q}_n$



D FLIP-FLOP

D	Q_{n+1}
0	0
1	1

16-BIT (2-BYTE) HEXADECIMAL/DECIMAL CONVERSION

More significant BYTE				Less significant BYTE			
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,584	E	224	E	14
F	61,440	F	3,840	F	240	F	15

2s COMPLEMENT NUMBER REPRESENTATION
(8-bit word)

2s complement binary			Decimal	
0000	0000)		0)	
0000	0001)	Most significant	+ 1)	Positive
	)	bit equals 0	)	
0111	1111)		+ 127)	
¦				
1000	0000)		- 128)	
1000	0001)	Most significant	- 127)	Negative
¦	)	bit equals 1	)	
1111	1111)		- 1)	

Number range for n-bit word:

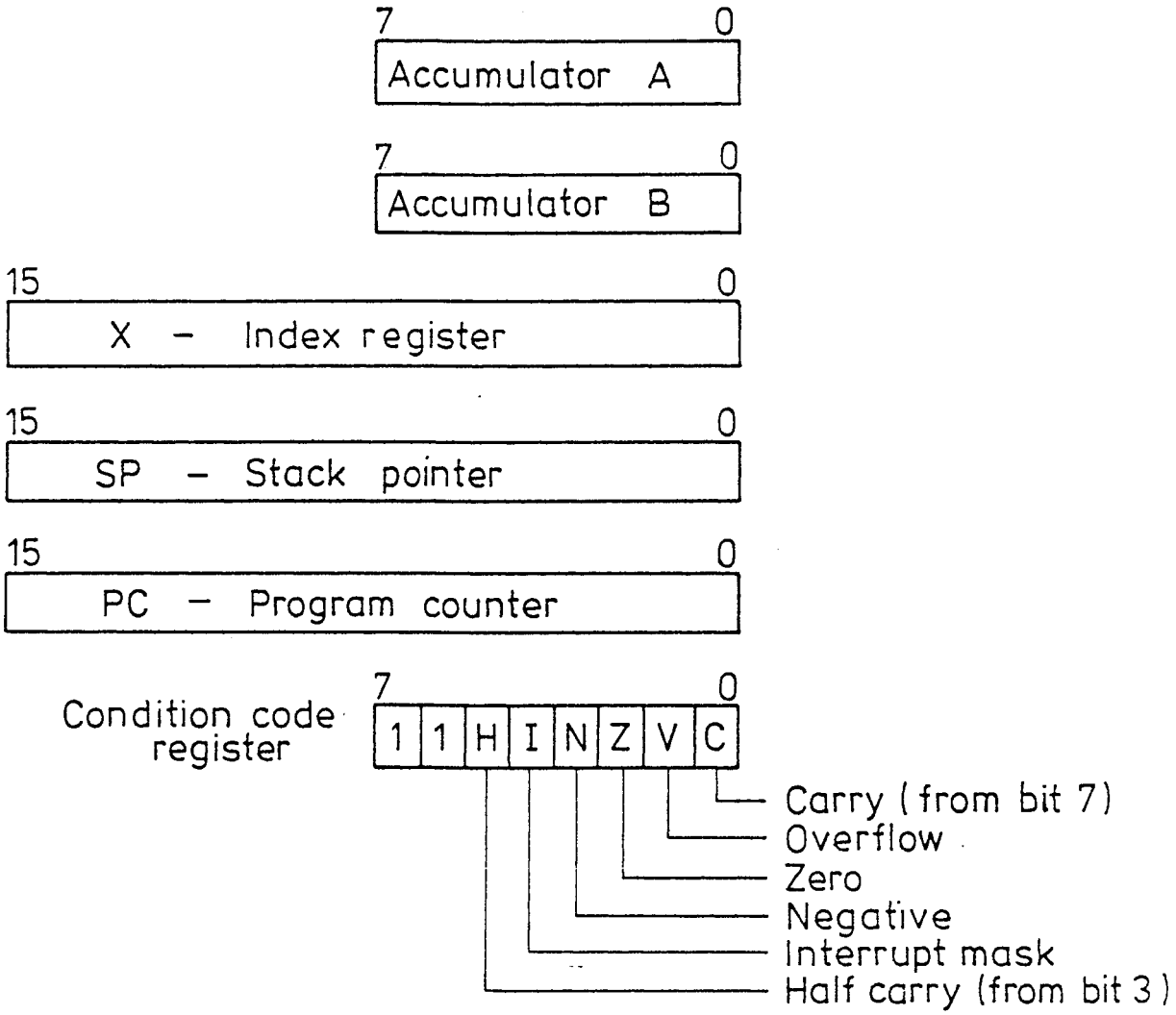
$$-2^{n-1} \leqslant x \leqslant 2^{n-1}-1$$

and the negative of a number is represented by its complement +1.

ASCII CHARACTER SET

DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII
0	00	^@	32	20		64	40	@	96	60	`
1	01	^A	33	21	!	65	41	A	97	61	a
2	02	^B	34	22	"	66	42	B	98	62	b
3	03	^C	35	23	#	67	43	C	99	63	c
4	04	^D	36	24	$	68	44	D	100	64	d
5	05	^E	37	25	%	69	45	E	101	65	e
6	06	^F	38	26	&	70	46	F	102	66	f
7	07	^G	39	27	'	71	47	G	103	67	g
8	08	^H	40	28	(	72	48	H	104	68	h
9	09	^I	41	29	)	73	49	I	105	69	i
10	0A	^J	42	2A	*	74	4A	J	106	6A	j
11	0B	^K	43	2B	+	75	4B	K	107	6B	k
12	0C	^L	44	2C	,	76	4C	L	108	6C	l
13	0D	^M	45	2D	–	77	4D	M	109	6D	m
14	0E	^N	46	2E	.	78	4E	N	110	6E	n
15	0F	^O	47	2F	/	79	4F	O	111	6F	o
16	10	^P	48	30	0	80	50	P	112	70	p
17	11	^Q	49	31	1	81	51	Q	113	71	q
18	12	^R	50	32	2	82	52	R	114	72	r
19	13	^S	51	33	3	83	53	S	115	73	s
20	14	^T	52	34	4	84	54	T	116	74	t
21	15	^U	53	35	5	85	55	U	117	75	u
22	16	^V	54	36	6	86	56	V	118	76	v
23	17	^W	55	37	7	87	57	W	119	77	w
24	18	^X	56	38	8	88	58	X	120	78	x
25	19	^Y	57	39	9	89	59	Y	121	79	y
26	1A	^Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	^[	59	3B	;	91	5B	[	123	7B	{
28	1C	^\	60	3C	<	92	5C	\	124	7C	|
29	1D	^]	61	3D	=	93	5D	]	125	7D	}
30	1E	^^	62	3E	>	94	5E	^	126	7E	~
31	1F	^_	63	3F	?	95	5F	_	127	7F	DEL

^@ means CTRL @



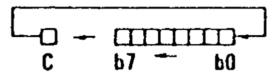
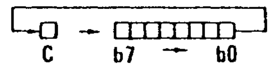
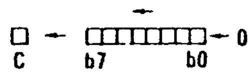
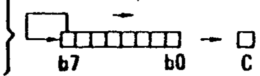
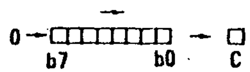
Permanent memory assignments for interrupt vectors

Contents	Address
RESET (low byte)	FFFF
RESET (high byte)	FFFE
NON-MASKABLE INTERRUPT (low byte)	FFFD
NON-MASKABLE INTERRUPT (high byte)	FFFC
SOFTWARE INTERRUPT (low byte)	FFFB
SOFTWARE INTERRUPT (high byte)	FFFA
INTERRUPT REQUEST (low byte)	FFF9
INTERRUPT REQUEST (high byte)	FFF8

MPU INSTRUCTIONS

Accumulator and Memory Instructions

OPERATIONS	MNEMONIC	IMMED OP	IMMED ~	IMMED #	DIRECT OP	DIRECT ~	DIRECT #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	IMPLIED OP	IMPLIED ~	IMPLIED #	BOOLEAN/ARITHMETIC OPERATION (All register labels refer to contents)	5 H	4 I	3 N	2 Z	1 V	0 C
Add	ADDA	8B	2	2	9B	3	2	AB	5	2	BB	4	3				A + M → A	↕	•	↕	↕	↕	↕
	ADDB	CB	2	2	DB	3	2	EB	5	2	FB	4	3				B + M → B	↕	•	↕	↕	↕	↕
Add Acmltrs	ABA													1B	2	1	A + B → A	↕	•	↕	↕	↕	↕
Add with Carry	ADCA	89	2	2	99	3	2	A9	5	2	B9	4	3				A + M + C → A	↕	•	↕	↕	↕	↕
	ADCB	C9	2	2	D9	3	2	E9	5	2	F9	4	3				B + M + C → B	↕	•	↕	↕	↕	↕
And	ANDA	84	2	2	94	3	2	A4	5	2	B4	4	3				A · M → A	•	•	↕	↕	R	•
	ANDB	C4	2	2	D4	3	2	E4	5	2	F4	4	3				B · M → B	•	•	↕	↕	R	•
Bit Test	BITA	85	2	2	95	3	2	A5	5	2	B5	4	3				A · M	•	•	↕	↕	R	•
	BITB	C5	2	2	D5	3	2	E5	5	2	F5	4	3				B · M	•	•	↕	↕	R	•
Clear	CLR							6F	7	2	7F	6	3				00 → M	•	•	R	S	R	R
	CLRA													4F	2	1	00 → A	•	•	R	S	R	R
	CLRB													5F	2	1	00 → B	•	•	R	S	R	R
Compare	CMPA	81	2	2	91	3	2	A1	5	2	B1	4	3				A − M	•	•	↕	↕	↕	↕
	CMPB	C1	2	2	D1	3	2	E1	5	2	F1	4	3				B − M	•	•	↕	↕	↕	↕
Compare Acmltrs	CBA													11	2	1	A − B	•	•	↕	↕	↕	↕
Complement, 1's	COM							63	7	2	73	6	3				$\overline{M} \rightarrow M$	•	•	↕	↕	R	S
	COMA													43	2	1	$\overline{A} \rightarrow A$	•	•	↕	↕	R	S
	COMB													53	2	1	$\overline{B} \rightarrow B$	•	•	↕	↕	R	S
Complement, 2's	NEG							60	7	2	70	6	3				00 − M → M	•	•	↕	↕	①	②
(Negate)	NEGA													40	2	1	00 − A → A	•	•	↕	↕	①	②
	NEGB													50	2	1	00 − B → B	•	•	↕	↕	①	②
Decimal Adjust, A	DAA													19	2	1	Converts Binary Add. of BCD Characters into BCD Format	•	•	↕	↕	↕	③
Decrement	DEC							6A	7	2	7A	6	3				M − 1 → M	•	•	↕	↕	4	•
	DECA													4A	2	1	A − 1 → A	•	•	↕	↕	4	•
	DECB													5A	2	1	B − 1 → B	•	•	↕	↕	4	•
Exclusive OR	EORA	88	2	2	98	3	2	A8	5	2	B8	4	3				A⊕M → A	•	•	↕	↕	R	•
	EORB	C8	2	2	D8	3	2	E8	5	2	F8	4	3				B⊕M → B	•	•	↕	↕	R	•
Increment	INC							6C	7	2	7C	6	3				M + 1 → M	•	•	↕	↕	⑤	•
	INCA													4C	2	1	A + 1 → A	•	•	↕	↕	⑤	•
	INCB													5C	2	1	B + 1 → B	•	•	↕	↕	⑤	•
Load Acmltr	LDAA	86	2	2	96	3	2	A6	5	2	B6	4	3				M → A	•	•	↕	↕	R	•
	LDAB	C6	2	2	D6	3	2	E6	5	2	F6	4	3				M → B	•	•	↕	↕	R	•
Or, Inclusive	ORAA	8A	2	2	9A	3	2	AA	5	2	BA	4	3				A + M → A	•	•	↕	↕	R	•
	ORAB	CA	2	2	DA	3	2	EA	5	2	FA	4	3				B + M → B	•	•	↕	↕	R	•
Push Data	PSHA													36	4	1	$A \rightarrow M_{SP}$, SP − 1 → SP	•	•	•	•	•	•
	PSHB													37	4	1	$B \rightarrow M_{SP}$, SP − 1 → SP	•	•	•	•	•	•
Pull Data	PULA													32	4	1	SP + 1 → SP, $M_{SP} \rightarrow A$	•	•	•	•	•	•
	PULB													33	4	1	SP + 1 → SP, $M_{SP} \rightarrow B$	•	•	•	•	•	•

