User's manual

ACORN COMPUTER
Acorn Computor Limited, 4a Market Hill, Cambridge,
CB2 3MJ, Telephone (0223) 312772

CONTENTS

PART 1

CHAPTER 1: AN INTRODUCTION TO THE BINARY NUMBER SYSTEM

1.1 BINARY NUMBERS:

NUMBERS IN EVERY DAY USE ARE WRITTEN IN THE DECIMAL SYSTEM, THAT IS, TO THE NUMBER BASE 1Ø. A POSITIONAL NOTATION IS USED REPRESENTING ONE '1ØØ's; TWO '1Ø's & EIGHT '1's AS THE SYMBOL 128. THE RIGHTMOST (i.e. LEAST SIGNIFICANT) DIGIT IS IN THE "UNITS" COLUMN, THE 2 IN THE "TENS" COLUMN, THE 1 IN THE "HUNDREDS" COLUMN, AND THE VALUE OF THE SYMBOL '128' IS EVALUATED AS 1x1ØØ+2x1Ø + 8x1 = 128. SIMILARLY '1Ø24' IS EVALUATED AS 1x1ØØØ + Øx1ØØ + 2x1Ø + 4x1 = 1Ø24, WHICH IS MORE CONVENIENTLY WRITTEN AS 1x1Ø^{3} + Øx1Ø^{2} + 2x1Ø^{1} 4x1Ø^{0} = 1Ø24, USING THE MATHEMATICAL SHORTHAND FOR 1ØØØ = 1Øx1Øx1Ø = 1Ø^{3}, AND THE CONVENTION "ANY NUMBER TO THE POWER ZERO IS 1" TO GIVE A CONSISTENT METHOD OF EVALUATING SUCH SYMBOLS.

SO 1Ø24
CAN BE WRITTEN IN COLUMNS

3	2	1	Ø
1	Ø	2	4

AND EVALUATED AS 1x1Ø^{3} + Øx1Ø^{2} + 2x1Ø^{1} + 4x1Ø^{0}
TO THE BASE 1Ø.

TO THE BASE 8, 1Ø24 WOULD MEAN 1x8^{3} + Øx8^{2} + 2x8^{1} + 4x8^{0} WHICH IS THE DECIMAL NUMBER 532.

TO THE BASE 16, 1Ø24 WOULD MEAN 1x16^{3} + Øx16^{2} + 2x16^{1} + 4x16^{0} WHICH IS THE DECIMAL NUMBER 4132.

TO DISTINGUISH THE BASE TO WHICH A NUMBER IS WRITTEN WE'LL WRITE ITS' BASE AFTER IT AS A SUBSCRIPT: 1Ø24$_{10}$ AND NOW WE CAN WRITE

1Ø24$_{8}$ = 532$_{10}$
1Ø24$_{16}$ = 4132$_{10}$
1ØØØØØØØ$_{2}$ = 128$_{10}$

JUST AS BASE TEN HAS THE NAME 'DECIMAL', BASE SIXTEEN HAS THE NAME 'HEXADECIMAL', BASE EIGHT HAS THE NAME 'OCTAL' AND BASE TWO 'BINARY'. THESE FOUR BASES ARE IN COMMON USE WITH MODERN COMPUTERS, ESPECIALLY HEXADECIMAL (HEX) AND BINARY. CONVERSION BETWEEN BINARY, OCTAL & HEX NUMBERS IS VERY SIMPLE. SINCE THEY ARE ALL POWERS OF TWO, NUMBERS JUST NEED DIVIDING UP:–

1ØØØØØØØ$_{2}$ = |1ØØØ||ØØØØ|$_{16}$ = 8Ø$_{16}$
= |Ø1Ø||ØØØ||ØØØ|$_{8}$ = 2ØØ$_{8}$

– EACH HEX DIGIT IS FOUR BINARY DIGITS (BITS) & EACH OCTAL DIGIT IS 3 BITS.

OCTAL DIGITS ARE Ø, 1, 2, 3, 4, 5, 6, 7.

HEX DIGITS ARE Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F A....F ARE USED INSTEAD OF 1Ø....15 TO ALLOW UNRESTRICTED USE OF THE POSITIONAL SYSTEM.

– PROGRAM COUNTER: 16 BIT REGISTER WHICH CONTAINS THE ADDRESS OF THE INSTRUCTION BEING EXECUTED. DURING EXECUTION THE PROGRAM COUNTER IS STEPPED UP TO POINT AT THE NEXT INSTRUCTION.
– PROM: PROGRAMMABLE READ ONLY MEMORY. THIS TYPE OF MEMORY ARRIVES BLANK. IT CAN BE PROGRAMMED BY THE USER WITH THE HELP OF A SPECIAL PROM BLOWER. ONCE THIS PROGRAM HAS BEEN PUT IN, IT CANNOT BE CHANGED.
– RAM: RANDOM ACCESS MEMORY. THIS IS THE STANDARD READ/WRITE MEMORY. DATA (AND PROGRAMS) ARE LOST WHEN THE POWER IS SWITCHED OFF.
– REGISTER: STORAGE LOCATION IN THE MICROPROCESSOR ITSELF. THERE ARE INTERNAL REGISTERS A, X, Y, PC, S, P.
– ROM: READ ONLY MEMORY. THIS IS MEMORY THAT HAS A PROGRAM PUT IN DURING PRODUCTION. THIS PROGRAM CANNOT EVER BE CHANGED, IT CAN ONLY BE READ.
– STORE: TRANSFERS DATA FROM AN INTERNAL REGISTER TO MEMORY.
– XTAL: THE CRYSTAL IN THE ACORN OSCILLIATES AT 1 MHZ. i.e. ONE MILLION TIMES A SECOND. IT DOES THIS WITH GREAT ACCURACY. SO YOU CAN BUILD A CLOCK FROM YOUR ACORN.

CONVERSION TABLE

HEX	DECIMAL	OCTAL	BINARY
Ø	Ø	Ø	Ø
1	1	1	1
2	2	2	1Ø
3	3	3	11
4	4	4	1ØØ
5	5	5	1Ø1
6	6	6	11Ø
7	7	7	111
8	8	1Ø	1ØØØ
9	9	11	1ØØ1
A	1Ø	12	1Ø1Ø
B	11	13	1Ø11
C	12	14	11ØØ
D	13	15	11Ø1
E	14	16	111Ø
F	15	17	1111
1Ø	16	2Ø	1ØØØØ
2Ø	32	4Ø	1ØØØØØ
4Ø	64	1ØØ	1ØØØØØØ
64	1ØØ	144	11ØØ1ØØ
8Ø	128	2ØØ	1ØØØØØØØ
1ØØ	256	4ØØ	1ØØØØØØØØ

THE ACORN MICROPROCESSOR IS DESIGNED TO DEAL WITH 8 BITS AT A TIME. THE COLLECTION OF 8 BITS IS GIVEN THE SPECIAL NAME 'BYTE', AND IS NORMALLY WRITTEN IN HEXADECIMAL OR BINARY. A BYTE THUS IS Ø....FF_{16}; Ø....11111111_2 OR Ø....255_{10}. THE MICROPROCESSOR CAN CARRY OUT LOGICAL AND ARITHMETICAL MANIPULATIONS ON BYTES.

1.2 LOGICAL MANIPULATIONS

THE MICROPROCESSOR CAN IMMEDIATELY CARRY OUT THE LOGICAL AND, EXCLUSIVE - OR & OR FUNCTIONS ON ALL 8 BITS SIMULTANEOUSLY, USING THE FOLLOWING TRUTH TABLES FOR EACH BIT (SYMBOL 'b')

AND ($\wedge$)

b_1	b_2	result
Ø	Ø	Ø
Ø	1	Ø
1	Ø	Ø
1	1	1

EXCLUSIVE - OR ($\forall$)

b_1	b_2	result
Ø	Ø	Ø
Ø	1	1
1	Ø	1
1	1	Ø

OR ($\vee$)

b_1	b_2	result
Ø	Ø	Ø
Ø	1	1
1	Ø	1
1	1	1

EXAMPLE
OPERANDS

ØØ1111ØØ		ØØ1111ØØ		ØØ1111ØØ	
Ø1Ø11Ø1Ø	AND	Ø1Ø11Ø1Ø	E-OR	Ø1Ø11Ø1Ø	OR
————	(OPERATOR)	————		————	
ØØØ11ØØØ	RESULT	Ø11ØØ11Ø		Ø111111Ø	

1.3 ARITHMETIC MANIPULATIONS

BINARY ADDITION
WITH CARRY OUTPUT

b_1	b_2	SUM	CARRY
Ø	Ø	Ø	Ø
Ø	1	1	Ø
1	Ø	1	Ø
1	1	Ø	1

BINARY ADDITION WITH CARRY FROM RIGHT

b_1	b_2	INPUT CARRY	SUM	OUTPUT CARRY TO LEFT
Ø	Ø	Ø	Ø	Ø
Ø	1	Ø	1	Ø
1	Ø	Ø	1	Ø
1	1	Ø	Ø	1
Ø	Ø	1	1	Ø
Ø	1	1	Ø	1
1	Ø	1	Ø	1
1	1	1	1	1

EXAMPLE:	ØØ1111ØØ	$3C_{16}$	$6Ø_{10}$
	ØlØ11Ø1Ø+	$5A_{16}$ +	$9Ø_{10}$ +
	1ØØ1Ø11Ø	96_{16}	$15Ø_{10}$

IN ORDER TO MAKE LONGER ADDITIONS EASIER TO PROGRAM, THE MICROPROCESSOR HAS A CARRY BIT (FLAG). AT THE START OF AN ADDITION THIS IS TREATED AS THE INPUT CARRY, AND AT THE END IT RECEIVES THE CARRY OUT FROM THE SUM AT BIT 7: ASSUMING WE HAVE A CARRY INPUT:

	11ØØØØ11		$C3_{16}$	195_{10}
	1Ø1ØØ1Ø1	CARRY IN	$A5_{16}$	165_{10}
	[1] +		1_{16} +	1_{10}+
CARRY OUT [1]	Ø11Ø1ØØ1		169_{16}	361_{10}

SUBSTRACTION OPERATES IN A SIMILAR MANNER, EXCEPT THAT THE CARRY (OR BORROW) FLAG OPERATES UPSIDE DOWN: A Ø CARRY FLAG IS TREATED AS REPRESENTING A BORROW FROM THE PREVIOUS STAGE:

	11111111	FF_{16}	255_{10}
	ØØØØØØØØ	$ØØ_{16}$	$ØØØ_{10}$
	Ø	$Ø_{16}$	$Ø_{10}$
[1]	1111111Ø	$1FE_{16}$	$51Ø_{10}$

NOT QUITE THE RESULTS ONE MIGHT HAVE WISHED FOR! (SUPERFICIALLY) THIS OCCURS BECAUSE OF THE HARDWARE IMPLEMENTATION OF SUBTRACTION A SUBTRACTION, (P–Q), IS REGARDED BY THE MICRO-PROCESSOR AS THE EQUIVALENT (P+(–Q)), BECAUSE THERE IS A SIMPLE WAY TO GENERATE THE NEGATIVE OF A NUMBER.
THE 'ONES-COMPLEMENT' OF A BINARY NUMBER IS SIMPLY GENERATED BY EXCHANGING 'Ø's & '1's:

'1's	$ØØØØ11ØØ_2$	$ØC_{16}$	12_{10}
COMPLEMENT	$1111ØØ11_2$	$F3_{16}$	243_{10}

IF THIS ONE'S-COMPLEMENT IS TO BE THE NEGATIVE OF A NUMBER, WE SHOULD GET Ø ON ADDITION:

GLOSSARY

- ACCUMULATOR: 8-BIT CENTRAL REGISTER IN THE MICROPROCESSOR. MOST INFORMATION HAS TO GO THROUGH IT.
- ADDRESS: 16 BIT POINTER TO A MEMORY LOCATION. THE 6502 MICRO-PROCESSOR CAN ADDRESS 65, 536 SUCH LOCATIONS (WHICH IS 2^{16}).
- ARITHMETIC LOGIC UNIT (A.L.U.): A SECTION OF THE MICROPROCESSOR WHICH CARRIES OUT ARITHMETIC (ADDITION, SUBTRACTION, INCREMENT, DECREMENT & COMPARE) AND LOGIC ("AND", "EOR", "OR", & BIT SHIFTS) MANIPULATIONS. THIS IS THE ONLY PART OF THE MICROPROCESSOR WHICH ALTERS DATA.
- COMMAND: THE MONITOR FUNCTIONS M,G,P,R,L,S,↑,↓.
- DATA: INFORMATION FOR THE PROCESSOR THAT DOES NOT HAVE TO BE TRANSLATED. e.g. "AD" AS DATA ACTUALLY MEANS $10x16+13x1 = 173_{10}$ WHEREAS THE INSTRUCTION "AD" GETS TRANSLATED INTO THE OPERATION "LOAD ACCUMULATOR ABSOLUTE".
- EPROM: ERASABLE PROGRAMMABLE READ ONLY MEMORY. THIS TYPE OF MEMORY IS LIKE A PROM, BUT CAN AGAIN BE ERASED BY EXPOSING THE CHIP TO ULTRAVIOLET LIGHT.
- FLAGS: ONE BIT INTERNAL REGISTERS. ALL SEVEN FLAGS CAN ALSO BE TREATED AS SEPARATE BITS OF THE P REGISTER (PROCESSOR STATUS).
- INDEX REGISTER: A REGISTER WHICH CAN BE USED TO MODIFY AN ADDRESS (USED IN REFERRING TO DATA) BY BEING ADDED TO IT, THUS ACCESSING A CERTAIN ELEMENT OF A TABLE. THE 6502 HAS TWO INDEX REGISTERS CALLED X & Y.
- INSTRUCTION: A FUNCTION OF THE MICROPROCESSOR LIKE LOAD AND STORE.
- I/0: INPUT/OUTPUT. THIS CHIP ALLOWS YOU TO COMMUNICATE WITH THE OUTSIDE WORLD. IN THE ACORN THE I/0 CHIP HAS 16 PROGRAMMABLE LINES WHICH CAN EITHER BE OUTPUTS OR INPUTS. IT ALSO HAS 128 BYTES OF RAM.
- IRQ: INTERRUPT REQUEST. IF FLAG I (INTERRUPT DISABLE) IS CLEAR AND A REQUEST IS MADE THE PROCESSOR WILL ATTEND TO IT AFTER SETTING FLAG I AND STORING THE PROGRAM COUNTER AND STATUS REGISTER.
- JUMP: THE PROGRAM COUNTER IS LOADED WITH A NEW ADDRESS. THE EXECUTION OF THE PROGRAM, WHICH IS NORMALLY USING CONSECUTIVE ADDRESSES, CONTINUES (JUMPS) AT THIS NEW ADDRESS.
- LOAD: TRANSFERS THE DATA OF A MEMORY LOCATION TO AN INTERNAL REGISTER.
- MNEMONIC: SUGGESTIVE ABBREVIATION OF AN INSTRUCTION e.g. THE INSTRUCTION "LOAD ACCUMULATOR ABSOLUTE" HAS THE MNEMONIC "LDA".
- NMI: NON MASKABLE INTERRUPT WHEN THE NON MASKABLE INTERRUPT IS ACTIVATED THE PROCESS WILL SET FLAG I, STORE AWAY ITS PROGRAM COUNTER AND STATUS REGISTER AND THEN IMMEDIATELY ATTEND TO THE INTERRUPT. THERE IS NO WAY OF PREVENTING THIS INTERRUPT. IT HAS PRIORITY OVER IRQ.
- OPCODE: HEXADECIMAL REPRESENTATION OF AN INSTRUCTION. e.g. THE INSTRUCTION "LOAD ACCUMULATOR ABSOLUTE" HAS THE MNEMONIC "LDA" AND THE OPCODE "AD".

ØØ1Ø	D,R4	BASE ADDRESS OF THE EIGHT DISPLAYED MEMORY LOCATIONS, REGISTER 4: TEMPORARILY PCH AFTER BREAK.
ØØ11	R5	REGISTER 5: TEMPORARILY PCL AFTER BREAK
ØØ12	R6	REGISTER 6: TEMPORARILY Ø1 AFTER BREAK
ØØ13	R7	REGISTER 7: TEMPORARILY S AFTER BREAK.
ØØ14-ØØ17		LAST 4 DISPLAYED MEMORY LOCATIONS.
ØØ18	P	SINGLE LEVEL OF STORAGE FOR PREVIOUS DATA AT BREAK POINTS.
ØØ19	COL	COLUMN OF KEY CURRENTLY BEING PROCESSED
ØØ1A	TX,TY	TEMPORARY STORAGE FOR X (IN DISPLAY) OR Y (VARIOUS PLACES).
ØØ1C,ØØ1D	USERNMI	ADDRESS OF USER'S NMI PROGRAM
ØØ1E,ØØ1F	USERIRQ	ADDRESS OF USER'S IRQ PROGRAM
ØØ1B	RECAL	CONTAINS PC RECALCULATION FACTOR FOR BREAK
FEØØ	QUAD	DISPLAY X–3,X–2,X–1,X ON THE DISPLAY; THEN ↓
FEØC	DISPLAY	STROBE KEYBOARD, MULTIPLEX DISPLAY, RETURN WITH KEY INFORMATION
FE5E	MHEXTD	DISPLAY A MEMORY BYTE ON RIGHT OF DISPLAY
FE6Ø	RDHEXTD	DISPLAY A ON RIGHT OF DISPLAY
FE64	QHEXTD1	DISPLAY X & X+1 ON DISPLAYS 1,2,3 & 4
FE66	QHEXTD2	DISPLAY X & X+1 ON DISPLAYS Y–2,Y–1, Y & Y+1
FE6F	DHEXTD	DISPLAY A ON DISPLAYS Y & Y+1
FE7A	HEXTD	DISPLAY BOTTOM 4 BITS OF A ON DISPLAY Y
FE88	QDATFET	FETCH AN ADDRESS INTO LOCATIONS X & X+1
FEAØ	COM16	INCREMENT & COMPARE TWO 16 BIT NOS X+6,X+7 & X+8,X+9
FEA6	NOINC	COMPARE X+6,X+7 & X+8,X+9 FOR EQUALITY
FEB1	PUTBYTE	A TO TAPE, DO 1 START & 1 STOP BITS, NO PARITY
FECD	WAIT	WAIT FOR CASSETTE TIMING
FEDØ	½ WAIT	HALF THE WAIT
FEDD	GETBYTE	TAPE TO A, WAIT FOR START BIT, CENTRE TIMING
FEF3	RESET	ENTRY TO MONITOR
FFØ4	RESTART	RE-ENTRY TO RUNNING MONITOR
FFB3	BREAK	ENTRY TO MONITOR FROM BRK INSTRUCTION, DISPLAY CPU
FFEA	FONT	SEVEN SEGMENT PICTURES OF THE HEX DIGITS
ØØ1B	RECAL	CONTAINS PC RECALCULATION FACTOR FOR BREAK

ØØØØ11ØØ$_2$	ØC$_{16}$	12$_{10}$
1111ØØ11$_2$ +	F3$_{16}$ +	243$_{10}$ +
11111111$_2$	FF$_{16}$	255$_{10}$

WHICH DOESN'T HAPPEN .... UNTIL WE ADD AN EXTRA 1:

ØØØØ11ØØ$_2$	ØC$_{16}$	12$_{10}$
1111ØØ11$_2$	F3$_{16}$	243$_{10}$
1$_2$ +	1$_{16}$ +	1$_{10}$ 16
1 ØØØØØØØØ$_2$	1ØØ$_{16}$	256$_{10}$

AND THEN TREAT THE OUTPUT CARRY AS INDICATING THE ABSENCE OF A BORROW FROM THE HIGHER ORDERS.
THE NUMBER (ONE'S-COMPLEMENT + 1) IS CALLED THE TWO'S-COMPLEMENT OF A NUMBER:

BINARY	HEXADECIMAL	DECIMAL
ØØØØØØØ1$_2$	Ø1$_{16}$	+1$_{10}$
ØØØØØØØØ$_2$	ØØ$_{16}$	+Ø$_{10}$ or –Ø$_{10}$
11111111$_2$	FF$_{16}$	–1$_{10}$
111111Ø2	FE$_{16}$	–2$_{10}$
:	:	:
1111Ø1ØØ$_2$	F4$_{16}$	–12$_{10}$
1ØØØØØØØ$_2$	8Ø$_{16}$	–128$_{10}$
Ø1111111$_2$	7F$_{16}$	+127$_{10}$

SO A BYTE CAN BE TREATED AS A 'SIGNED BINARY NUMBER' IN THE RANGE +127..... Ø..... –128, OR AS A BINARY NUMBER IN THE RANGE Ø.....+255. NOW THE SUBTRACTION ABOVE SHOULD BE CLEAR : INTERNALLY, THE MICRO-PROCESSOR ONE'S-COMPLEMENTS ONE OF THE NUMBERS AND THEN EXECUTES A NORMAL ADDITON WITH CARRY.

1.4 BINARY CODED DECIMAL (BCD) ARITHMETIC

99$_{16}$ LOOKS VERY LIKE 99$_{10}$ THEY BEHAVE THE SAME WAY AS THEY ARE MOVED AROUND AND UNDERGO LOGICAL OPERATIONS SINCE THEY ARE WRITTEN THE SAME WAY. THE BINARY REPRESENTATION OF 99$_{10}$ WOULD NORMALLY BE Ø11ØØØ11$_2$, AND OF 99$_{16}$ IT WOULD BE 1ØØ11ØØ1$_1$. WE NOW DEFINE THE BINARY CODED DECIMAL VERSION OF 99$_{10}$ AS BEING THE BINARY REPRESENTATION OF THE DECIMAL DIGITS IN THE ORIGINAL POSITIONAL NOTATION, MAKING THE DIFFERENCE BETWEEN THE BINARY REPRESENTATIONS OF 99$_{16}$ & 99$_{10}$ A MATTER OF SUPSCRIPTS:

99$_{16}$ = 1ØØ11ØØ1$_2$
99$_{10}$ = 1ØØ11ØØ1 B.C.D.

THE B.C.D. AND BINARY NUMBERS DIFFER IN HANDLING ONLY IN ARITHMETIC:

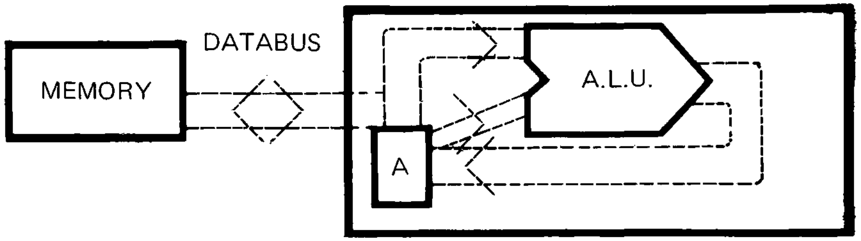
79$_{16}$			79$_{10}$	
22$_{16}$	+	BUT	22$_{10}$	+
9B$_{16}$			1Ø1$_{10}$	

THE MICROPROCESSOR CAN BE 'TOLD' WHICH TYPE OF ARITHMETIC TO CARRY OUT, BY SETTING (PUTTING A ONE INTO) OR CLEARING (PUTTING A ZERO INTO) AN INTERNAL BIT, THE 'DECIMAL MODE' FLAG.

CHAPTER 2: WELCOME TO THE MACHINE

2.1 HOW ACORN'S MICROPROCESSOR WORKS

TO CARRY OUT THE ABOVE OPERATIONS THE MICROPROCESSOR HAS AN INTERNAL ARITHMETIC LOGIC UNIT (A.L.U.) WHOSE OUTPUT IS SENT TO AN INTERNAL REGISTER OF ONE BYTE LENGTH CALLED THE ACCUMULATOR 'A', THIS REGISTER ALSO ACTS AS ONE OF THE OPERANDS; THE OTHER BEING DRAWN FROM THE MEMORY EXTERNAL TO THE μPROCESSOR, WHICH IS CONNECTED TO THE μP BY 8 LINES CALLED THE DATABUS:



DATA CAN BE TRANSFERRED ALONG THE DATABUS IN EITHER DIRECTION, THIS DIRECTION IS CHOSEN BY THE, μP AND INDICATED TO THE EXTERNAL UNITS BY A SINGLE 'R/W' LINE : WHEN HIGH, '1' , THE μP IS RECEIVING DATA FROM THE MEMORY, 'READING'; WHEN LOW, 'Ø', THE μP IS SENDING DATA TO THE MEMORY, 'WRITING'. ALL INFORMATION USED BY THE μP TRAVELS ALONG THE DATABUS, INCLUDING THE INSTRUCTIONS. SO THAT THE μP KNOWS WHERE ITS INSTRUCTIONS ARE IT HAS A TWO BYTE (16_{10} BIT) REGISTER CALLED THE PROGRAM COUNTER, 'PC', WHICH POINTS AT THE INSTRUCTIONS BEING EXECUTED. THE MEMORY CAN BE VIEWED AS A BOOK OF 256 PAGES, THE PARTICULAR PAGE BEING DECIDED BY THE MOST SIGNIFICANT 8 BITS (BITS 15–8) OF THE 16 BIT ADDRESS, EACH PAGE CONTAINING 256 BYTES, THE PARTICULAR BYTE BEING DECIDED BY THE LEAST SIGNIFICANT 8 BITS (BITS 7–Ø) OF THE 16 BIT ADDRESS.

IN THE KIT, PAGES FE_{16} & FF_{16} ARE OCCUPIED BY A NON-ERASEABLE PROGRAM TO INTERFACE BETWEEN THE MICROPROCESSOR AND THE KEYBOARD & DISPLAY UNIT. TO START THE μP IN THIS PROGRAM (AT THE CORRECT PLACE) THERE IS A RESET BUTTON WHICH INITIALIZES THE PROGRAM COUNTER. IN PAGE $ØØ_{16}$ THERE IS SOME ALTERABLE MEMORY, OF WHICH THE BOTTOM IF_{16} BYTES ARE GIVEN SPECIAL USES BY THE FE_{16} & FF_{16} MONITOR PROGRAM, SO, UNLESS PRESSED FOR SPACE, IT'S BEST TO STAY OUT OF THEM.

APPENDIX C HEXADECIMAL TO DECIMAL

1st DIGIT / 2nd DIGIT

	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Ø	Ø	1	2	3	4	5	6	7	8	9	1Ø	11	12	13	14	15
1	16	17	18	19	2Ø	21	22	23	24	25	26	27	28	29	3Ø	31
2	32	33	34	35	36	37	38	39	4Ø	41	42	43	44	45	46	47
3	48	49	5Ø	51	52	53	54	55	56	57	58	59	6Ø	61	62	63
4	64	65	66	67	68	69	7Ø	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	9Ø	91	92	93	94	95
6	96	97	98	99	1ØØ	1Ø1	1Ø2	1Ø3	1Ø4	1Ø5	1Ø6	1Ø7	1Ø8	1Ø9	11Ø	111
7	112	113	114	115	116	117	118	119	12Ø	121	122	123	124	125	126	127
8	128	129	13Ø	131	132	133	134	135	136	137	138	139	14Ø	141	142	143
9	144	145	146	147	148	149	15Ø	151	152	153	154	155	156	157	158	159
A	16Ø	161	162	163	164	165	166	167	168	169	17Ø	171	172	173	174	175
B	176	177	178	179	18Ø	181	182	183	184	185	186	187	188	189	19Ø	191
C	192	193	194	195	196	197	198	199	2ØØ	2Ø1	2Ø2	2Ø3	2Ø4	2Ø5	2Ø6	2Ø7
D	2Ø8	2Ø9	21Ø	211	212	213	214	215	216	217	218	219	22Ø	221	222	223
E	224	225	226	227	228	229	23Ø	231	232	233	234	235	236	237	238	239
F	24Ø	241	242	243	244	245	246	247	248	249	25Ø	251	252	253	254	255

HEX	DEC
1ØØ	256
2ØØ	512
4ØØ	1Ø24
8ØØ	2Ø48
1ØØØ	4Ø96
2ØØØ	8192
4ØØØ	16384
8ØØØ	32768
1ØØØØ	65536

APPENDIX D ACORN MONITOR ADDRESS INFORMATION

ADDRESS	LABEL	COMMENT
ØØØØ,ØØØ1	MAP	LOW AND HIGH BYTES OF THE M ADDRESS
ØØØ2,ØØØ3	GAP	LOW AND HIGH BYTES OF THE GO ADDRESS
ØØØ4,ØØØ5	PAP	LOW AND HIGH BYTES OF THE BREAKPOINT ADDRESS
ØØØ6,ØØØ7	FAP	LOW AND HIGH BYTES OF THE TAPE FROM ADDRESS
ØØØ8,ØØØ9	TAP	LOW AND HIGH BYTES OF THE TAPE TO ADDRESS
ØØØA	RØ	REGISTER Ø: CONTAINS A AFTER BREAK.
ØØØB	R1	REGISTER 1: CONTAINS X AFTER BREAK.
ØØØC	R2	REGISTER 2: CONTAINS Y AFTER BREAK.
ØØØD	R3, KEY	REGISTER 3: TEMPORARILY P AFTER BREAK, CONTAINS LAST PRESSED KEY FOR DISPLAY
ØØØE	REPEAT	MSB=1 SETS REPEATEDLY SCANNED DISPLAY, OTHERWISE SINGLE SCAN.
ØØØF	EXEC	EXECUTION STATUS OF THE KEY PROCESSING ROUTINE

DISSASSEMBLY CHART

1ST DIGIT \ 2ND DIGIT	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	BRK	ORD (I,X)				ORA ZERO	ASL ZERO		PHP	ORA IMMED	ASLA			ORA ABSOLUTE	ASL ABSOLUTE	
1	BPL	ORD (I),Y				ORD Z,X	ASL Z,X		CLC	ORA A,Y				ORA A,X	ASL A,X	
2		AND (I,X)			BIT ZERO	AND ZERO	ROL ZERO		PLP	AND IMMED	ROLA		BIT ABSOLUTE	AND ABSOLUTE	ROL ABSOLUTE	
3	BMI	AND (I),Y				AND Z,X	ROL Z,X			AND A,Y				AND A,X	ROL A,X	
4	RTI	EOR (I,X)				EOR ZERO	LSR ZERO		PHA	EOR IMMED	LSRA		JMP ABSOLUTE	EOR ABSOLUTE	LSR ABSOLUTE	
5	BVC	EOR (I),Y				EOR Z,X	LSR Z,X		CLI	EOR A,Y				EOR A,X	LSR A,X	
6	RTS	ADC (I,X)				ADC ZERO	FOR ZERO		PLA	ADC IMMED	RORA		JMP INDIRECT	ADC ABS	ROR ABSOLUTE	
7	BVS	ADC (I),Y				ADC Z,X	ROR Z,X			ADC A,Y				ADC A,X	ROR A,X	
8		STA (I,X)			STY ZERO	STA ZERO	STX ZERO		DEY	STA A,Y	TXA		STY ABSOLUTE	STA ABSOLUTE	STX ABSOLUTE	
9	BCC	STA (I),Y			STY Z,X	STA Z,X	STX Z,Y		TYA		TXS			STA A,X		
A	LDY IMMED	LDA (I,X)	LDX IMMED		LDY ZERO	LDA ZERO	LDX ZERO		TAY	LDA IMMED	TAX		LDY ABSOLUTE	LDA ABSOLUTE	LDX ABSOLUTE	
B	BCS	LDA (I),Y			LDY Z,X	LDA Z,X	LDX Z,Y		CLV	LDA A,Y	TSX		LDY A,X	LDA A,X	LDX A,Y	
C	CPY IMMED	CMP (I,X)			CPY ZERO	CMP ZERO	DEC ZERO		INY	CMP IMMED	DEX		CPY ABSOLUTE	CMP ABSOLUTE	DEC ABSOLUTE	
D	BNE	CMP (I),Y				CMP Z,X	DEC Z,X		CLD	CMP A,Y				CMP A,X	DEC A,X	
E	CPX IMMED	SBC (I,X)			CPX ZERO	SBC ZERO	INC ZERO		INX	SBC IMMED	NOP		CPX ABSOLUTE	SBC ABSOLUTE	INC ABSOLUTE	
F	BEQ	SBC (I),Y				SBC Z,X	INC Z,X			SBC A,Y				SBC A,X	INC A,X	

2.2 THE MONITOR COMMANDS M,↑,↓

THE FIRST FEATURE OF THE MONITOR IS THE MEMORY INSPECT & MODIFY CONTROL SWITCH ON, AND PRESS THE RESET BUTTON:

MODE	ADDRESS		DATA
.	....	.	..