Push Data	PSHA														36	4	1	A → M_{SP}, SP − 1 → SP	●	●	●	●	●	●
	PSHB														37	4	1	B → M_{SP}, SP − 1 → SP	●	●	●	●	●	●
Pull Data	PULA														32	4	1	SP + 1 → SP, M_{SP} → A	●	●	●	●	●	●
	PULB														33	4	1	SP + 1 → SP, M_{SP} → B	●	●	●	●	●	●
Rotate Left	ROL							69	7	2	79	6	3					M	●	●	↕	↕	(6)	↕
	ROLA														49	2	1	A 	●	●	↕	↕	(6)	↕
	ROLB														59	2	1	B	●	●	↕	↕	(6)	↕
Rotate Right	ROR							66	7	2	76	6	3					M	●	●	↕	↕	(6)	↕
	RORA														46	2	1	A 	●	●	↕	↕	(6)	↕
	RORB														56	2	1	B	●	●	↕	↕	(6)	↕
Shift Left, Arithmetic	ASL							68	7	2	78	6	3					M	●	●	↕	↕	(6)	↕
	ASLA														48	2	1	A 	●	●	↕	↕	(6)	↕
	ASLB														58	2	1	B	●	●	↕	↕	(6)	↕
Shift Right, Arithmetic	ASR							67	7	2	77	6	3					M	●	●	↕	↕	(6)	↕
	ASRA														47	2	1	A 	●	●	↕	↕	(6)	↕
	ASRB														57	2	1	B	●	●	↕	↕	(6)	↕
Shift Right, Logic	LSR							64	7	2	74	6	3					M	●	●	R	↕	(6)	↕
	LSRA														44	2	1	A 	●	●	R	↕	(6)	↕
	LSRB														54	2	1	B	●	●	R	↕	(6)	↕
Store Acmltr.	STAA				97	4	2	A7	6	2	B7	5	3					A → M	●	●	↕	↕	R	●
	STAB				D7	4	2	E7	6	2	F7	5	3					B → M	●	●	↕	↕	R	●
Subtract	SUBA	80	2	2	90	3	2	A0	5	2	B0	4	3					A − M → A	●	●	↕	↕	↕	↕
	SUBB	C0	2	2	D0	3	2	E0	5	2	F0	4	3					B − M → B	●	●	↕	↕	↕	↕
Subtract Acmltrs.	SBA														10	2	1	A − B → A	●	●	↕	↕	↕	↕
Subtr. with Carry	SBCA	82	2	2	92	3	2	A2	5	2	B2	4	3					A − M − C → A	●	●	↕	↕	↕	↕
	SBCB	C2	2	2	D2	3	2	E2	5	2	F2	4	3					B − M − C → B	●	●	↕	↕	↕	↕
Transfer Acmltrs	TAB														16	2	1	A → B	●	●	↕	↕	R	●
	TBA														17	2	1	B → A	●	●	↕	↕	R	●
Test, Zero or Minus	TST							6D	7	2	7D	6	3					M − 00	●	●	↕	↕	R	R
	TSTA														4D	2	1	A − 00	●	●	↕	↕	R	R
	TSTB														5D	2	1	B − 00	●	●	↕	↕	R	R
																			H	I	N	Z	V	C

LEGEND:

OP	Operation Code (Hexadecimal);	+	Boolean Inclusive OR;
~	Number of MPU Cycles;	⊕	Boolean Exclusive OR;
#	Number of Program Bytes;	$\overline{M}$	Complement of M;
+	Arithmetic Plus;	→	Transfer Into;
−	Arithmetic Minus;	0	Bit = Zero;
·	Boolean AND;	00	Byte = Zero;
M_{SP}	Contents of memory location pointed to be Stack Pointer;		

Note – Accumulator addressing mode instructions are included in the column for IMPLIED addressing

CONDITION CODE SYMBOLS:

H	Half-carry from bit 3;
I	Interrupt mask
N	Negative (sign bit)
Z	Zero (byte)
V	Overflow, 2's complement
C	Carry from bit 7
R	Reset Always
S	Set Always
↕	Test and set if true, cleared otherwise
●	Not Affected

Index Register and Stack Manipulation Instructions

POINTER OPERATIONS	MNEMONIC	IMMED OP	IMMED ~	IMMED #	DIRECT OP	DIRECT ~	DIRECT #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	IMPLIED OP	IMPLIED ~	IMPLIED #	BOOLEAN/ARITHMETIC OPERATION	5 H	4 I	3 N	2 Z	1 V	0 C
Compare Index Reg	CPX	8C	3	3	9C	4	2	AC	6	2	BC	5	3				$X_H - M, X_L - (M+1)$	•	•	⑦	↕	⑦	•
Decrement Index Reg	DEX													09	4	1	$X - 1 \rightarrow X$	•	•	•	↕	•	•
Decrement Stack Pntr	DES													34	4	1	$SP - 1 \rightarrow SP$	•	•	•	•	•	•
Increment Index Reg	INX													08	4	1	$X + 1 \rightarrow X$	•	•	•	↕	•	•
Increment Stack Pntr	INS													31	4	1	$SP + 1 \rightarrow SP$	•	•	•	•	•	•
Load Index Reg	LOX	CE	3	3	DE	4	2	EE	6	2	FE	5	3				$M \rightarrow X_H, (M+1) \rightarrow X_L$	•	•	⑨	↕	R	•
Load Stack Pntr	LDS	8E	3	3	9E	4	2	AE	6	2.	BE	5	3				$M \rightarrow SP_H, (M+1) \rightarrow SP_L$	•	•	⑨	↕	R	•
Store Index Reg	STX				DF	5	2	EF	7	2	FF	6	3				$X_H \rightarrow M, X_L \rightarrow (M+1)$	•	•	⑨	↕	R	•
Store Stack Pntr	STS				9F	5	2	AF	7	2	BF	6	3				$SP_H \rightarrow M, SP_L \rightarrow (M+1)$	•	•	⑨	↕	R	•
Indx Reg → Stack Pntr	TXS													35	4	1	$X - 1 \rightarrow SP$	•	•	•	•	•	•
Stack Pntr → Indx Reg	TSX													30	4	1	$SP + 1 \rightarrow X$	•	•	•	•	•	•

Jump and Branch Instructions

OPERATIONS	MNEMONIC	RELATIVE OP	RELATIVE ~	RELATIVE #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	IMPLIED OP	IMPLIED ~	IMPLIED #	BRANCH TEST	5 H	4 I	3 N	2 Z	1 V	0 C
Branch Always	BRA	20	4	2										None	•	•	•	•	•	•
Branch If Carry Clear	BCC	24	4	2										C = 0	•	•	•	•	•	•
Branch If Carry Set	BCS	25	4	2										C = 1	•	•	•	•	•	•
Branch If = Zero	BEQ	27	4	2										Z = 1	•	•	•	•	•	•
Branch If ≥ Zero	BGE	2C	4	2										$N \oplus V = 0$	•	•	•	•	•	•
Branch If > Zero	BGT	2E	4	2										$Z + (N \oplus V) = 0$	•	•	•	•	•	•
Branch If Higher	BHI	22	4	2										C + Z = 0	•	•	•	•	•	•
Branch If ≤ Zero	BLE	2F	4	2										$Z + (N \oplus V) = 1$	•	•	•	•	•	•
Branch If Lower Or Same	BLS	23	4	2										C + Z = 1	•	•	•	•	•	•
Branch If < Zero	BLT	2D	4	2										$N \oplus V = 1$	•	•	•	•	•	•
Branch If Minus	BMI	2B	4	2										N = 1	•	•	•	•	•	•
Branch If Not Equal Zero	BNE	26	4	2										Z = 0	•	•	•	•	•	•
Branch If Overflow Clear	BVC	28	4	2										V = 0	•	•	•	•	•	•
Branch If Overflow Set	BVS	29	4	2										V = 1	•	•	•	•	•	•
Branch If Plus	BPL	2A	4	2										N = 0	•	•	•	•	•	•
Branch To Subroutine	BSR	8D	8	2											•	•	•	•	•	•
Jump	JMP				6E	4	2	7E	3	3				See Special Operations	•	•	•	•	•	•
Jump To Subroutine	JSR				AD	8	2	BD	9	3					•	•	•	•	•	•
No Operation	NOP										02	2	1	Advances Prog. Cntr. Only	•	•	•	•	•	•
Return From Interrupt	RTI										3B	10	1		⑩					
Return From Subroutine	RTS										39	5	1		•	•	•	•	•	•
Software Interrupt	SWI										3F	12	1	See Special Operations	•	•	•	•	•	•
Wait for Interrupt	WAI										3E	9	1		•	⑪	•	•	•	•

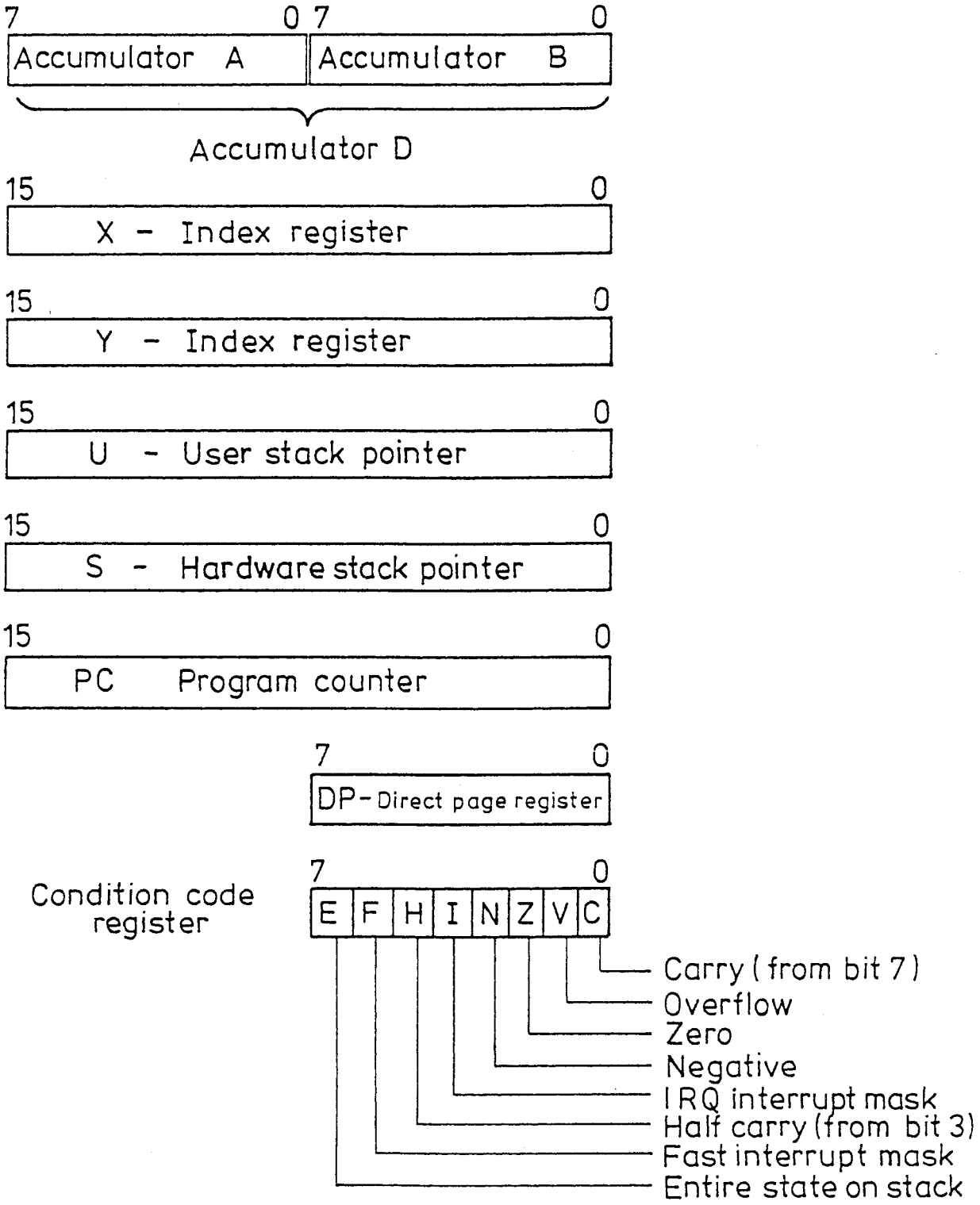
Condition Code Register Manipulation Instructions

OPERATIONS	MNEMONIC	IMPLIED OP	~	#	BOOLEAN OPERATION	COND. CODE REG. 5 H	4 I	3 N	2 Z	1 V	0 C
Clear Carry	CLC	0C	2	1	0 → C	•	•	•	•	•	R
Clear Interrupt Mask	CLI	0E	2	1	0 → I	•	R	•	•	•	•
Clear Overflow	CLV	0A	2	1	0 → V	•	•	•	•	R	•
Set Carry	SEC	0D	2	1	1 → C	•	•	•	•	•	S
Set Interrupt Mask	SEI	0F	2	1	1 → I	•	S	•	•	•	•
Set Overflow	SEV	0B	2	1	1 → V	•	•	•	•	S	•
Acmltr A → CCR	TAP	06	2	1	A → CCR	(12)					
CCR → Acmltr A	TPA	07	2	1	CCR → A	•	•	•	•	•	•

CONDITION CODE REGISTER NOTES:

(Bit set if test is true and cleared otherwise)

1 (Bit V) Test: Result = 10000000?
2 (Bit C) Test: Result = 00000000?
3 (Bit C) Test: Decimal value of most significant BCD Character greater than nine? (Not cleared if previously set.)
4 (Bit V) Test: Operand = 10000000 prior to execution?
5 (Bit V) Test: Operand = 01111111 prior to execution?
6 (Bit V) Test: Set equal to result of N⊕C after shift has occurred.
7 (Bit N) Test: Sign bit of most significant (MS) byte = 1?
8 (Bit V) Test: 2's complement overflow from subtraction of MS bytes?
9 (Bit N) Test: Result less than zero? (Bit 15 = 1)
10 (All) Load Condition Code Register from Stack. (See Special Operations)
11 (Bit I) Set when interrupt occurs. If previously set, a Non-Maskable Interrupt is required to exit the wait state.
12 (All) Set according to the contents of Accumulator A.


7 0 7 0
Accumulator A
Accumulator B
Accumulator D
15 0
X - Index register
15 0
Y - Index register
15 0
U - User stack pointer
15 0
S - Hardware stack pointer
15 0
PC Program counter
7 0
DP - Direct page register
Condition code register
7 0
E F H I N Z V C
Carry (from bit 7)
Overflow
Zero
Negative
IRQ interrupt mask
Half carry (from bit 3)
Fast interrupt mask
Entire state on stack

INSTRUCTION/ FORMS	DESCRIPTION	5 H	3 N	2 Z	1 V	0 C	Number of clock cycle ADDRESSING MODES: Inherent	Direct	Extended	Immediate	Indexed	Relative
ABX	B + X → X (UNSIGNED)	●	●	●	●	●	3					
ADC ADCA ADCB	A + M + C → A B + M + C → B	↕ ↕	↕ ↕	↕ ↕	↕ ↕	↕ ↕		4 4	5 5	2 2	4+ 4+	
ADD ADDA ADDB ADDD	A + M → A B + M → B D + M:M + 1 → D	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕		4 4 6	5 5 7	2 2 4	4+ 4+ 6+	
AND ANDA ANDB ANDCC	A∧M → A B∧M → B CC ∧ IMM → CC	● ●	↕ ↕	↕ ↕	0 0	● ● 1		4 4	5 5	2 2 3	4+ 4+	
ASL ASLA ASLB ASL	A, B, M } C b₁ b₀	8 8 8	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕	2 2	6	7		6+	
ASR ASRA ASR ASR	A, B, M } b₇ b₀ C	8 8 8	↕ ↕ ↕	↕ ↕ ↕	● ● ●	↕ ↕ ↕	2 2	6	7		6+	
B C C BCC LBCC	Branch C = 0 Long Branch C = 0	● ●	● ●	● ●	● ●	● ●						3 5(6)
BCS BCS LBCS	Branch C = 1 Long Branch C = 1	● ●	● ●	● ●	● ●	● ●						3 5(6)
BEQ BEQ LBEQ	Branch Z = 0 Long Branch Z = 0	● ●	● ●	● ●	● ●	● ●						3 5(6)
BGE BGE LBGE	Branch ≥ Zero Long Branch ≥ Zero	● ●	● ●	● ●	● ●	● ●						3 5(6)
BGT BGT LBGT	Branch > Zero Long Branch > Zero	● ●	● ●	● ●	● ●	● ●						3 5(6)
BHI BHI LBHI	Branch Higher Long Branch Higher	● ●	● ●	● ●	● ●	● ●						3 5(6)
BHS BHS LBHS	Branch Higher or Same Long Branch Higher or Same	● ●	● ●	● ●	● ●	● ●						3 5(6)
BIT BITA BITB	Bit Test A (M ∧ A) Bit Test B (M ∧ B)	● ●	↕ ↕	↕ ↕	0 0	● ●		4 4	5 5	2 2	4+ 4+	-
BLE BLE LBLE	Branch ≤ Zero Long Branch ≤ Zero	● ●	● ●	● ●	● ●	● ●						3 5(6)

INSTRUCTION/ FORMS		DESCRIPTION	5	3	2	1	0	Number of clock cycles ADDRESSING MODES					
			H	N	Z	V	C	Inherent	Direct	Extended	Immediate	Indexed	Relative
BLO	BLO LBLO	Branch Lower Long Branch Lower	● ●	● ●	● ●	● ●	● ●						3 5(4)
BLS	BLS LBLS	Branch Lower or Same Long Branch Lower or Same	● ●	● ●	● ●	● ●	● ●						3 5(4)
BLT	BLT LBLT	Branch < Zero Long Branch < Zero	● ●	● ●	● ●	● ●	● ●						3 5(4)
BMI	BMI LBMI	Branch Minus Long Branch Minus	● ●	● ●	● ●	● ●	● ●						3 5(4)
BNE	BNE LBNE	Branch Z ≠ 0 Long Branch Z ≠ 0	● ●	● ●	● ●	● ●	● ●						3 5(4)
BPL	BPL LBPL	Branch Plus Long Branch Plus	● ●	● ●	● ●	● ●	● ●						3 5(4)
BRA	BRA LBRA	Branch Always Long Branch Always	● ●	● ●	● ●	● ●	● ●						3 5
BRN	BRN LBRN	Branch Never Long Branch Never	● ●	● ●	● ●	● ●	● ●						3 5
BSR	BRS LBSR	Branch to Subroutine Long Branch to Subroutine	● ●	● ●	● ●	● ●	● ●						7 9
BVC	BVC LBVC	Branch V = 0 Long Branch V = 0	● ●	● ●	● ●	● ●	● ●						3 5(4)
BVS	BVS LBVS	Branch V = 1 Long Branch V = 1	● ●	● ●	● ●	● ●	● ●						3 5(4)
CLR	CLRA CLRB CLR	0 → A 0 → B 0 → M	● ● ●	0 0 0	1 1 1	0 0 0	0 0 0	2 2	 6	 7		 6+	
CMP	CMPA CMPB CMPD CMPS CMPU CMPX CMPY	Compare M from A Compare M from B Compare M:M + 1 from D Compare M:M + 1 from S Compare M:M + 1 from U Compare M:M + 1 from X Compare M:M + 1 from Y	8 8 ● ● ● ● ●	↕ ↕ ↕ ↕ ↕ ↕ ↕	↕ ↕ ↕ ↕ ↕ ↕ ↕	↕ ↕ ↕ ↕ ↕ ↕ ↕	↕ ↕ ↕ ↕ ↕ ↕ ↕		4 4 7 7 7 6 7	5 5 8 8 8 7 8	2 2 5 5 5 4 5	4+ 4+ 7+ 7+ 7+ 6+ 7+	

INSTRUCTION/ FORMS		DESCRIPTION	5	3	2	1	0	Number of clock cycles ADDRESSING MODES					
			H	N	Z	V	C	Inherent	Direct	Extended	Immediate	Indexed	Relative
COM	COMA	$\overline{A} \rightarrow A$	●	↕	↕	0	1	2					
	COMB	$\overline{B} \rightarrow B$	●	↕	↕	0	1	2					
	COM	$\overline{M} \rightarrow M$	●	↕	↕	0	1		6	7		6+	
CWAI		CC ∧ IMM → CC; Wait for Interrupt					1	20					
DAA		Decimal Adjust A	●	↕	↕	0	↕	2					
DEC	DECA	A − 1 → A	●	↕	↕	↕	●	2					
	DECB	B − 1 → B	●	↕	↕	↕	●	2					
	DEC	M − 1 → M	●	↕	↕	↕	●		6	7		6+	
EOR	EORA	A ⊻ M → A	●	↕	↕	0	●		4	5	2	4+	
	EORB	B ⊻ M → B	●	↕	↕	0	●		4	5	2	4+	
EXG	R1, R2	R1 ↔ R2[2]	●	●	●	●	●	7					
INC	INCA	A + 1 → A	●	↕	↕	↕	●	2					
	INCB	B + 1 → B	●	↕	↕	↕	●	2					
	INC	M + 1 → M	●	↕	↕	↕	●		6	7		6+	
JMP		EA[3] → PC	●	●	●	●	●		3	4		3+	
JSR		Jump to Subroutine	●	●	●	●	●		7	8		7+	
LD	LDA	M → A	●	↕	↕	0	●		4	5	2	4+	
	LDB	M → B	●	↕	↕	0	●		4	5	2	4+	
	LDD	M:M + 1 → D	●	↕	↕	0	●		5	6	3	5+	
	LDS	M:M + 1 → S	●	↕	↕	0	●		6	7	4	6+	
	LDU	M:M + 1 → U	●	↕	↕	0	●		5	6	3	5+	
	LDX	M:M + 1 → X	●	↕	↕	0	●		5	6	3	5+	
	LDY	M:M + 1 → Y	●	↕	↕	0	●		6	7	4	6+	
LEA	LEAS	EA[3] → S	●	●	●	●	●					4+	
	LEAU	EA[3] → U	●	●	●	●	●					4+	
	LEAX	EA[3] → X	●	●	↕	●	●					4+	
	LEAY	EA[3] → Y	●	●	↕	●	●					4+	
LSL	LSLA	A, B, M: C ← b7 ← b6 ← b5 ← b4 ← b3 ← b2 ← b1 ← b0 ← 0	●	↕	↕	↕	↕	2					
	LSLB		●	↕	↕	↕	↕	2					
	LSL		●	↕	↕	↕	↕		6	7		6+	
LSR	LSRA	A, B, M: 0 → b7 → b6 → b5 → b4 → b3 → b2 → b1 → b0 → C	●	0	↕	●	↕	2					
	LSRB		●	0	↕	●	↕	2					
	LSR		●	0	↕	●	↕		6	7		6+	
MUL		A x B → D (Unsigned)	●	●	↕	●	9	11					
NEG	NEGA	$\overline{A} + 1 \rightarrow A$	8	↕	↕	↕	↕	2					
	NEGB	$\overline{B} + 1 \rightarrow B$	8	↕	↕	↕	↕	2					
	NEG	$\overline{M} + 1 \rightarrow M$	8	↕	↕	↕	↕		6	7		6+	
NOP		No Operation	●	●	●	●	●	2					
OR	ORA	A ∨ M → A	●	↕	↕	0	●		4	5	2	4+	
	ORB	B ∨ M → B	●	↕	↕	0	●		4	5	2	4+	
	ORCC	CC ∨ IMM → CC					7				3		
PSH	PSHS	Push Registers on S Stack	●	●	●	●	●	5+[4]					
	PSHU	Push Registers on U Stack	●	●	●	●	●	5+[4]					

NOTES:
1. Given in the table are the base cycles. To determine the total cycles add the values from the '6809' indexing 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. SW1 sets I & F bits. SW12 and SW13 do not affect I & 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.