THEN PRESS THE MODIFY KEY, M. THIS GETS YOU INTO THE MEMORY INSPECTION AND MODIFY MODE. THE MODE INDICATOR SHOWS 'A' FOR ALTER. THIS FIRST PHASE OF 'A' ALLOWS YOU TO CHOOSE ANY ADDRESS IN MEMORY.

A. XXXX .

APPEARS ON THE DISPLAY, WHERE X REPRESENTS ANY OF THE 16 HEX CARACTERS SIGNIFYING THE ADDRESS.NOW PRESS THE KEYS F, E, Ø, Ø (IF YOU MAKE A MISTAKE, E.G. PRESSED F, D, JUST START OFF FROM THE F AGAIN). AS EACH KEY IS PRESSED THE INFORMATION ON THE DISPLAY SHIFTS TO THE LEFT:

A. XXXF .
A. XXFE .
A. XFEØ .
A. FEØØ .

AND SO YOU END UP WITH FEØØ ON THE DISPLAY. PRESS ANY OF THE EIGHT COMMAND KEYS (IT DOES NOT MATTER WHICH) AND YOU CAN INSPECT THE CONTENTS OF THIS MEMORY ADDRESS. THIS IS PHASE TWO OF MODE 'A' AND ALLOWS YOU TO INSPECT AND ALTER THE DATA OF THE MEMORY ADDRESS CHOSEN IN PHASE ONE.

A. FEØØ . AØ

THIS IS THE INFORMATION STORED AT THE VERY BEGINNING OF THE MONITOR. IF YOU PRESS THE ↑ KEY

A. FEØ1 . Ø6

UP WE GO. NATURALLY THE ↓ KEY BRINGS BACK

A. FEØØ . AØ

AND EITHER KEY MAY BE USED ANY NUMBER OF TIMES IN SUCCESSION. NOW, IF, WITHOUT TURNING OFF, YOU PRESS RESET

. ...

AND THEN M

A. FEØØ .

THE SYSTEM HAS REMEMBERED THE ADDRESS YOU WERE USING (WHICH DOESN'T HAVE TO BE FEØØ) TO INSPECT MEMORY NOW ENTER THE ADDRESS ØØ3Ø AND TERMINATE WITH ANY COMMAND KEY

A. ØØ3Ø . XX

ØØ3Ø IS AN ADDRESS IN THE ALTERABLE SECTION OF THE MEMORY. PRESSING DIGIT KEYS NOW WILL CAUSE THE INFORMATION IN ØØ3Ø TO CHANGE (WHAT HAPPENS AT FEØØ?? TRY IT! YOU CANNOT WRITE INTO THE MONITOR PROM, (i.e. THE PROGRAMMABLE READ ONLY MEMORY). PRESS Ø, 1.

A. ØØ3Ø . Ø1

PRESS 2,3

A. ØØ3Ø . 23

AS BEFORE INFORMATION IS SHIFTED IN UNTIL TERMINATED BY ANY COMMAND KEY. BUT, UNLIKE THE ADDRESS FETCHING PHASE, THE COMMAND KEY WILL BE EXECUTED. USEFUL TERMINATORS ARE THE M, ↑ & ↓ KEYS. PRESS ↑.

A. ØØ31 . XX

PRESS 4,5

A. ØØ31 . 45

PRESS ↓

A. ØØ3Ø . 23

& ↑ AGAIN

A ØØ31 . 45

YOU CAN GO UP AND DOWN INSPECTING & MODIFYING THE MEMORY CONTENTS IF THERE IS NO ALTERABLE MEMORY (E.G. A PROM) AT A PARTICULAR ADDRESS, THE INFORMATION WILL NOT CHANGE. TO CLOSE THIS SECTION WE'LL MAKE THE MONITOR DO A LITTLE TRICK. M,Ø,Ø,Ø,E, k
(k ≡ ANY COMMAND KEY)

IV REGISTERS OTHER THAN ACCUMULATOR

MNEMONIC	VERBAL	FLAGS	IMMED	ZERO	Z,X	Z,Y	ABSOLUTE	A,X	A,Y	INDIRECT	
ADDRESSING MODE (BYTES / TIME)			2 / 2	2 / 3	2 / 4	2 / 4	3 / 4	3 / 4+	3 / 4+	3 / 5	BYTES / TIME
BIT	LOGICAL AND WITH MASK AND TEST BITS	NZV	–	24	–	–	2C	–	–	–	$A \wedge M \rightarrow Z, M_6 \rightarrow V, M_7 \rightarrow N$
CPX	COMPARE X	NZC	EØ	E4	–	–	EC	–	–	–	X – M
CPY	COMPARE Y	NZC	CØ	C4	–	–	CC	–	–	–	Y – M
JMP	JUMP	–	–	–	–	–	4C	–	–	6C	M → PCL M + 1 → PCH
LDX	LOAD X REGISTER	NZ	A2	A6	–	B6	AE	–	BE	–	M → X
LDY	LOAD Y REGISTER	NZ	AØ	A4	B4	–	AC	BC	–	–	M → Y
STX	STORE X REGISTER		–	86	–	96	8E	–	–	–	X → M
STY	STORE Y REGISTER		–	84	94	–	8C	–	–	–	Y → M

V READ – MODIFY – WRITE

MNEMONIC	VERBAL	FLAGS	ZERO	Z,X	ABS	A,X	
ADDRESSING MODE (BYTES / TIME)			2 / 5	2 / 6	3 / 6	3 / 7	BYTES / TIME
ASL	ARITHMETIC SHIFT LEFT	NZC	Ø6	16	ØE	1E	C ← [A] ← Ø
DEC	BINARY DECREMENT	NZ	C6	D6	CE	DE	M – 1 → M
INC	BINARY INCREMENT	NZ	E6	F6	EE	FE	M + 1 → M
LSR	LOGICAL SHIFT RIGHT	NZC	46	56	4E	5E	Ø → [M] → C (Ø → N)
ROL	ROTATE LEFT	NZC	26	36	2E	3E	C ← [M] ← C
ROR	ROTATE RIGHT	NZC	66	76	6E	7E	C → [M] → C

+ : EXTRA CYCLE IF OPERATION INVOLVES PAGE CROSSING
+t: + Ø IF NO BRANCH; + 1 IF BRANCH; + 2 IF BRANCH & CROSS PAGE

III IMPLIED. SINGLE BYTE WITH NO ADDRESS MODE

MNEMORIC	VERBAL	FLAGS		TIME		
ASLA	ARITHMETIC SHIFT LEFT A	NZC	ØA	2	C ← [A] ← Ø BINARY X 2	ALSO KNOWN AS ACCUMULATOR ADDRESS MODE
BRK	BREAK	B	ØØ	7	1 → B then on IRQ	ALSO KNOWN AS 'SOFTWARE INTERRUPT'
CLC	CLEAR CARRY FLAG	C	18	2	Ø → C	
CLD	CLEAR DECIMAL MODE	D	D8	2	Ø → D	– OPERATE IN BINARY ARITH.
CLI	CLEAR INTERRUPT DISABLE	I	58	2	Ø → I	– ALLOWS INTERRUPTS
CLV	CLEAR OVERFLOW	V	B8	2	Ø → V	
DEX	BINARY DEC X	NZ	CA	2	X – 1 → X	
DEY	BINARY DEC Y	NZ	88	2	Y – 1 → Y	
INX	BINARY INC X	NZ	E8	2	X + 1 → X	
INY	BINARY INC Y	NZ	C8	2	Y + 1 → Y	
LSRA	LOGICAL SHIFT RIGHT A	NZC	4A	2	C Ø→ [A] (Ø→N) BINARY ÷ 2	– ACCUMULATOR ADDRESSING
NOP	NO OPERATION		EA	2		
PHA	PUSH A		48	3	A → Ø1ØØ + S ; S – 1 → S	
PHP	PUSH PROCESSOR STATUS		Ø8	3	P → Ø1ØØ + S ; S – 1 → S	
PLA	PULL A	NZ	68	4	S + 1 → S Ø1ØØ + S → A	
PLP	PULL PROCESSOR STATUS	ALL	28	4	S + 1 → S Ø1ØØ + S → P	
ROLA	ROTATE LEFT A	NZC	2A	2	C ← [A] ←	–'ACCUMULATOR ADDRESSING'
RORA	ROTATE RIGHT A	NZC	6A	2	C → [A]	–'ACCUMULATOR ADDRESSING'
RTI	RETURN FROM INTERRUPT	ALL	4Ø	6	S + 1 → S Ø1ØØ + S → P S + 1 → S Ø1ØØ + S → PCL S + 1 → S Ø1ØØ + S → PCH	
RTS	RETURN FROM SUBROUTINE		6Ø	6	S + 1 → S Ø1ØØ + S → PCL S + 1 → S Ø1ØØ + S → PCH	–OPPORTUNITY FOR A PHP ON ENTRY & AN RTI ON EXIT
TAX	TRANSFER A TO X	NZ	AA	2	A → X	
TAY	TRANSFER A TO Y	NZ	A8	2	A → Y	
TSX	TRANSFER S TO X	NZ	BA	2	S → X	
TXA	TRANSFER X TO A	NZ	8A	2	X → A	
TXS	TRANSFER X TO S	NZ	9A	2	X → S	
TYA	TRANSFER Y TO A	NZ	98	2	Y → A	

PRESS 1,6. (IF YOU GET BORED, YOU CAN GO THE OTHER WAY BY 1,7) (ESCAPE BY RESET). THE MONITOR SCANS THROUGH ALL MEMORY SUCCESSIVELY SHOWING ITS CONTENTS (DATA). WHERE THERE IS NO MEMORY AT ALL YOU WILL PROBABLY SEE THE FIRST TWO ADDRESS DIGITS.