INSTRUCTION/ FORMS		DESCRIPTION	5 H	3 N	2 Z	1 V	0 C	Number of clock cycles ADDRESSING MODES: Inherent	Direct	Extended	Immediate	Indexed	Relative
PUL	PULS	Pull Registers from S Stack	●	●	●	●	●	5+					
	PULU	Pull Registers from U Stack	●	●	●	●	●	5+					
ROL	ROLA	A ⟲ c b7 ← b0	●	↕	↕	↕	↕	2					
	ROLB	B ⟲ c b7 ← b0	●	↕	↕	↕	↕	2					
	ROL	M ⟲ c b7 ← b0	●	↕	↕	↕	↕		6	7		6+	
ROR	RORA	A ⟲ c b7 → b0	●	↕	↕	●	↕	2					
	RORB	B ⟲ c b7 → b0	●	↕	↕	●	↕	2					
	ROR	M ⟲ c b7 → b0	●	↕	↕	●	↕		6	7		6+	
RTI		Return From Interrupt					7	6/15					
RTS		Return From Subroutine	●	●	●	●	●	5					
SBC	SBCA	A - M - C → A	8	↕	↕	↕	↕		4	5	2	4+	
	SBCB	B - M - C → B	8	↕	↕	↕	↕		4	5	2	4+	
SEX		Sign Extend B into A	●	↕	↕	0	●	2					
ST	STA	A → M	●	↕	↕	0	●		4	5		4+	
	STB	B → M	●	↕	↕	0	●		4	5		4+	
	STD	D → M:M + 1	●	↕	↕	0	●		5	6		5+	
	STS	S → M:M + 1	●	↕	↕	0	●		6	7		6+	
	STU	U → M:M + 1	●	↕	↕	0	●		5	6		5+	
	STX	X → M:M + 1	●	↕	↕	0	●		5	6		5+	
	STY	Y → M:M + 1	●	↕	↕	0	●		6	7		6+	
SUB	SUBA	A - M → A	8	↕	↕	↕	↕		4	5	2	4+	
	SUBB	B - M → B	8	↕	↕	↕	↕		4	5	2	4+	
	SUBD	D - M:M + 1 → D	●	↕	↕	↕	↕		6	7	4	6+	
SWI	SWI[6]	Software Interrupt 1	●	●	●	●	●	19					
	SW12[6]	Software Interrupt 2	●	●	●	●	●	20					
	SW13[6]	Software Interrupt 3	●	●	●	●	●	20					
SYNC		Synchronize to Interrupt	●	●	●	●	●	≥2					
TFR	R1, R2	R1 → R2[2]	●	●	●	●	●	7					
TST	TSTA	Test A	●	↕	↕	0	●	2					
	TSTB	Test B	●	↕	↕	0	●	2					
	TST	Test M	●	↕	↕	0	●		6	7			

6809 STACKING ORDER

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

6809 VECTORS

FFFE	Restart
FFFC	NMI
FFFA	SWI
FFF8	IRQ
FFF6	FIRQ
FFF4	SW12
FFF2	SW13
FFF0	Reserved

Simple Conditional Branches

Condition	*Complement*
BEQ	BNE
BMI	BPL
BCS	BCC
BVS	BVC

Signed Conditional Branches

Condition	*Complement*
BGT	BLE
BGE	BLT
BEQ	BNE
BLE	BGT
BLT	BGE

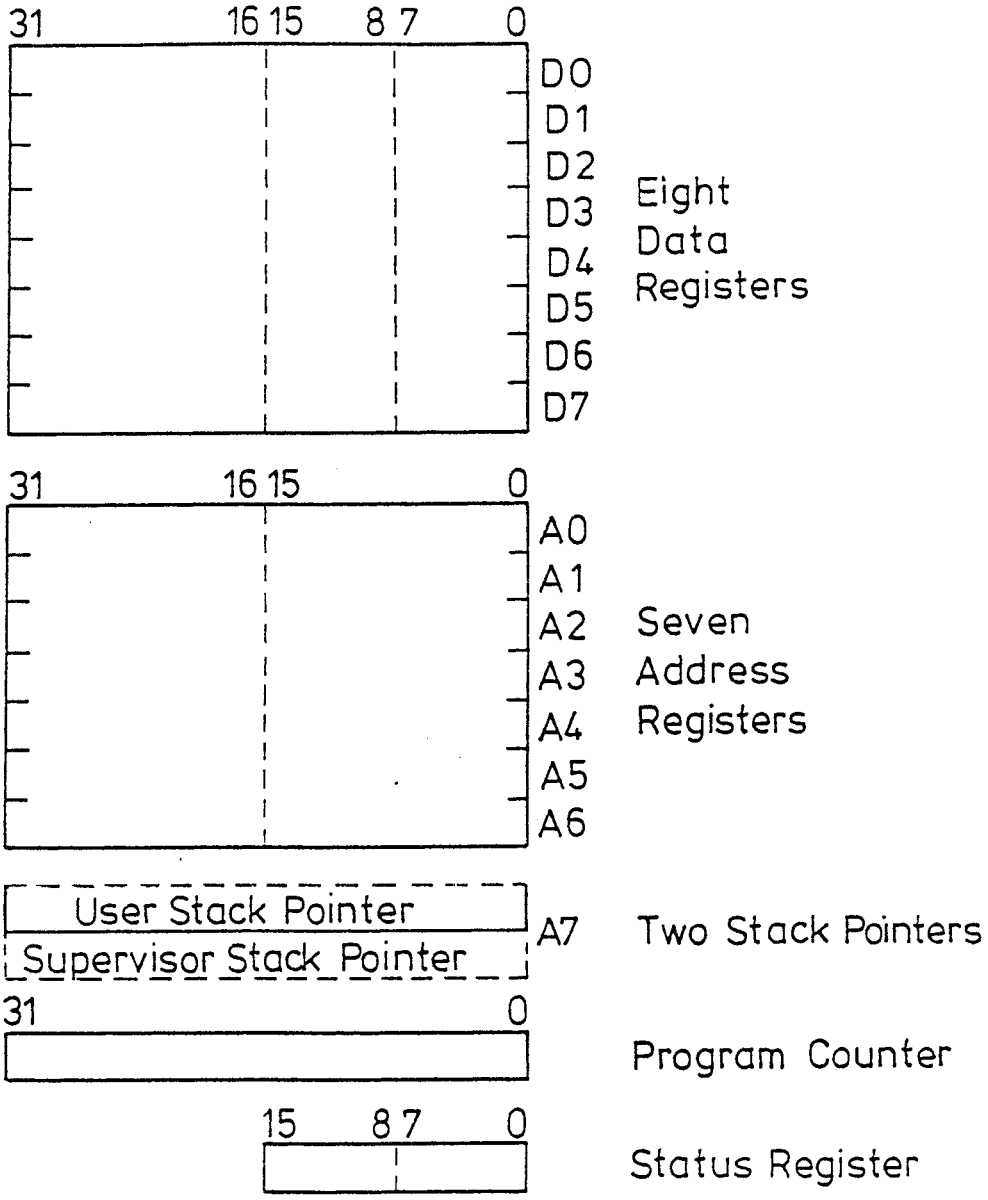
Unsigned Conditional Branches

Condition	*Complement*
BHI	BLS
BHS	BLO
BEQ	BNE
BLS	BHI
BLO	BHS

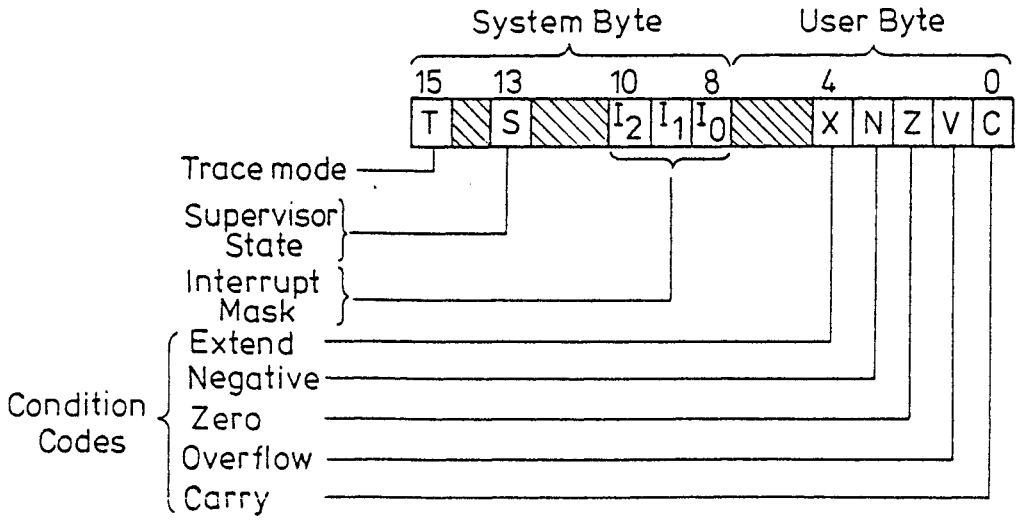
INDEXED ADDRESSING MODES

		NON INDIRECT		INDIRECT	
TYPE	FORMS	Assembler Form	+ cycles	Assembler Form	+ cycles
CONSTANT OFFSET FROM R	NO OFFSET	,R	0	[,R]	3
	5 BIT OFFSET	n,R	1	defaults to 8-bit	
	8 BIT OFFSET	n,R	1	[n,R]	4
	16 BIT OFFSET	n,R	4	[n,R]	7
ACCUMULATOR OFFSET FROM R	A-REGISTER OFFSET	A,R	1	[A,R]	4
	B-REGISTER OFFSET	B,R	1	[B,R]	4
	D-REGISTER OFFSET	D,R	4	[D,R]	7
AUTO INCREMENT/DECREMENT R	INCREMENT BY 1	,R+	2	not allowed	
	INCREMENT BY 2	,R++	3	[,R++]	6
	DECREMENT BY 1	,-R	2	not allowed	
	DECREMENT BY 2	,--R	3	[,--R]	6
CONSTANT OFFSET FROM PC	8 BIT OFFSET	n,PCR	1	[n,PCR]	4
	16 BIT OFFSET	n,PCR	5	[n,PCR]	8
EXTENDED INDIRECT	16 BIT ADDRESS	-	-	[n]	5

R = X,Y,U, or S



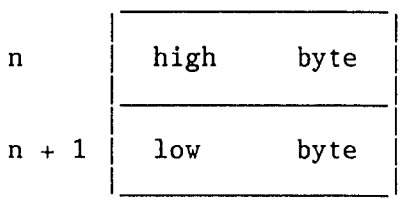
STATUS REGISTER



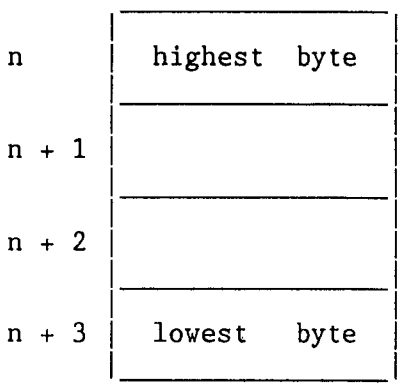
Data organisation in memory

An address specifies a particular byte in the memory space of the 68000. Words and long words may only be accessed at even addresses, this address being that of the high order byte of the word. Thus where n is an even address the byte order of words and long words is as follows:

word:



long word:



Instruction set

In the following table the complete set of instruction mnemonics known to the assembler is listed. Where given, one of the extension letters shown in braces {} should be used; the variants shown in brackets [] are optional.

Mnemonic	Description
ABCD	Add Decimal with Extend
ADD[QX].{BWL}	Add
AND.{BWL}	Logical And
ASL.{BWL}	Arithmetic Shift Left
ASR.{BWL}	Arithmetic Shift Right
Bcc	Branch Conditionally
BCHG	Bit Test and Change
BCLR	Bit Test and Clear
BRA	Branch Always
BSET	Bit Test and Set
BSR	Branch to Subroutine
BTST	Bit Test
CHK	Check Register Against Bounds

CLR.{BWL}	Clear Operand
CMP[M].{BWL}	Compare
DBcc	Test Cond.,Decrement and Branch
DIVS	Signed Divide
DIVU	Unsigned Divide
EOR.{BWL}	Exclusive Or
EXG	Exchange Registers
EXT.{WL}	Sign Extend
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link Stack
LSL.{BWL}	Logical Shift Left
LSR.{BWL}	Logical Shift Right
MOVE[Q].{BWL}	Move
MOVEM.{WL}	Move Multiple Registers
MOVEP.{WL}	Move Peripheral Data
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG.{BWL}	Negate
NOP	No Operation
NOT.{BWL}	One's Complement
OR.{BWL}	Logical Or
PEA	Push Effective Address
RESET	Reset External Devices
ROL.{BWL}	Rotate Left without Extend
ROR.{BWL}	Rotate Right without Extend
ROXL.{BWL}	Rotate Left with Extend
ROXR.{BWL}	Rotate Right with Extend
RTE	Return from Exception
RTR	Return and Restore
RTS	Return from Subroutine
SBCD	Subtract Decimal with Extend
Scc	Set Conditional
STOP	Stop
SUB[QX].{BWL}	Subtract
SWAP	Swap Data Register Halves
TAS	Test and Set Operand
TRAP	Trap
TRAPV	Trap on Overflow
TST.{BWL}	Test
UNLK	Unlink

In the above table BWL represent byte, word, long respectively; Q indicates a quick variant where the immediate data forms part of the instruction op-code, for ADDQ and SUBQ data must lie in range 1 to 8, for MOVEQ range is -128 to 127; X indicates that the X bit of the CCR is used (for multiple precision arithmetic). The branch instructions Bcc, BRA and BSR may option-ally be followed by .S for branches within -128 to 127 (excluding 0, -1) bytes.

Condition Codes

The following two letter codes may be substituted in place of cc for the conditional instructions in the above table.

CC	carry clear	$\overline{C}$
CS	carry set	C
EQ	equal to zero	Z
F	always false (not branches)	0
GE	greater than or equal to zero	$N.V+\overline{N}.\overline{V}$
GT	greater than zero	$N.V.\overline{Z}+\overline{N}.\overline{V}.\overline{Z}$
HI	higher	$\overline{C}.\overline{Z}$
LE	less than or equal to zero	$Z+N.\overline{V}+\overline{N}.V$
LT	less than zero	$N.\overline{V}+\overline{N}.V$
MI	minus	N
NE	not equal to zero	$\overline{Z}$
PL	plus	$\overline{N}$
RA	always true (branches only)	1
T	always true (not branches)	1
VC	overflow clear	$\overline{V}$
VS	overflow set	V

In the above the following letters are used for the condition code bits:

Z set if result zero, cleared otherwise
V set if overflow generated, cleared otherwise
C set if a carry generated, cleared otherwise
X extension bit, used for multiple precision

Addressing Modes

Dn	Data register direct
An	Address register direct
(An)	Address register indirect
(An)+	Address register indirect with postincrement
-(An)	Address register indirect with predecrement
VAL(An)	Address register indirect with displacement
VAL(An,Rn:{WL})	Address register indirect with index
ADDRESS:{WL}	Absolute address
#VAL	Immediate data

Numerical values are preceded by $ = hex, % = binary, ' = ASCII, ^ = octal. Ones followed by a full stop are decimal.

In the above:-

An is an address register, a0-a7 (SP (stack pointer) is a synonym for A7).

Dn is a data register, d0-d7.

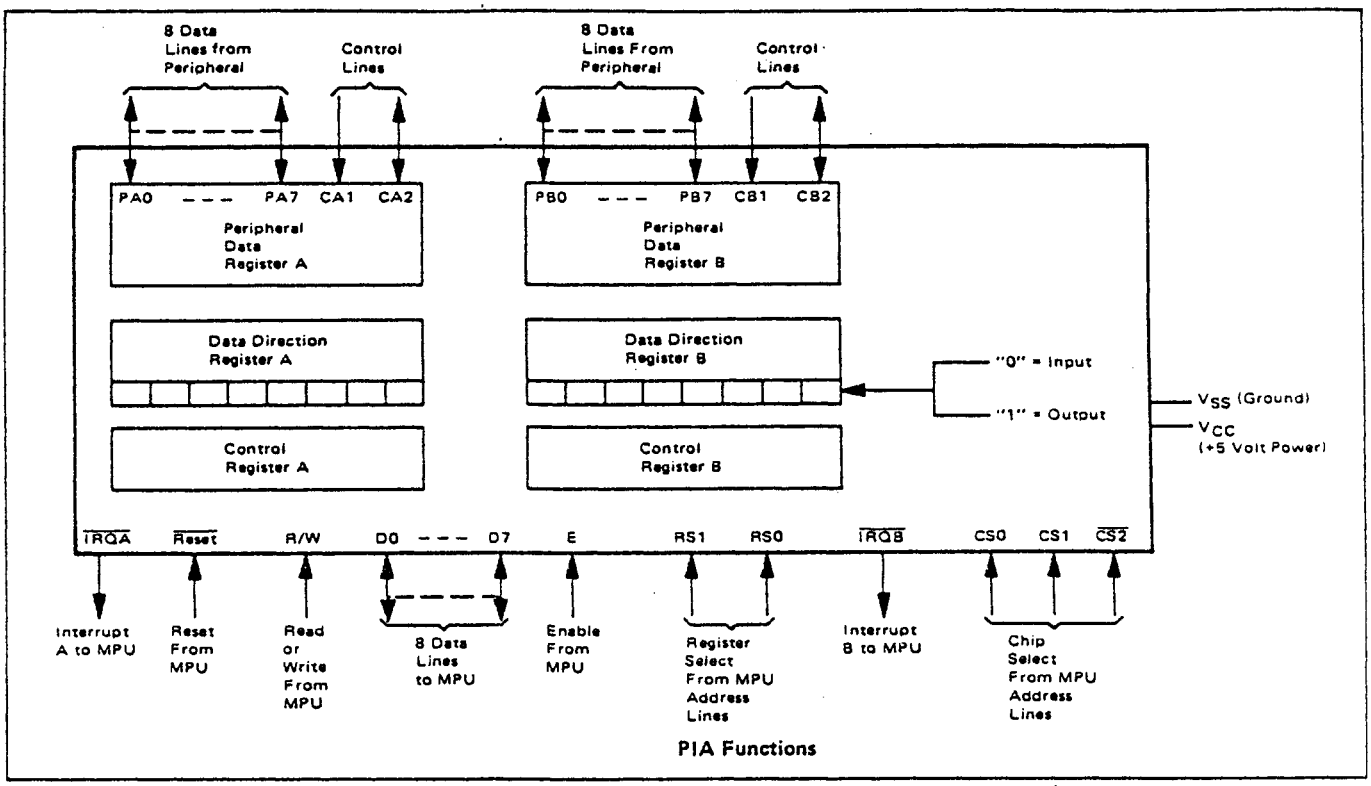
Rn is a data or address register.

Other registers are:-

USP = user stack pointer,
CCR = condition code register (lower 8 bits of SR); SR = status register;
PC = program counter.

Assembler Directives (Pseudo-ops)

.ascii 'string'	initialise store with string
.asciz 'string'	initialise store with null terminated string
.blkb count	zero fill count bytes of store
.blkw count	zero fill count words of store
.blkl count	zero fill count long words of store
.bss	switch to BSS segment
.byte val1,...,valN	initialise store with byte values
.comm name,count	reserve count bytes with label name in BSS segment
.data	switch to data segment
.end	Program end.
.globl symbol1,...,symbolN	declare global symbols
.insrt file	take input from file
.long val1,...,valN	initialise store with long word values
.text	switch to text segment
.word val1,...,valN	initialise store with word values
LABEL=expression	assign value to LABEL



PIA Functions

TABLE 1 – INTERNAL ADDRESSING

RS1	RS0	Control Register Bit CRA-2	Control Register Bit CRB-2	Location Selected
0	0	1	X	Peripheral Register A
0	0	0	X	Data Direction Register A
0	1	X	X	Control Register A
1	0	X	1	Peripheral Register B
1	0	X	0	Data Direction Register B
1	1	X	X	Control Register B

X = Don't Care

TABLE 2 – CONTROL WORD FORMAT

	7	6	5	4	3	2	1	0
CRA	IRQA1	IRQA2	CA2 Control			DDRA Access	CA1 Control	
CRB	IRQB1	IRQB2	CB2 Control			DDRB Access	CB1 Control	

TABLE 3 – CONTROL OF INTERRUPT INPUTS CA1 AND CB1

CRA-1 (CRB-1)	CRA-0 (CRB-0)	Interrupt Input CA1 (CB1)	Interrupt Flag CRA-7 (CRB-7)	MPU Interrupt Request IRQA (IRQB)
0	0	↓ Active	Set high on ↓ of CA1 (CB1)	Disabled — IRQ remains high
0	1	↓ Active	Set high on ↓ of CA1 (CB1)	Goes low when the interrupt flag bit CRA-7 (CRB-7) goes high
1	0	↑ Active	Set high on ↑ of CA1 (CB1)	Disabled — IRQ remains high
1	1	↑ Active	Set high on ↑ of CA1 (CB1)	Goes low when the interrupt flag bit CRA-7 (CRB-7) goes high

Notes:
1. ↑ indicates positive transition (low to high)
2. ↓ indicates negative transition (high to low)
3. The Interrupt flag bit CRA-7 is cleared by an MPU Read of the A Data Register, and CRB-7 is cleared by an MPU Read of the B Data Register.
4. If CRA-0 (CRB-0) is low when an interrupt occurs (Interrupt disabled) and is later brought high, IRQA (IRQB) occurs on the positive transition of CRA-0 (CRB-0).

TABLE 4 – CONTROL OF CA2 AND CB2 AS INTERRUPT INPUTS
CRA5 (CRB5) is low

CRA-5 (CRB-5)	CRA-4 (CRB-4)	CRA-3 (CRB-3)	Interrupt Input CA2 (CB2)	Interrupt Flag CRA-6 (CRB-6)	MPU Interrupt Request IRQA (IRQB)
0	0	0	↓ Active	Set high on ↓ of CA2 (CB2)	Disabled — IRQ remains high
0	0	1	↓ Active	Set high on ↓ of CA2 (CB2)	Goes low when the interrupt flag bit CRA-6 (CRB-6) goes high
0	1	0	↑ Active	Set high on ↑ of CA2 (CB2)	Disabled — IRQ remains high
0	1	1	↑ Active	Set high on ↑ of CA2 (CB2)	Goes low when the interrupt flag bit CRA-6 (CRB-6) goes high

Notes: 1. ↑ indicates positive transition (low to high)

2. ↓ indicates negative transition (high to low)

3. The Interrupt flag bit CRA-6 is cleared by an MPU Read of the A Data Register and CRB-6 is cleared by an MPU Read of the B Data Register.