2.3 AT LAST, A PROGRAM

2.3.1 ASSEMBLY LANGUAGE, MACHINE LANGUAGE, THE INSTRUCTIONS LOAD, STORE AND JUMP

A PROGRAM IS THE NAME FOR A SET OF STORED COMMANDS THAT THE MICROPROCESSOR WILL EXECUTE. THESE ARE STORED IN BINARY, SINCE THAT'S ALL THAT ANYTHING CAN BE STORED IN, (ENTERED BY YOU IN HEX) AND ARE INDISTINGUISHABLE FROM ANYTHING ELSE. IF IT GETS THE CHANCE THE μP (MICROPROCESSOR) WILL BUSY ITSELF TREATING INFORMATION WHICH YOU MEANT AS DATA AS A PROGRAM. IT PROBABLY WON'T BE DOING ANYTHING INTELLIGENT AND WILL HAVE TO BE SUMMONED BACK WITH THE RESET KEY. SOME SORT OF TRANSLATION BETWEEN THE STORED BINARY/HEX AND YOU IS NEEDED. $1\emptyset1\emptyset11\emptyset1_2$ MEANS A GREAT DEAL TO THE μP BUT LITTLE TO YOU. IT ACTUALLY MEANS "LOAD THE ACCUMULATOR WITH THE CONTENTS OF THE MEMORY ADDRESS DEFINED BY THE FOLLOWING TWO BYTES, OF WHICH THE FIRST IS THE LEAST SIGNIFICANT ADDRESS". THIS IS A LITTLE LONG FOR WRITING STRAIGHT INTO A PROGRAM AND IS USUALLY ABBREVIATED TO LDA ABS, OR JUST LDA.ABSOLUTE MEANS ANYWHERE IN THE 64K. THE 6502 CAN ADDRESS 64K OF MEMORY WHICH IS DIVIDED INTO PAGES 256 BYTES LONG THE FIRST PAGE IS CALLED ZERO PAGE. LOCATIONS IN ZERO PAGE CAN BE ADDRESSED BY JUST ONE BYTE. THERE ARE SPECIAL INSTRUCTIONS TO DO THIS. AT THE END OF THE MANUAL THERE IS A LIST OF ALL THESE MNEMONICS WITH THEIR HEX EQUIVALENTS IN APPENDIX B. SO IF WE WROTE THE PROGRAM IN MNEMOMICS IT WOULD LOOK LIKE.

LDA FE ØØ

AND WE WOULD TRANSLATE IT FOR THE μP AS THE THREE BYTES

AD	LOAD ABSOLUTE
ØØ	LOWER BYTE OF ADDRESS
FE	HIGH BYTE OF ADDRESS

WHICH WOULD CAUSE THE μP TO PUT AØ (THE DATA STORED IN FEØØ) IN ITS ACCUMULATOR (REMEMBER USING THE MONITOR TO LOOK AT FEØØ?). THE TRANSLATION PROCESS IS CALLED ASSEMBLING AND COMPUTER PROGRAMS WHICH DO IT ARE CALLED ASSEMBLERS. A RESIDENT ASSEMBLER IS ONE THAT RUNS (OPERATES) ON THE SAME MACHINE THAT IT ASSEMBLES FOR; A CROSS ASSEMBLER RUNS ON A DIFFERENT MACHINE. THE MNEMONICS LDA, STA etc ARE OFTEN CALLED ASSEMBLY LANGUAGE, THE GENERATED BINARY IS CALLED MACHINE CODE.

WE CAN LOAD THE ACCUMULATOR IN TEN OTHER WAYS; HERE ARE TWO OF THEM.

INSTRUCTION LENGTH IN BYTES	TYPE	HEX	MNEMONIC	EXECUTION TIME,µS	BRIEF EXPLANATION
2	1	A9	LDA #	2	PUT THE NEXT BYTE IN ACCUMULATOR. "LOAD IMMEDIATE'.
2	2	A5	LDA Ƶ	3	SHORTENED FORM OF LOAD ABS ØØXX 'LOAD ZERO PAGE'.
3	3	AD	LDA	4	LOAD A ABSOLUTE.

THE FIRST OF THESE INSTRUCTIONS IS VERY IMPORTANT. IF WE KNOW THAT WE WANT AØ IN THE ACCUMULATOR THEN IT IS WASTEFUL TO FIND A MEMORY LOCATION WHICH HAPPENS TO CONTAIN IT, SINCE TWO BYTES ARE NEEDED (GENERALLY) TO SPECIFY WHERE IT IS AND SO WE IMPLY, BY THE IMMEDIATE INSTRUCTION, WHERE IT IS & ACTUALLY ENTER IT IN THE PROGRAM. THERE ARE COMPLEMENTARY STORE ACCUMULATOR 'STA' INSTRUCTIONS TO LDA Ƶ AND LDA.

BYTES	TYPE	HEX	MNEMONIC	TIME µS	
2	2	85	STA Ƶ	2	STORE A ZERO PAGE (IN THE FIRST 256 BYTES)
3	3	8D	STA	3	STORE A ABSOLUTE (ANYWHERE IN MEMORY)

WE CAN ALSO LOAD THE PROGRAM COUNTER. THE PROGRAM COUNTER IS AN INTERNAL REGISTER THAT POINTS TO THE NEXT LINE OF THE PROGRAM. THE MNEMONIC FOR THIS IS NOT LDPC BECAUSE WHEN THE P.C. IS LOADED WITH A NEW VALUE IT GIVES THE MICROPROCESSOR A DIFFERENT PLACE TO LOOK FOR INSTRUCTIONS: THE PROGRAM JUMPS. SO 'LOAD P.C. WITH NEXT TWO BYTES' (LDPC) IS JMP, THIS IS REFERRED TO AS JUMP ABSOLUTE SINCE THE PROGRAM JUMPS TO A NEW ABSOLUTE ADDRESS. SO IF WE ARE NOT IN THE MONITOR AND WANT TO BE, JMP FFØ4 WILL ENTER THE MONITOR. NOW WHAT HAPPENS IF THE FOLLOWING PROGRAM IS RUN?

```
LDA      FEØØ
STA  Ƶ    2Ø
JMP      FFØ4
```

THE FIRST INSTRUCTION GETS THE CONTENTS OF FEØØ, AND PUTS IT IN THE ACCUMULATOR. THE SECOND STORES THE ACCUMULATOR IN LOCATION ØØ2Ø.THE FIRST TWO Ø'S REFER TO ZERO PAGE AND ARE ASSUMED BY THE PROCESSOR IN THE ZERO PAGE MODE. THE THIRD GETS BACK TO THE MONITOR, SO THAT YOU CAN INSPECT LOCATION 2Ø. THIS READS AS.

```
ØØ3Ø   AD (OPCODE)    LDA FEØØ
ØØ31   ØØ (DATA)
ØØ32   FE (DATA)
ØØ33   85 (OPCODE)    STA Ƶ 2Ø
ØØ34   2Ø (DATA)
ØØ35   4C (OPCODE)    JMP FFØ4
ØØ36   Ø4 (DATA)
ØØ37   FF (DATA)
```

II RELATIVE: RELATIVE ADDRESSING MODE

2 BYTES 2+† CYCLES

MNEMONIC	VERBAL		
BCC	BRANCH IF CARRY CLEAR	9Ø	BRANCH IF C = Ø
BCS	BRANCH IF CARRY SET	BØ	C = 1
BEQ	BRANCH IF EQUAL (TO ZERO)	FØ	Ƶ = 1
BMI	BRANCH IF MINUS	3Ø	N = 1
BNE	BRANCH IF NOT EQUAL	DØ	Ƶ = Ø
BPL	BRANCH IF PLUS	1Ø	N = Ø
BVC	BRANCH IF OVERFLOW CLEAR	5Ø	V = Ø
BVS	BRANCH IF OVERFLOW SET	7Ø	V = 1

APPENDIX B INSTRUCTION SET

I ACCUMULATOR REFERENCE: ACCUMULATOR, OPERATION, MEMORY → ACCUMULATOR

MNEMONIC	VERBAL	ADDRESSING MODE	IMMED	ZERO	Z,X	ABSOLUTE	A,X	A,Y	(I,X)	(I),Y	
		FLAGS IN P AFFECTED	2 2	2 3	2 4	3 4	3 4+	3 4+	2 6	2 5+	BYTES CYCLES = SPEED µS
ADC	ADD WITH CARRY	NƵCV	69	65	75	6D	7D	79	61	71	A + M + C → A
AND	LOGICAL AND	NƵ	29	25	35	2D	3D	39	21	31	A → M → A
CMP	LOGICAL COMPARE	NƵC	C9	C5	D5	CD	DD	D9	C1	D1	A – M
EOR	LOGICAL EXCLUSIVE OR	NƵ	49	45	55	4D	5D	59	41	51	A V M → A
LDA	LOAD ACCUMULATOR	NƵ	A9	A5	B5	AD	BD	B9	A1	B1	M → A
ORA	LOGICAL OR	NƵ	Ø9	Ø5	15	ØD	1D	19	Ø1	11	A V M → A
SBC	SUBTRACT WITH BORROW/CARRY	NƵCV	E9	E5	F5	ED	FD	F9	E1	F1	A – M + C → A
STA	STORE ACCUMULATOR		–	85	95	8D	9D	99	81	91	A → M

THE ADDRESS ØØ3Ø IS THE STARTING ADDRESS OF THE PROGRAM. THIS PARTICULAR PROGRAM WILL WORK WITH ANY STARTING ADDRESS – IT IS SAID TO BE 'POSITION INDEPENDENT' OR 'RELOCATABLE' – BUT OTHER PROGRAMS MAY NOT. IF YOU ARE NEW TO THE GAME, IT WILL BE EASIER IF YOU ENTER PROGRAMS AT THE STARTING ADDRESS SHOWN IN THE MANUAL.

2.3.2 ENTERING A PROGRAM, THE GO COMMAND

TO ENTER THIS PROGRAM, WE'LL GO THROUGH IT STEP BY STEP.

I ENTER THE STARTING ADDRESS: PRESS M,Ø.Ø,3,Ø, k

II ENTER A BYTE OF DATA A,D

III USE THE ↑ KEY TO TERMINATE DATA ENTRY AND STEP UP
– CONTINUE WITH Ø,Ø,↑,F,E,↑,8,5,↑,2,Ø,↑,4,C,↑,Ø,4,↑,F,F

IV CHECK THAT THE PROGRAM IS ENTERED CORRECTLY BY, E.G, USING ↓ TO GO BACK DOWN THROUGH IT.
– REMEMBER THAT MISTAKES AT KEY ENTRY (E.G. PRESSED 8,6) MAY BE CORRECTED BY CONTINUING (PRESS 8,5) –

NOW THAT THE PROGRAM IS LOADED PRESS ONLY ONCE THE 'GO' (G) KEY

K. XXXX .

APPEARS THE K (Ꞙ.) REMINDS YOU OF TWO THINGS: ⊥ THIS IS A DIFFERENT STORED ADDRESS TO THE A. ADDRESS. ⊥⊥ YOU CAN'T GO BACK! (UNLESS YOU EITHER PRESS RESET OR ENTER ADDRESS FFØ4, THE MONITOR ENTRY ADDRESS, AND GO) THE NEXT COMMAND KEY YOU PRESS WILL CAUSE THE µP TO DO A KAMI-KAZE DIVE TO THE ADDRESS SHOWN, SO ITS AS WELL TO GET IT RIGHT!! ENTER Ø,Ø,3,Ø

K. ØØ3Ø .

AND PRESS ANY COMMAND KEY. NOTHING HAPPENED? WELL IT DID, REALLY. IT JUST HAPPENED VERY QUICKLY:

PROGRAM		EXECUTION TIMES, µS
LDA	FEØØ	4
STA Ƶ	2Ø	3
JMP	FFØ4	3
		TOTAL $1Ø_{10}$ µS

IT TOOK TEN MILLIONTHS OF A SECOND TO HAPPEN. WE'RE NOW BACK IN THE MONITOR. PRESSING ANY DIGIT KEY WILL CAUSE THE (BY NOW) FAMILIAR DOTS TO REAPPEAR. PRESS M,Ø,Ø,2,Ø k :

A. ØØ2Ø . AØ

WHICH CHECKS THAT THE PROGRAM ACTUALLY DID WORK. YOU COULD CHANGE ØØ2Ø AND RUN THE PROGRAM AGAIN BY THE KEYS

F, F, G, G, M, M

WHICH SUCCESSIVELY PUT FF IN ØØ2Ø, RUN THE PROGRAM AND RE-EXAMINE

LOCATION ØØ2Ø. A LOT QUICKER FOR YOU THE SECOND TIME, WASN'T IT? THIS IS BECAUSE M & G REMEMBER WHAT THEY WERE POINTING AT. LET'S MAKE THE PROGRAM BETTER. AT THE MOMENT WE HAVE NO IDEA IF IT RAN, AND WE DON'T KNOW IF IT RAN CORRECTLY UNTIL WE LOOK AT ØØ2Ø. IF THE PROGRAM WROTE OUT THE BYTE ON THE DISPLAY AS WELL AS STORING IT IN ØØ2Ø, WE'D KNOW THAT IT HAD ALL HAPPENED. INSIDE THE ACORN MONITOR PROGRAM IS A SET OF INSTRUCTIONS TO WRITE A BYTE ONTO THE TWO RIGHT HAND DISPLAY DIGITS. THIS PROGRAM IS LOCATED AT FE6Ø AND EXPECTS THE BYTE TO BE DISPLAYED TO BE IN THE ACCUMULATOR, WHICH IT IS. THE PROGRAM DESTROYS THIS BYTE AS IT PUTS IT ONTO THE DISPLAY SO WE MUST PUT IT IN ØØ2Ø BEFORE USING THE PROGRAM.

2.3.3 INSTRUCTIONS JMP, JSR

IF WE SIMPLY WENT JMP FE6Ø THIS WOULD CORRECTLY EXECUTE THE PROGRAM BUT WE WOULD BE LEFT IN THE MIDDLE OF THE MONITOR SOMEWHERE SINCE THE PROGRAM DOES NOT HAVE AN ADDRESS TO JUMP BACK TO. WE CAN GIVE IT SUCH AN ADDRESS WITH THE INSTRUCTION JSR (OPCODE 2Ø HEX) THIS IS EXACTLY LIKE A JUMP BUT IT SAVES THE PROGRAM COUNTER BEFORE JUMPING. THEN THE SINGLE BYTE INSTRUCTION RTS (OPCODE 6Ø HEX) RESTORES THE PROGRAM COUNTER AND WE GET BACK AGAIN. JSR IS "JUMP TO SUBROUTINE" AND RTS IS "RETURN FROM SUBROUTINE". THE PROGRAM AT FE6Ø HAS AN RTS ATTACHED AT ITS END, AND SO CAN TRANSFER CONTROL BACK TO THE PROGRAM WHICH CALLED IT. OUR NEW PROGRAM IS 3 BYTES LONGER:

```
ØØ3Ø    AD    LDA FEØØ
ØØ31    ØØ
ØØ32    FE
ØØ33    85    STA Ƶ 2Ø
ØØ34    2Ø
ØØ35    2Ø    JSR FE6Ø
ØØ36    6Ø
ØØ37    FE
ØØ38    4C    JMP FFØ4
ØØ39    Ø4
ØØ3A    FF
```

AND WE WILL HAVE TO ENTER 6 BYTES FROM ØØ35 TO ØØ3A WITH M,Ø,Ø,3,5, k , 2,Ø,↑,6,D,↑,F,E,↑,4,C,↑,Ø,4,↑,F,F. WE HAVEN'T CHANGED THE START OF THE PROGRAM SO G, G WILL RUN IT.

K.   ØØ3Ø   .   AØ

APPEARS MEANING THAT ØØ2Ø HAS AGAIN HAD AØ WRITTEN INTO IT. INSTEAD OF STORING THINGS IN ØØ2Ø, LET'S USE ITS INFORMATION AS PART OF A LOGICAL OPERATION.

APPENDIX A
64 CHARACTER ASCII ON ACORN'S 7 SEGMENT DISPLAY

ASCII CODE	DISPLAY	CHARACTER	HEX	ASCII CODE	DISPLAY	CHARACTER	HEX
Ø		@	5F	2Ø			ØØ
1		A	77	21		!	86
2		B	7C	22		"	22
3		C	58	23		#	63
4		D	5E	24		£	3B
5		E	79	25		%	2D
6		F	71	26		&	7B
7		G	3D	27		'	Ø2
8		H	34	28		(	B9
9		I	Ø5	29		)	8F
A		J	ØD	2A		*	76
B		K	75	2B		+	42
C		L	38	2C		,	Ø4
D		M	37	2D		–	4Ø
E		N	54	2E		.	8Ø
F		0	5C	2F		/	52
1Ø		P	73	3Ø		Ø	3F
11		Q	67	31		1	Ø6
12		R	5Ø	32		2	5B
13		S	ED	33		3	4F
14		T	78	34		4	66
15		U	9C	35		5	6D
16		V	1C	36		6	7D
17		W	7E	37		7	Ø7
18		X	49	38		8	7F
19		Y	6E	39		9	6F
1A		Ƶ	BD	3A		:	82
1B		[	39	3B		;	84
1C		\	64	3C		⟨	46
1D		]	ØF	3D		=	48
1E		Λ	23	3E		⟩	7Ø
1F		–	Ø8	3F		?	D3

PROGRAM. THE STRATEGY OF THE PROGRAM IS NOT OBVIOUS, AND IS LEFT AS AN EXERCISE TO THE READER. A SMALL PRIZE WILL REWARD THE SUBMISSION OF A SHORTER, FASTER PROGRAM; NOTE THAT WORKSPACE REQUIREMENTS CONTRIBUTE TO THE LENGTH!

8 QUEENS PROGRAM

```
Ø2ØØ  F8              MAIN    SED
Ø2Ø1  A2  2Ø                  LDX #2Ø
Ø2Ø3  84  1F                  STY COUNT      – CLEAR COUNT
Ø2Ø5  84  2Ø                  STY ROW        – CLEAR ROW OCCUPIED
Ø2Ø7  84  29                  STY LEFT       – CLEAR LEFT DIAGONAL ATTACKS
Ø2Ø9  84  32                  STY RIGHT      – CLEAR RIGHT DIAGONAL ATTACKS
Ø2ØB  2Ø  16  Ø2              JSR TRY        – FIND THE NO OF WAYS
Ø2ØE  A5  1F                  LDA COUNT
Ø21Ø  2Ø  6Ø  FE              JSR RDHEXTD    – DISPLAY ANSWER
Ø213  4C  Ø4  FF              JMP RESTART
Ø216  B5  ØØ          TRY     LDAƵX ØØ       – FINISHED YET?
Ø218  C9  FF                  CMP #FF
Ø21A  DØ  Ø7                  BNE CONTINUE
Ø21C  A5  1F                  LDA COUNT      – FINISHED, SO INCREMENT COUNT
Ø21E  69  ØØ                  ADC #ØØ
Ø22Ø  85  1F                  STA COUNT
Ø222  6Ø              FINISH  RTS
Ø223  15  Ø9                  ORAƵX Ø9       – CURRENT LEFT
Ø225  15  12                  ORAƵX 12       – CURRENT RIGHT
Ø227  A8              LOOP    TAY
Ø228  49  FF                  EOR #FF
Ø22A  FØ  F6                  BEQ FINISH     – NO CHANCE
Ø22C  95  18                  STAƵX 1B       – CURRENT POSSIBLE PLACE
Ø22E  C8                      INY
Ø22F  98                      TAY
Ø23Ø  35  1B                  ANDƵX 1B
Ø232  A8                      TYAY
Ø233  15  ØØ                  CRAƵX ØØ
Ø235  95  Ø1                  STAƵX Ø1       – NEW ROW
Ø237  98                      TYA
Ø238  15  Ø9                  ORAƵX Ø9
Ø23A  ØA                      ASLA
Ø23B  95  ØA                  STAƵX ØA       – NEW LEFT ATTACK
Ø23D  98                      TYA
Ø23E  15  12                  ORAƵX 12
Ø24Ø  4A                      LSRA
Ø241  95  13                  STAƵX 13       – NEW RIGHT ATTACK
Ø243  E8                      INX
Ø244  2Ø  16  Ø2              JSR TRY
Ø247  CA                      DEX
Ø248  B5  Ø1                  LDAƵX Ø1
Ø24A  49  FF                  EOR #FF
Ø24C  35  1B                  ANDƵX 1B
Ø24E  49  FF                  EOR #FF
Ø25Ø  4C  27  Ø2              JMP LOOP
Ø253
```

2.3.4 THE LOGIC INSTRUCTIONS 'ORA', 'AND', 'EOR'.

IF WE PUT $6Ø_{16}$ IN LOCATION ØØ2Ø (M,Ø,Ø,2,Ø, k, ,6,Ø : YOU SHOULD KNOW BY NOW) AND ALTER THE STA Ƶ INSTRUCTION AT ØØ33 TO, SAY, ORA Ƶ (OPCODE Ø5 HEX) (THE PROGRAM READS

```
LDA   FEØØ
ORA   Ƶ 2Ø
JSR   FE6Ø
JMP   FFØ4)
```

WE HAVE A PROGRAM THAT DISPLAYS THE LOGICAL 'OR' BETWEEN THE CONTENTS OF FEØØ (AØ) AND ØØ2Ø, (6Ø). THE HEX FOR ORA Ƶ IS Ø5 AND IT CARRIES OUT A LOGICAL 'OR' BETWEEN THE ACCUMULATOR AND THE SPECIFIED LOCATION IN Ƶ PAGE. M,Ø,Ø,3,3, k Ø,5 IS THE MODIFICATION TO THE PROGRAM, THEN SINCE WE STILL START AT ØØ3Ø, G,G RUNS IT :

K.   ØØ3Ø   .   EØ

THE OPERATION WAS 'OR' :

```
AØ         1Ø1ØØØØØ
6Ø   or    Ø11ØØØØØ   or
EØ         111ØØØØØ
```

TRY CHANGING ØØ2Ø TO 4Ø AND RUNNING THE PROGRAM AGAIN IS THE ANSWER WHAT YOU EXPECTED?