4. If CRA-3 (CRB-3) is low when an interrupt occurs (Interrupt disabled) and is later brought high, IRQA (IRQB) occurs on the positive transition of CRA-3 (CRB-3).

TABLE 5 – CONTROL OF CB2 AS AN OUTPUT
CRB-5 is high

CRB-5	CRB-4	CRB-3	CB2	
			Cleared	Set
1	0	0	Low on the positive transition of the first E pulse following an MPU Write "B" Data Register operation.	High when the interrupt flag bit CRB-7 is set by an active transition of the CB1 signal.
1	0	1	Low on the positive transition of the first E pulse following an MPU Write "B" Data Register operation.	High on the positive transition of the next "E" pulse.
1	1	0	Low when CRB-3 goes low as a result of an MPU Write in Control Register "B".	Always low as long as CRB-3 is low. Will go high on an MPU Write in Control Register "B" that changes CRB-3 to "one".
1	1	1	Always high as long as CRB-3 is high. Will be cleared when an MPU Write Control Register "B" results in clearing CRB-3 to "zero".	High when CRB-3 goes high as a result of an MPU write into control register "B".

TABLE 6 – CONTROL OF CA2 AS AN OUTPUT
CRA-5 is high

CRA-5	CRA-4	CRA-3	CA2	
			Cleared	Set
1	0	0	Low on negative transition of E after an MPU Read "A" Data operation.	High on an active transition of the CA1 signal.
1	0	1	Low immediately after an MPU Read "A" Data operation.	High on the negative edge of the next "E" pulse.
1	1	0	Low when CRA-3 goes low as a result of an MPU Write in Control Register "A".	Always low as long as CRA-3 is low.
1	1	1	Always high as long as CRA-3 is high.	High when CRA-3 goes high as a result of a Write in Control Register "A".

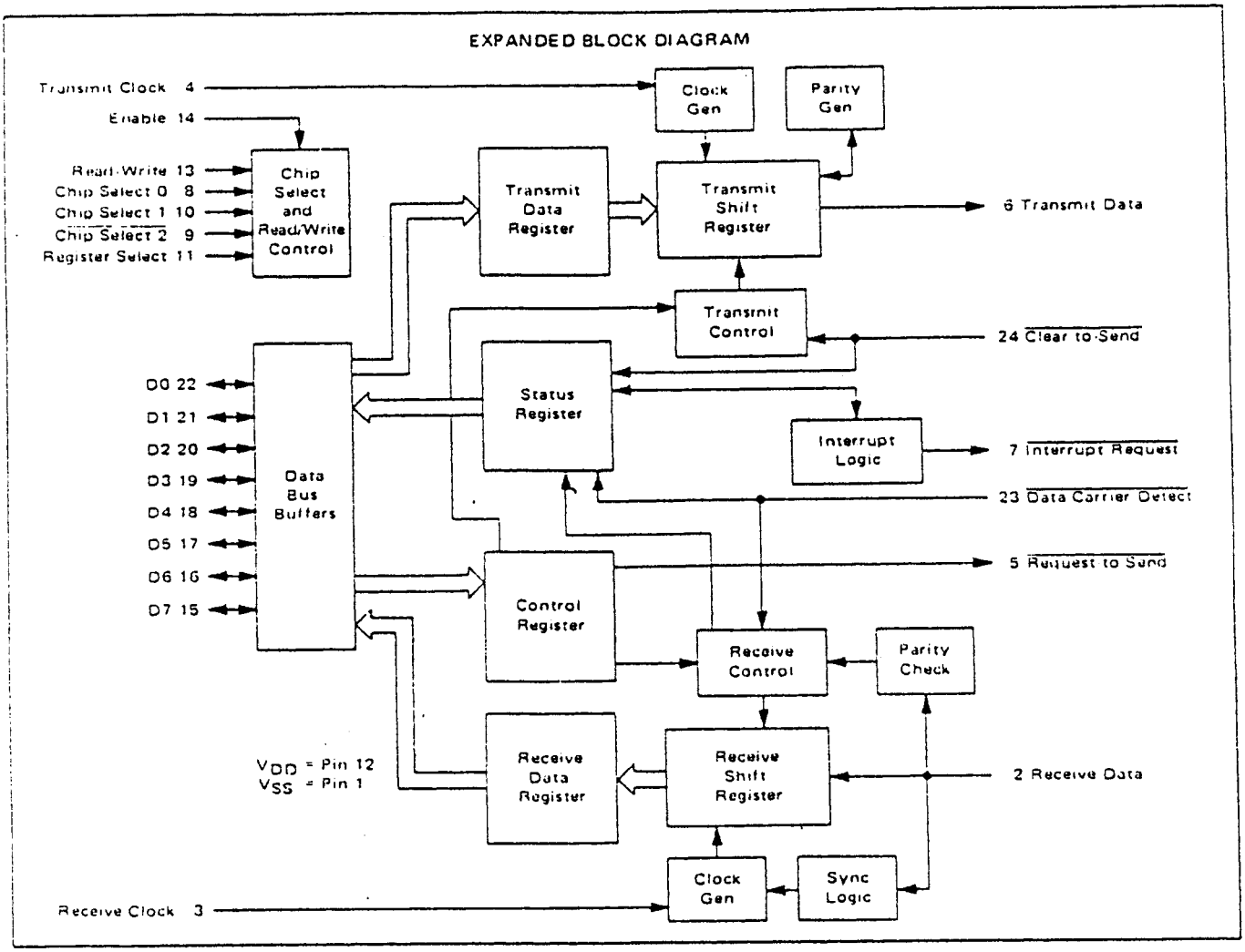


TABLE 1 – DEFINITION OF ACIA REGISTER CONTENTS

Data Bus Line Number	Buffer Address			
	RS • $\overline{R/W}$ Transmit Data Register	RS • R/W Receive Data Register	$\overline{RS}$ • $\overline{R/W}$ Control Register	$\overline{RS}$ • R/W Status Register
	(Write Only)	(Read Only)	(Write Only)	(Read Only)
0	Data Bit 0*	Data Bit 0	Counter Divide Select 1 (CR0)	Receive Data Register Full (RDRF)
1	Data Bit 1	Data Bit 1	Counter Divide Select 2 (CR1)	Transmit Data Register Empty (TDRE)
2	Data Bit 2	Data Bit 2	Word Select 1 (CR2)	$\overline{\text{Data Carrier Detect}}$ ($\overline{DCD}$)
3	Data Bit 3	Data Bit 3	Word Select 2 (CR3)	$\overline{\text{Clear to Send}}$ ($\overline{CTS}$)
4	Data Bit 4	Data Bit 4	Word Select 3 (CR4)	Framing Error (FE)
5	Data Bit 5	Data Bit 5	Transmit Control 1 (CR5)	Receiver Overrun (OVRN)
6	Data Bit 6	Data Bit 6	Transmit Control 2 (CR6)	Parity Error (PE)
7	Data Bit 7***	Data Bit 7**	Receive Interrupt Enable (CR7)	Interrupt Request (IRQ)

* Leading bit = LSB = Bit 0
** Data bit will be zero in 7-bit plus parity modes.
*** Data bit is "don't care" in 7-bit plus parity modes.

COUNTER DIVIDE SELECT BITS (CR0 AND CR1)

CR1	CR0	Function
0	0	÷ 1
0	1	÷ 16
1	0	÷ 64
1	1	Master Reset

WORD SELECT BITS (CR2, CR3, AND CR4)

CR4	CR3	CR2	Function
0	0	0	7 Bits + Even Parity + 2 Stop Bits
0	0	1	7 Bits + Odd Parity + 2 Stop Bits
0	1	0	7 Bits + Even Parity + 1 Stop Bit
0	1	1	7 Bits + Odd Parity + 1 Stop Bit
1	0	0	8 Bits + 2 Stop Bits
1	0	1	8 Bits + 1 Stop Bit
1	1	0	8 Bits + Even Parity + 1 Stop Bit
1	1	1	8 Bits + Odd Parity + 1 Stop Bit

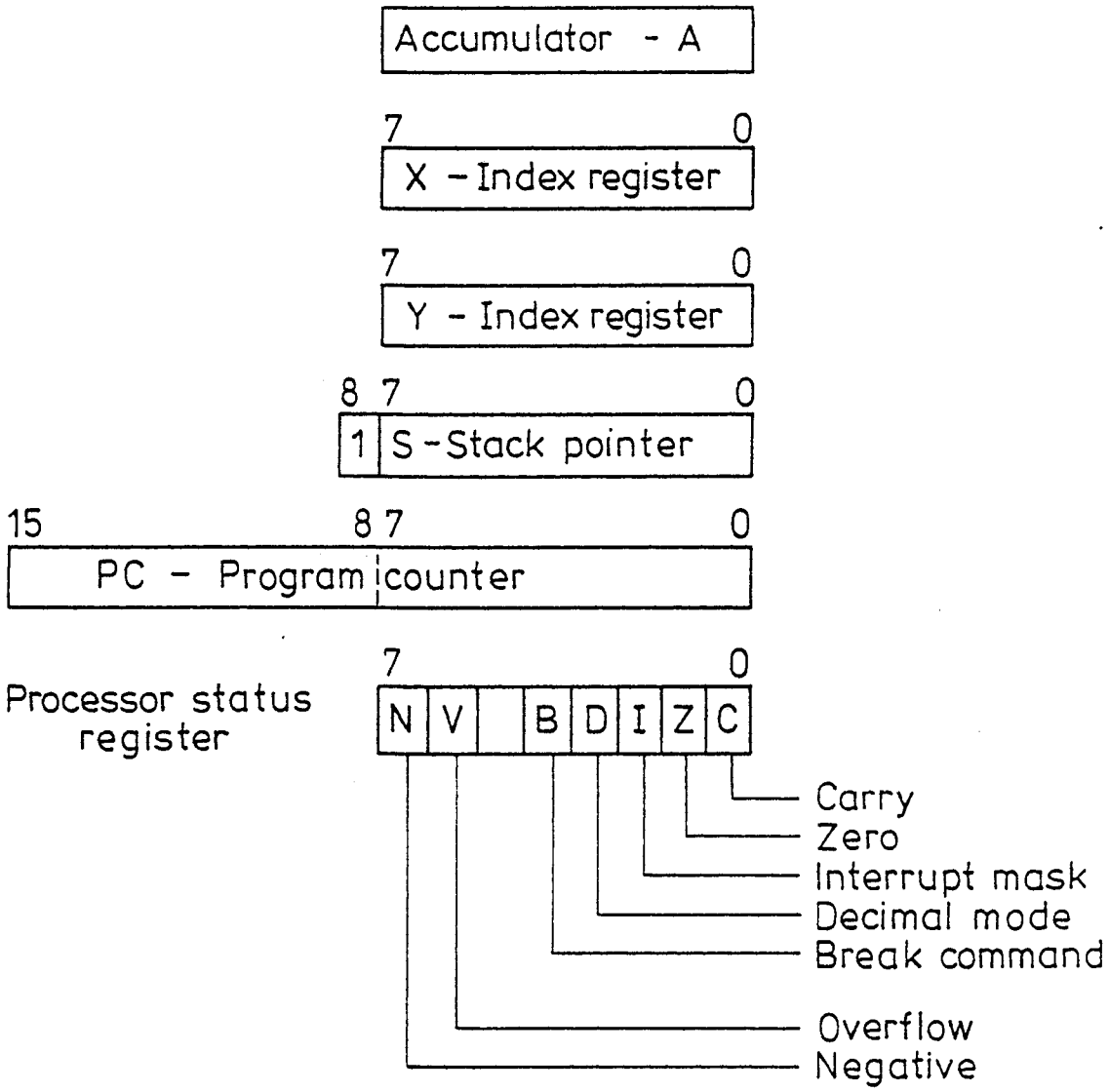
TRANSMITTER CONTROL BITS (CR5 AND CR6)

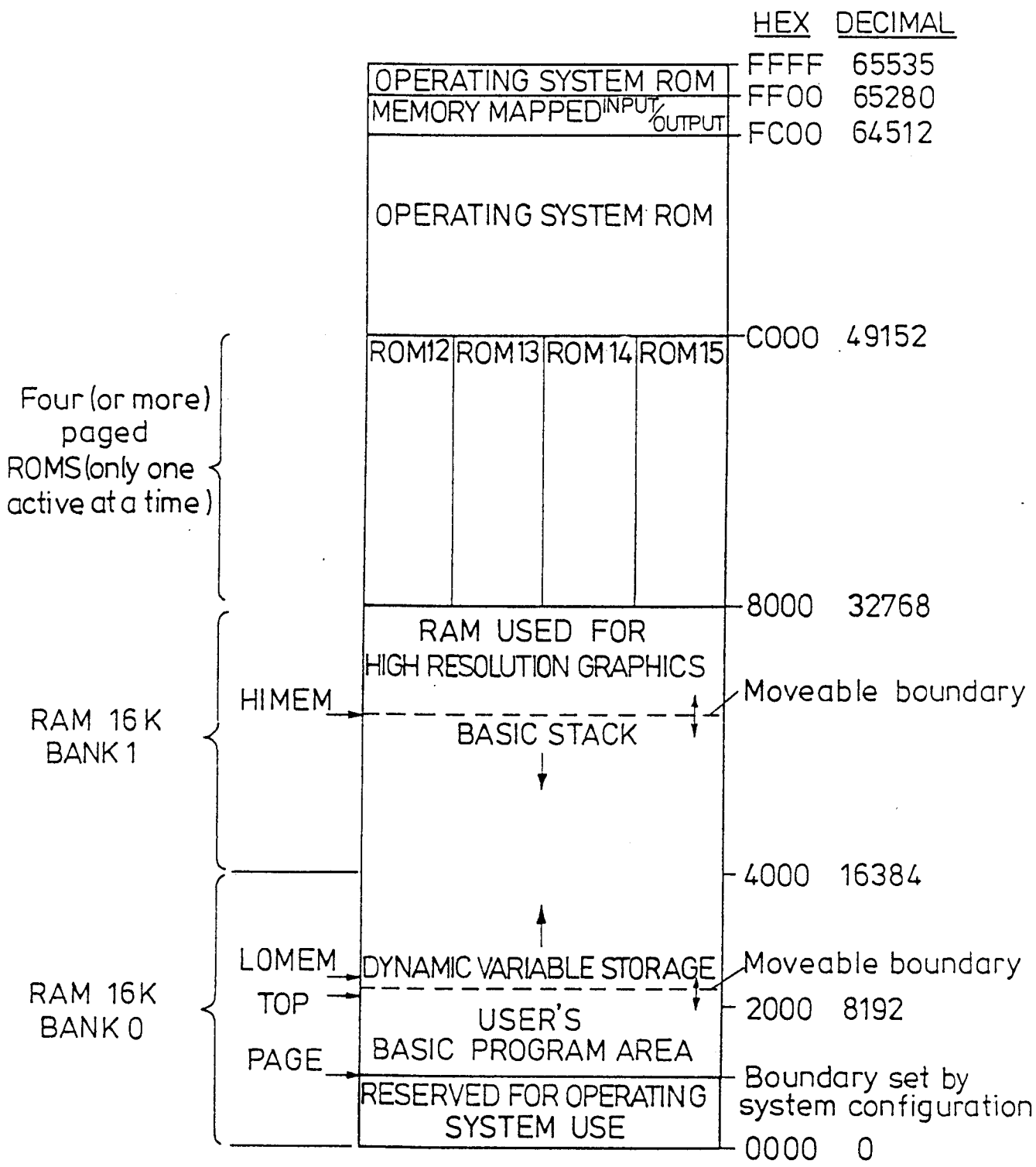
CR6	CR5	Function
0	0	$\overline{RTS}$ = low, Transmitting Interrupt Disabled.
0	1	$\overline{RTS}$ = low, Transmitting Interrupt Enabled
1	0	$\overline{RTS}$ = high, Transmitting Interrupt Disabled
1	1	$\overline{RTS}$ = low, Transmits a Break level on the Transmit Data Output. Transmitting Interrupt Disabled.

Receive Interrupt Enable Bit (CR7) – The following interrupts will be enabled by a high level in bit position 7 of the Control Register (CR7): Receive Data Register Full, Overrun, or a low to high transistion on the Data Carrier Detect ($\overline{DCD}$) signal line.

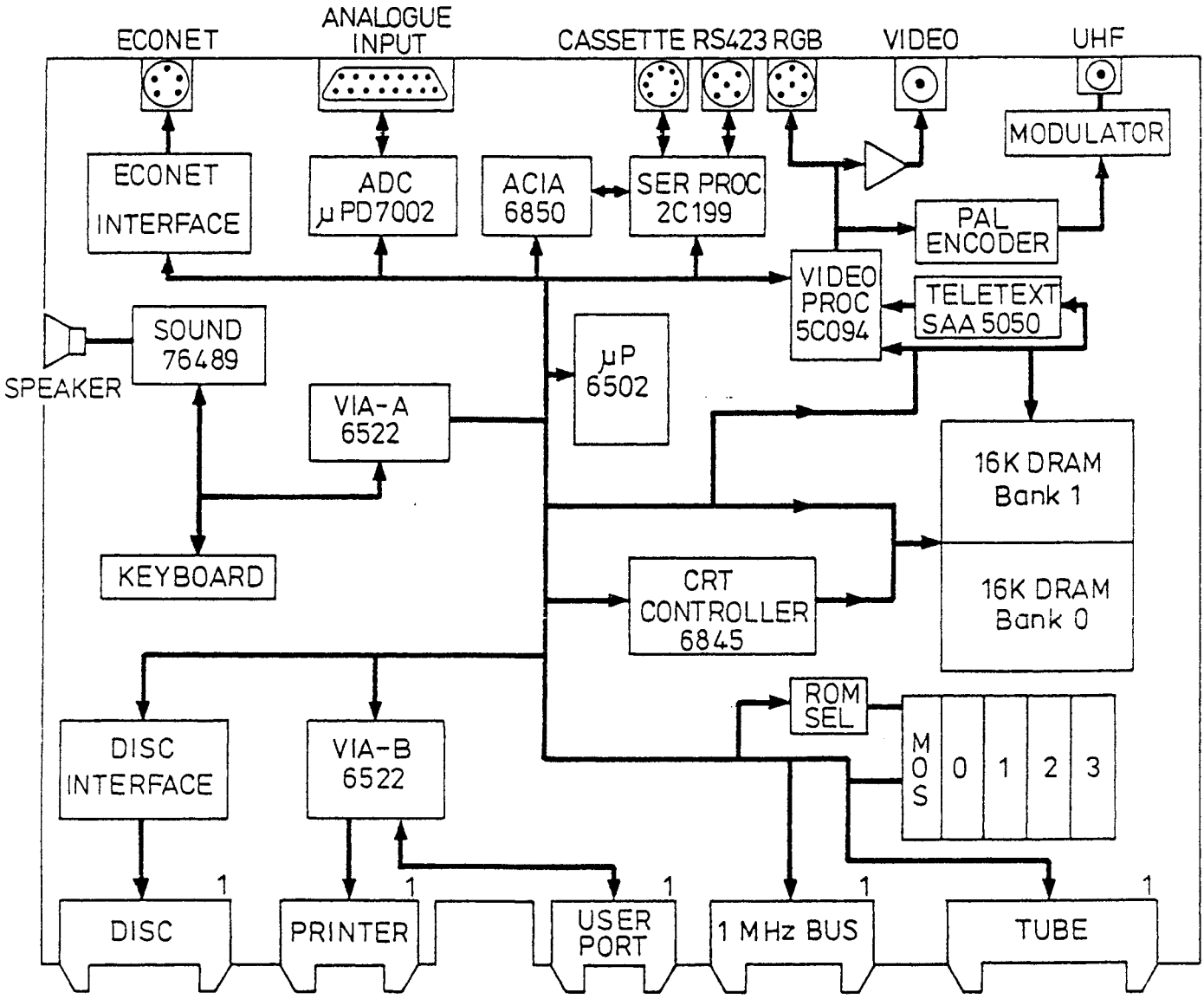
NOTES

1. RDRF is cleared by reading the data register.
2. TDRE indicates that a character can be written to the data register.
3. If $\overline{DCD}$ is high then RDRF will indicate empty unconditionally.
4. If $\overline{CTS}$ is high then TDRE is inhibited. Master reset has no effect on the $\overline{CTS}$ status bit.


Accumulator - A
7
0
X - Index register
7
0
Y - Index register
8 7
0
1
S - Stack pointer
15
8 7
0
PC - Program counter
Processor status register
7
0
N V B D I Z C
Carry
Zero
Interrupt mask
Decimal mode
Break command
Overflow
Negative


HEX
DECIMAL
OPERATING SYSTEM ROM
FFFF 65535
MEMORY MAPPED INPUT/OUTPUT
FF00 65280
FC00 64512
OPERATING SYSTEM ROM
C000 49152
ROM12
ROM13
ROM14
ROM15
Four (or more) paged ROMS (only one active at a time)
8000 32768
RAM USED FOR HIGH RESOLUTION GRAPHICS
HIMEM
Moveable boundary
BASIC STACK
RAM 16K BANK 1
4000 16384
LOMEM
DYNAMIC VARIABLE STORAGE
Moveable boundary
TOP
2000 8192
USER'S BASIC PROGRAM AREA
PAGE
Boundary set by system configuration
RESERVED FOR OPERATING SYSTEM USE
RAM 16K BANK 0
0000 0

BBC MICROCOMPUTER BLOCK DIAGRAM



BBC MICROCOMPUTER REFERENCE SUMMARY (abridged)
(From the Computing Service document by K. Edgcombe, 1983)

1. Typing in and running BASIC programs

AUTO n,m	line numbering: start m, step n	AU.
RENUMBER m,n	start m, step n	REN.
LIST n,m	n to m (or n or n, or ,m)	L.
LISTOn	sets list options (0 to 7)	LISTO
CTRL U	delete line being typed	
line-number	alone deletes one line	
DELETE n,m	delete lines n to m inclusive	DEL.
CTRL N, CTRL O	set & unset page mode (SHIFT steps on)	
:	separates commands on same line	
RUN	runs current program	RUN

2. Saving and loading BASIC programs

Program name (on tape) = up to 10 characters, no spaces
(on disc) = :n.x.name, name up to 7 chars

SAVE "name"	current program to tape (or disc)	SA.
LOAD "name"	into memory; ""= next on tape	LO.
CHAIN "name"	load and run; "" as above	CH.
CLEAR	vars. from memory (not A% to Z%)	CL.
NEW	before new program	NEW
OLD	restore old program after BREAK	O.

3. Special keys

ESCAPE (key)	normally stops a BASIC program
BREAK (key)	soft reset: clears variables and program
CTRL BREAK (keys)	hard reset including soft keys & clock
SHIFT BREAK (keys)	carry out auto-boot option (disc)
COPY (key)	from text cursor to block cursor
*KEY n text	programs user-defined (soft) keys:
(red keys 1-9, 10=BREAK, 11-15=COPY and arrows, see *FX4,2)	
	¦M for return, ¦x for CTRL x,
	¦! add 128 to next ASCII code
*FX18	reset (cancel) user-defined keys
VDU 23,n,m1,..m8	programs ASCII n to new char.