WE CAN CHANGE ØØ33 TO MAKE THE PROGRAM DO LOGICAL 'AND' OR 'EXCLUSIVE – OR'. THE MNEMONICS AND OPCODES ARE:

AND	Ƶ	25_{16}	LOGICAL AND ACCUMULATOR AND Ƶ PAGE MEMORY
EOR	Ƶ	45_{16}	LOGICAL EXCLUSIVE–OR ACCUMULATOR AND Ƶ PAGE MEMORY

AND THE PROGRAMS WOULD READ

```
LDA FEØØ     &     LDA  FEØØ
AND Ƶ 2Ø           EOR  Ƶ 2Ø
JSR  FE6Ø          JSR  FE6Ø
JMP  FFØ4          JMP  FFØ4
```

BY NOW YOU MUST BE GETTING TIRED OF THE AØ IN FEØØ SO WE'LL CHANGE THE PROGRAM TO READ

```
LDA  Ƶ 21
EOR  Ƶ 2Ø
JSR  FE6Ø
JMP  FFØ4
```

THE SPACE TAKEN UP BY LDA Ƶ 21 IS ONE BYTE LESS THAN THAT USED BY LDA FEØØ. WE COULD SIMPLY WRITE THE NEW TWO BYTES IN AT LOCATIONS ØØ31 & ØØ32 AND CHANGE THE GO ADDRESS TO ØØ31. THIS IS VERY SIMPLE HERE SINCE THAT IS ALL WE HAVE TO DO. BUT IF THERE WERE MANY REFERENCES TO ØØ3Ø AS THE START OF THIS PROGRAM IT WOULD TAKE A LONG TIME TO FIND AND CHANGE THEM ALL, AND IF WE DIDN'T CHANGE THEM ALL SOMETHING WOULD GO WRONG. WE CAN'T MOVE THE REST OF THE PROGRAM DOWN ONE BYTE: SOMETHING MIGHT BE REFERRING TO IT. THE PROBLEM ARISES BECAUSE LDA Ƶ IS SHORTER THAN LDA. WE COULD SIMPLY USE LDA WITH A ZERO PAGE ADDRESS BUT THIS TAKES A WHOLE μS

LONGER THAN LDA Ƶ! THE SOLUTION IS TO USE LDA Ƶ AND TO INCORPORATE AN EXTRA BYTE IN ØØ3Ø AS PADDING. THIS MUST BE A SINGLE-BYTE INSTRUCTION, THAT DOES NOTHING TO AFFECT THE PROGRAM, AND ONE IS SPECIFICALLY PROVIDED

NOP	EA	"NO OPERATION"

THE PROGRAM READS

```
ØØ3Ø EA          NOP
ØØ31 A5 21       LDA Ƶ 21
ØØ33 45 2Ø       EOR Ƶ 2Ø
ØØ35 26 6Ø FE    JSR FE6Ø
ØØ38 4C Ø4 FF    JMP FFØ4
```

—NOTICE THE MORE COMPACT MODE OF WRITING IT DOWN. THIS IS MORE CONSISTENT WITH THE WAY MNEMONICS ARE WRITTEN. IT IS EXACTLY EQUIVALENT TO

```
ØØ3Ø EA          NOP
ØØ31 A5          LDA Ƶ 21
ØØ32 21
ØØ33 45          EOR Ƶ 2Ø
ØØ34 2Ø
ØØ35 2Ø          JSR FE6Ø
ØØ36 6Ø
ØØ37 FE
ØØ38 4C          JMP FFØ4
ØØ39 Ø4
ØØ3A FF
```

AND IT WILL BE USED THROUGHOUT THE REST OF THE MANUAL:
THIS PROGRAM TAKES THE CONTENTS OF (WHICH MAY BE WRITTEN BY PUTTING BRACKETS AROUND THE PARTICULAR ADDRESS) ØØ2Ø & ØØ21 AND PRESENTS THEIR LOGICAL EXCLUSIVE – OR ON THE DISPLAY. APART FROM THEIR LOGICAL FUNCTIONS, THESE OPERATORS ARE OFTEN USED TO MANIPULATE SINGLE BITS. FOR INSTANCE ORA #Ø1 WOULD SET BIT Ø OF THE ACCUMULATOR, AND # FE WOULD CLEAR IT AND EOR #Ø1 WOULD COMPLENT IT, ALL WITHOUT AFFECTING ANY OTHER BITS IN THE ACCUMULATOR.

2.3.5 ARITHMETIC INSTRUCTIONS 'ADC', 'SEC', 'CLC'.

FROM LOGIC OPERATIONS WE PROGRESS AGAIN TO ARITHMETIC. LOOKING AT ORA Ƶ, EOR Ƶ, AND Ƶ WOULD LEAD ONE TO ASSUME THE EXISTENCE OF ADD Ƶ. WELL, THERE ISN'T ONE, THERE'S ONLY ADC Ƶ.

BYTES:			
2	ADC Ƶ	65	"ADD WITH CARRY, ZERO PAGE"
1	SEC	38	"SET CARRY FLAG"
1	CLC	18	"CLEAR CARRY FLAG"

THIS IS MOST UNUSUAL AND A TRAP FOR UNWARY PROGRAMMERS, ESPECIALLY THOSE USED TO µPs WHICH POSSESS AN ADD INSTRUCTION: THE CARRY FLAG MUST BE CLEARED BEFORE AN ADC (OR IT MUST BE IN A KNOWN STATE E.G. { SEC / ADC #ØØ } ≡ { CLC / ADC #Ø1 } OR 'UNEXPECTED' ANSWERS WILL APPEAR. WHEN THE µP LEAVES THE MONITOR USING THE GO ROUTINE THE CARRY FLAG IS SET: FAILURE TO CLEAR IT BEFORE AN ADC RESULTS IN AN ANSWER 1 GREATER THAN EXPECTED.

3

THE METRONOME PRODUCES A PULSE AT THE TAPE OUTPUT PIN, PA6, WITH A REGULAR PERIOD. THE "UP" AND "DOWN" KEYS WILL INCREASE AND DECREASE THE PERIOD RESPECTIVELY. WITH SUITABLE ADDITIONAL CIRCUITRY THIS COULD DRIVE A LOUDSPEAKER OR A 'STROBE' LIGHT. IN FACT A SMALL SOUND CAN BE OBTAINED BY SIMPLY CONNECTING A LOUD-SPEAKER ACROSS THE TAPE OUTPUT AND EARTH PINS.
THE CONSTANTS USED AT PRESENT MEAN THAT THE PULSE IS OF 1/3ØØ SEC. AND THE DELAY BETWEEN PULSES CAN BE VARIED FROM 1/2Ø SEC. TO ABOUT 13 SECS. YOU CAN DEFINE THE PERIOD BEFORE STARTING THE PROGRAM BY PUTTING THE REQUIRED VALUE INTO MEMORY LOCATION ØØ2Ø. 2Ø WILL GIVE ABOUT 1 SEC BETWEEN PULSES, AND ANYTHING ELSE PROPORTIONATELY MORE OR LESS. ONCE THE PROGRAM IS RUNNING THE 'UP' AND 'DOWN' KEYS WILL INCREMENT AND DECREMENT THE PERIOD BY ABOUT 1/2Ø SEC EACH TIME THEY ARE PRESSED. THEY ALSO RESET THE CYCLE. THIS FACILITY COULD USEFULLY BE USED FOR FINE TUNING BUT WOULD BE TEDIOUS FOR LARGE CHANGES OF PERIOD.

METRONOME

ADDR	HEX CODE			LABEL	INSTRUCTION	COMMENTS
Ø2ØØ	A9	1F			LDA #1F	
Ø2Ø2	85	ØE			STA REPEAT	– SET DISPLAY TO SINGLE SCAN
Ø2Ø4	A9	4Ø		PULSE	LDA #4Ø	
Ø2Ø6	8D	22	ØE		STA 1ADDR	– DEFINE PA6 AS OUTPUT
Ø2Ø9	8D	16	ØE		STA SET PIA6	– USE INS8154 SET BIT MODE
Ø2ØC	2Ø	CD	FE		JSR WAIT	– USE THE 3ØØ BAND WAIT
Ø2ØF	8D	Ø6	ØE		STA CLR PIA6	– USE IN58154 CLEAR BIT MODE
Ø212	A6	2Ø			LDXƵ PERIOD	
Ø214	2Ø	ØC	FE	DELZ	JSR DISPLAY	– LOOK AT KEYBOARD
Ø217	C9	16			CMP #16	– UP KEY?
Ø219	DØ	Ø4			BNE DOWN	– NO
Ø21B	E6	2Ø			INCƵ PERIOD	– INCREASE PERIOD
Ø21D	BØ	E5			BCS PULSE	– CARRY WAS SET BY THE COMPARE: ALWAYS
Ø21F	C9	17		DOWN	CMP #17	– DOWN KEY?
Ø221	DØ	Ø4			BNE DELI	– NO
Ø223	C6	2Ø			DECƵ PERIOD	– DECREASE PERIOD
Ø225	BØ	DD			BCS PULSE	– CARRY WAS SET BY THE COMPARE: ALWAYS
Ø227	AØ	ØC		DELI	LDY #ØC	– CYCLE TIME OF µ½Ø SEC.
Ø229	2Ø	CD	FE	DELJ	JSR WAIT	
Ø22C	88				DEY	
Ø22D	1Ø	FA			BPL DELJ	
Ø22F	CA				DEX	
Ø23Ø	DØ	E2			BNE DEL2	
Ø232	FØ	DØ			BEQ PULSE	– END OF THIS PERIOD SO PULSE
Ø234						

4

THE EIGHT QUEENS PROBLEM IS TO FIND THE NUMBER OF WAYS IN WHICH EIGHT QUEENS MAY BE PLACED ON A CHESS BOARD WITHOUT ATTACKING EACH OTHER. THE PROGRAM FINDS 92 WAYS SINCE IT COUNTS ROTATIONS AND REFLECTIONS, ALLPOSSIBLE POSITIONS ARE TRIED AS SOLUTIONS IN THIS HIGH SPEED RECURSIVE (I.E. IS DEFINED IN TERMS OF ITSELF)