4. Text cursor

Arrow keys	step left, right, down,up	
VDU 8 (9,10 or 11)	left, right, down, up one step	
CTRL H (I,J or K)	ditto	
VDU 31,m,n	to m across, n down (n range 0-24, m range 0-19 or 0-39 or 0-79)	V.
PRINT TAB(m,n)	(TAB(m) to position m across)	P.TAB
VDU 30	to top left of text window	V.
VDU 13	to left of current line	V.
POS	=horizontal position	POS
VPOS	=vertical position	VP.
VDU 23,1,0;0;0;0;	switch cursor off	V.
VDU 23,1,1;0;0;0;	switch cursor on	V.

5. Screen and window control

```
CLS                          clear screen (text window)          CLS
*TV n                        move display (1=up 1 line, 255=down)
*TV 0,1                      turn off display interlace          *TV
VDU 21                       disable screen output               V.
VDU 6                        re-enable screen output             V.
VDU 28,xmin,ymin,xmax,ymax   define text window                  V.
VDU 26                       reset text and graphics windows     V.
VDU 5                        text at graphics cursor             V.
VDU 4                        text at text cursor                 V.
```

6. BASIC variables and constants

```
Variable name:      letter + letters, digits or _
Integer variable name : name%
String variable name:   name$
Resident integers:      A% to Z% and @%
Range: reals            + 2x10^-39 to 2x10^+38
       integers         -2147483648 to +2147483647
       strings          up to 255 characters
Accuracy (reals)        9 significant figures
Special variable names:
  FALSE = 0                                                  FA.
  TRUE  = -1                                                 TRUE
  PI    = 3.14159265                                         PI
  TIME  = centiseconds from arbitrary start                  TI.
[LET] variable=exprn    assignment                           LET
DIM name1(n),name2(m)   declare arrays                       DIM
DIM name3(n1,n2)..      etc.
LOCAL name1,name2..     local vars in procedure              LOC.
```

7. Structure in a BASIC program

```
REPEAT : commands : UNTIL condition                          REP. U.
IF condition THEN commands (terminated by end of line)
IF condition THEN commands ELSE commands                     IF.TH.EL.
FOR v=n1 TO n2 STEP n3 : commands : NEXT [v]                 F.TO S.N.
GOTO n                  unconditional jump                   G.
GOSUB n                 jump keeping return address          GOS.
END                     end of execution                     END
STOP (fatal in Basic 2)end and output message                STO.
RETURN                  to last stored return address        R.
ON v GOTO n1,n2..       if v=1 goto n1 etc.                  ON G.
ON v GOSUB n1,n2..      similarly                            ON GOS.
DEF PROCname(v1,v2,...) DEF
  commands              defines procedure                    PRO.
  ENDPROC                                                    E.
PROCname(arg1,arg2,..)  calls procedure                      PRO.
DEF FNname(v1,v2,...)                                        DEF
  commands              defines function                     FN
  =expression
v=FNname(arg1,arg2..)   calls function                       FN
Conditions:- TRUE, FALSE, any numeric (0=FALSE),             TRUE,FA.
             comparisons: = <> < > <= or >=
             compounds with AND, OR, NOT and ()              A. OR NOT
```

8. Operators and precedence

```
Group 1: unary + and -, NOT, functions, brackets, ?, !, $
Group 2: ^ (to power)
Group 3: *, /, DIV, MOD
Group 4: +, -
Group 5: =, <>, <, >, <=, >=
Group 6: AND
Group 7: OR, EOR
```

9. Built-in functions

```
Random numbers:
  RND(X), RND(-X) (reset generator), RND (maximum range)
Arithmetic:
  n MOD m, n DIV m, SQR, ABS, SGN, LN, LOG, EXP
Trigonometric:
  SIN, COS, TAN, ASN, ACS, ATN, DEG, RAD
Strings:
  EVAL(string)          evaluate expression in string        EV.
  VAL(string)           turn number string into number       VAL
  STR$(num)             turn number into number string       STR.
  ASC(string)           ASCII value of 1st char in string    ASC
  CHR$(n)               character whose ASCII value is n     CHR.
STRINGS(n,string)       multiple copies to length n          STRI.
LEN(string)             length                               LEN
INSTR(str1,str2)        posn of str2 in str1                 INS.
LEFT$(string,n)         left n characters of string          LE.
RIGHT$(string,n)        rt. n characters of string           RI.
MID$(str,n1,n2)         n2 chars from n1 on                  M.
```

10. Input

```
INPUT ["text",]vars     [prompt and] read from keyboard      I.
INPUT LINE stringvar    include commas & leading spaces      I.LIN.
INPUT TAB(x,y) "text",vars move to (x,y) first               I.TAB
INPUT SPC(n)            move on n spaces                     I.SPC
GET                     =ASCII number of key pressed         GET
GET$                    =string containing key pressed       GE.
INKEY(n)                as GET but wait only n centisecs.    INKEY
                        gives -1 if no key pressed
INKEY(-n)               test if a key is currently pressed
INKEY$(n)               as GET$ but wait only n centisecs.   INK.
*FX 15,1                flushes keyboard input buffer
READ var1,var2...       from DATA list                       REA.
DATA val1,val2...       set up DATA list                     D.
RESTORE n               data pointer to line n               RES.
ADVAL(n)                reads fromm analogue input channel n AD.
ADVAL(0)                `fire' buttons, last conversion, etc.
ADVAL(-n)               tests various internal buffers
```

11. Output to screen or printer

```
PRINT varlist           vars in list separated by            P.
                        commas for 10-character fields on same line
                        (strings at left of field, numbers at rt.),
                        semicolons to print adjacent on same line,
                        ' to insert new line.
                        ~var prints var in hexadecimal
```

```
PRINT TAB(x,y) or TAB(x) start at (x,y) or x                    P.TAB
PRINT SPC(n)           print n spaces                           P.SPC
COUNT                  =no. of chars. since last newline        COU.
```

Print formats:

@%=&B4B3B2B1 (default value 10) where

- B1=print field width
- B2=digits (total, mantissa, or after decimal pt)
- B3=0 for G format, 1 for E format, 2 for F format
- B4=1 if @% is to affect STR$, 0 otherwise

Printer:

```
VDU 2                  enable printer
CTRL B                 enable printer
VDU 3                  disable printer
CTRL C                 disable printer
*FX5,1                 use parallel printer line
*FX5,2                 use serial printer line
*FX6,0                 allow linefeeds to be sent to printer
```

12. File handling

```
file name (tape)  =    up to 10 characters, no spaces
file name (disc)  =    :n.x.name (name up to 7 chars)
n=OPENIN(string)       open for input                           OP.
n=OPENUP(string)       open to update (Basic 2 only)
n=OPENOUT(string)      open for output                          OPENO.
INPUT# n,varlist       read variables (internal format)         I.#
PRINT# n,varlist       output (internal format)                 P.#
m=BGET#(n)             read one byte                            B.#
BPUT# n,m              write one byte                           BP.#
CLOSE# n               close file                               CLO.#
m=EOF#(n)              -1 if end, 0 otherwise                   EOF#
m=PTR# n               pointer(disc only)                       PT.#
m=EXT#(n)              length (disc only)                       EXT#
```

13. Assembler and memory control

```
CALL n [parms]         to m/c code routine at address n         CA.
USR(n)                 =result of m/c code function at n        USR
HIMEM                  top of available memory                  H.
LOMEM                  bottom of available memory               LOM.
TOP                    top of user's program                    TOP
PAGE                   bottom of user's program                 PA.
?M                     contents of byte M
M?N                    contents of M+N (M must be variable)
!M                     4 bytes at M
M!N                    4 bytes at M+N (M must be variable)
$M                     string at byte M onwards
                       (newline (ASCII 13) terminates)
[ and ]                bracket Assembler code
```

Call to OSCLI (CALL &FFF7) passes string to OS cmd line interpreter (string addressed by X% and Y%)

14. Sound

```
VDU 7                   short beep (bell)                    V.
*FX211,n                alter bell channel
*FX213,n                alter bell frequency
*FX214,n                alter bell duration

SOUND &HSFC,A,P,D       sound defined by                     SO.
  H=1 to allow previous note to continue, 0 otherwise
  S=n to synchronise with n other channels (n=0 to 2)
  F=1 to override and flush channel, 0 otherwise
  C=channel number, 0 to 3
  A=amplitude (-15 to 0) or envelope number (1-4)
  P=pitch (0-255, middle C=53) for channels 1-3
  P=pitch range for channel 0 (noise):
    0-3 periodic: high, medium, low or as channel 1
    4-7 white: high, medium, low or as channel 1
  D=duration (unit 1/20 sec) 0 to 254 or -1 (indefinite)

ENVELOPE N,T,PI1,PI2,PI3,PN1,PN2,PN3,AA,AD,AS,AR,ALA,ALD
  N=envelope number                                         ENV.
  T=step length (centiseconds, 0 to 127):
    add 128 to switch off auto-repeat
  PI1 to PI3 pitch change/step in 3 sections (-128 to 127)
  PN1 to PN3 no. of steps in sections 1 to 3 (0 to 255)
  AA,AD,AS,AR: amplitude change/step in Attack, Decay,
    Sustain and Release phases
    (-127 to 127 (AA & AD) or to 0 (AS & AR))
  ALA and ALD: target amplitude at end of
    Attack and Decay phases (0-126)

*FX21,n   flushes sound channels, n=4-7 for channels 0-3
*FX210,1  suppresses all sound
```

15. Modes

Mode	Graphics	Colours	Text	screen map
0	640x256	2	80x32	20K
1	320x256	4	40x32	20K
2	160x256	16	20x32	20K
3	-	2	80x25	16K
4	320x256	2	40x32	10K
5	160x256	4	20x32	10K
6	-	2	40x25	8K
7	-	Teletext display	40x25	1K

```
Modes available on Model A: 4, 5, 6 and 7
MODE n                  changes to mode n                    MO.
VDU 22,n                ditto                                V.
Mode at switch-on is usually 7.
```

16. Colour (Modes 0 to 6)

COLOUR n	set text colour	C.
GCOL m,n	set graphics action and colour	GC.

Action in GCOL:
0: plot the colour specified
1, 2, 3: OR, AND or EOR colour with that already there
4: invert colour already there

POINT(x,y)	= logical colour at graphical (x,y)	PO.
VDU 19,m,n,0,0,0	sets logical colours: m=logical, n=actual	V.
VDU 20	resets all colours to defaults	V.

2-colour modes	0=black	1=white
4-colour modes	0=black	1=red
	2=yellow	3=white

16-colour mode (2)

0=black	1=red	2=green	3=yellow
4=blue	5=magenta	6=cyan	7=white
8=flashing black-white		9=flashing red-cyan	
10=flashing green-magenta		11=flashing yellow-blue	
12=flashing blue-yellow		13=flashing magenta-green	
14=flashing cyan-red		15=flashing white-black	

To change background, add 128 to colour

17. Graphics

Origin initially at bottom left:
Range of x: 0 to 1279. Range of y: 0 to 1023.

DRAW x,y	draw straight line to x,y	DR.
MOVE x,y	move without drawing	MOV.
PLOT k,x,y	draw to x,y as defined by k	PL.

Values of k:
0 move relative to last point
1 draw relative to last point
2 draw relative in logical inverse colour
3 draw relative in background colour
4 move to absolute posion (=MOVE)
5 draw to absolute position (=DRAW)
6 draw absolute in logical inverse colour
7 draw absolute in background colour
Add 8 to above to omit last point in 'inverting actions'
Add 16 to above for a dotted line
Add 24 to above for effects of 8 and 16
~~Add 64 to above to plot single point only~~
Add 80 to above to fill triangle (x,y + last 2 pts).

VDU 24,xmin;ymin;xmax;ymax;	sets graphics window	V.
VDU 29,x;y;	moves graphics origin	V.