COUNTER KEYBOARD

```
ADDR  HEX          LABEL    INSTRUCTION    COMMENTS
      CODE
                                           – START OF ∅∅1C
∅∅1D  2∅  ∅C  FE   DISP     JSR DISPLAY    – LOOK FOR KEY
∅∅2∅  9∅  ∅A                BCC CHANGE     – CHECK IF CONTROL KEY CARRY
                                             SET IF SO
∅∅22  C9  ∅7                CMP #'∅7
∅∅24  F∅  1F                BEQ DOWN
∅∅26  C9  ∅6                CMP # ∅6
∅∅28  F∅  11                BEQ UP
∅∅2A  D∅  F1                BNE DISP
∅∅2C  C9  ∅∅       CHANGE   CMP # ∅∅
∅∅2E  85  19                STA COUNT
∅∅3∅  F∅  ED       MORE     BEQ DISP     }
∅∅32  2∅  6∅  ∅∅            JSR INCR     }
∅∅35  C6  19                DEC COUNT    }   INCREMENT NO OF TIME OF TEY
∅∅37  1∅  F7                BPL MORE     }
∅∅39  3∅  E2                BMI DISP     }
∅∅3B  2∅  6∅  ∅∅   UP       JSR INCR     }
∅∅3E  2∅  45  ∅∅            JSR ZOOM     }
∅∅N                                      }   RAPID INCREMENT
∅∅41  D∅  CF                BNE DISP     }
∅∅43  F∅  F6                BEQ UP       }
∅∅45  2∅  69  ∅∅   DOWN     JSR DECR     }
∅∅48  2∅  4F  ∅∅            JSR ZOOM     }   RAPID INCREMENT
∅∅4B  D∅  D9                BNE DISP     }
∅∅4D  F∅  F6                BEQ DOWN     }
∅∅4F  A9  1F       ZOOM     LDA #1F
∅∅51  85  ∅E                STA ∅E           SET FOR ONE SCAN ONLY
∅∅53  2∅  ∅C  FE            JSR DISPLAY
∅∅56  9∅  ∅3                BCC STOP         CHECK IF KEY DEPRESSED CLEAR
                                             IF ONE IS
∅∅58  A9  ∅∅                LDA # ∅∅
∅∅5A  6∅                    RTS
∅∅5B  A9  FF                LDA # FF         RESET SO THAT JSR DISPLAY
                                             WAITS FOR INPUT
∅∅5D  A9  FF                LDA # FF
∅∅5D  85  ∅E                STA ∅E
∅∅5F  6∅                    RTS
```

COUNTER SUBROUTINE

```
ADDR  HEX          LABEL    INSTRUCTION    COMMENTS
      CODE

∅∅6∅  F6  1A       INCR     INC CNTL
∅∅62  D∅  ∅D                BNE UPDATE
∅∅64  E6  1B                INC CNTH
∅∅66  38                    SEC
∅∅67  B∅  ∅8                BCS UPDATE
∅∅69  A5  1A       DECR     LDA CNTL
∅∅63  D∅  ∅2                BNE NOT
∅∅6D  C6  1B                DEC CNTH
∅∅6F  C6  1A       NOT      DEC CNTL
∅∅71  A2  1A       UPDATE   LDX #IE
∅∅73  2∅  64  FE            JSR QHEXTD1
∅∅76  6∅                    RTS
```

ANOTHER TRAP FOR THOSE USED TO DIFFERENT μPs IS THE DECIMAL FLAG. INSTEAD OF A SINGLE "DECIMAL ADJUST" INSTRUCTION TO ADJUST THE RESULT OF BINARY ARITHMETIC ON B.C.D. NUMBERS TO B.C.D. THERE ARE TWO INSTRUCTIONS

```
BYTES: 1    SED    F8    "SET DECIMAL MODE"
       1    CLD    D8    "CLEAR DECIMAL MODE"
```

WHICH INSTRUCT THE PROCESSOR TO DO AUTOMATICALLY (OR NOT DO) THE ADJUSTMENT AFTER ARITHMETIC OPERATIONS. THIS RESULTS IN SHORTER, FASTER PROGRAMS FOR HANDLING B.C.D. ARITHMETIC ..... WHICH, MERELY BY CHANGING THE DECIMAL MODE FLAG, WILL HANDLE BINARY ARITHMETIC. IN ORDER TO FULLY UTILISE THE μP's POWER THE MONITOR SUBROUTINES FOR FETCHING KEYS & OUTPUTTING DATA TO THE DISPLAY HAVE BEEN WRITTEN WITHOUT ARITHMETIC SO THEY MAY BE CALLED WITH THE DECIMAL FLAG SET OR CLEARED & THEY WILL NOT AFFECT IT.
SO LET'S DO A DECIMAL ADDITION;

```
∅∅2F    F8          SED
∅∅3∅    18          CLC
∅∅31    A5 21       LDA Ƶ 21
∅∅33    65 2∅       ADC Ƶ 2∅
∅∅35    2∅ 6∅ FE    JSR FE6∅
∅∅38    4C∅4 FF     JMP FF∅4
```

OUR STANDARD PROGRAM HAS BEEN EXTENDED BACKWARDS BY ONE BYTE, THE SED INSTRUCTION. THIS SHOULD BE INCLUDED (BY ,∅,∅,2,F,∅) THE FIRST TIME THE PROGRAM IS RUN, BUT MAY BE OMMITTED (K,∅,∅,3,∅,4) ON SUBSEQUENT RUNS. THIS LITTLE PROGRAM WILL TELL US THAT 22 + 11 = 33, IT WILL SAY THAT 35 + 26 = 61 AND THAT 5∅ + 51 = ∅1 WHOOPS! THE PROGRAM AT FE6∅ ONLY DEALS WITH PUTTING THE BYTE IN THE ACCUMULATOR ON THE DISPLAY. IT PAYS NO ATTENTION TO THE CARRY FLAG, INDEED IT CHANGES THE STATE OF THE CARRY FLAG ITSELF, SO THAT WE CAN'T IMMEDIATELY CALL FE6∅, HAVE IT WRITE ON THE DISPLAY & RETURN THEN WRITE OUT THE STATE OF THE CARRY SOMEHOW. WHAT WE NEED IS:

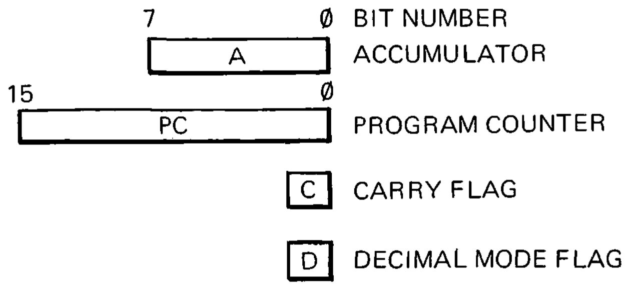
I SAVE THE CARRY FLAG
II USE FE6∅
III GET THE CARRY FLAG BACK & WRITE IT OUT SOMEHOW

A FRENZIED SEARCH THROUGH THE MNEMONICS REVEALS THAT THERE ARE NO MNEMONICS LIKE LDC (LOAD C) OR STC (STORE C)
A CLOSER LOOK AT THE MICROPROCESSOR IS REQUIRED.

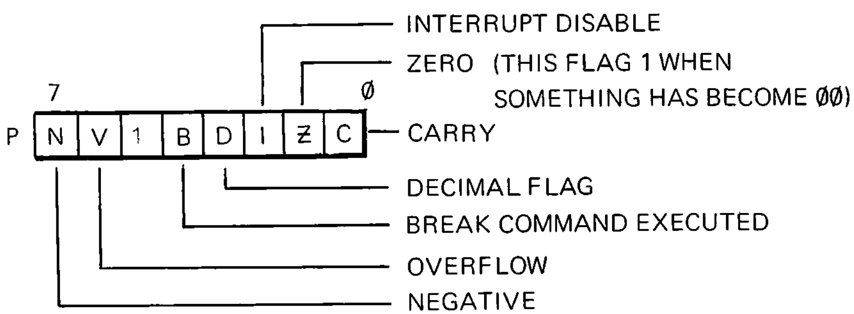
CHAPTER 3: INSIDE THE 65Ø2

SO FAR THE PROCESSOR'S INTERNAL WORKINGS ARE

3.1 THE ACCUMULATOR, PROGRAM COUNTER, STATUS REGISTER



THE CARRY & DECIMAL MODE FLAGS HAVE BEEN TREATED SEPARATELY TO DATE. THEY ARE ACTUALLY MEMBERS OF A SPECIAL REGISTER CALLED THE PROCESSOR STATUS REGISTER,P.



CAN WE, THEN, USE LDP & STP? NO, THEY DON'T EXIST EITHER.(FUME). IN ORDER TO SOLVE THIS PROBLEM WE MUST INTRODUCE THE STACK.
DID YOU WONDER JUST WHAT HAPPENED TO PC DURING A JSR? YOU WERE TOLD THAT IT WAS 'SAVED'. WHERE? HOW? IT WOULD BE TERRIBLE TO HAVE TO SPECIFY WHERE IT HAD TO BE STORED. WHAT'S NEEDED IS SOME PLACE WHERE IT CAN BE PUT DOWN AND PICKED UP AGAIN. IT WOULD BE GOOD TO ALLOW NESTED SUBROUTINES:

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
Ø213	95 1Ø		STA Ƶ X 1Ø	
Ø215	A9 61	INSERT	LDA #DUCK	– PUT NEW DUCK ON
Ø217	CA		DEX	– IN NEW POSITION
Ø218	1Ø Ø2		BPL OLDX	– BUT NOT OVER THE END OF THE DISPLAY
Ø21A	A2 Ø7		LDX #Ø7	
Ø21C	95 1Ø	OLDX	STA Ƶ X 1Ø	
Ø21E	86 2Ø		STX Ƶ 2Ø	
Ø22Ø	A2 ØE		LDX #TIME	– DISPLAY INTERVAL IS SET BY THE BYTE LOADED INTO X
Ø222	2Ø ØC FE	WAIT	JSR DISPLAY	
Ø225	C5 2Ø		CMP Ƶ 2Ø	– HIT?
Ø227	FØ Ø5		BEQ H1T	
Ø229	CA		DEX	
Ø22A	DØ F6		BNE WAIT	
Ø22C	FØ E1		BEQ REMOVE	– FINISHED WAIT TIME
Ø22E	A9 1C	HIT	LDA #DEAD DUCK	– PUT IN A DEAD DUCK
Ø23Ø	A6 2Ø		LDX Ƶ 2Ø	
Ø232	95 1Ø		STA Ƶ X 1Ø	
Ø234	A9 FF		LDA #FF	
Ø236	85 ØE		STX Ƶ ØE	
Ø238	2Ø ØC FE		JSR DISPLAY	– TEST FOR CONTINUATION
Ø23B	9Ø C3		BCC BEGIN	
Ø23D	4C Ø4 FF		JMP RESTART	– OR BACK TO THE MONITOR
Ø23F				

MISCELLANEOUS

1

THE COUNTER PROGRAM COULD BE USED AS A SUBROUTINE IN A LONGER PROGRAM WHEN "JSR INCR" AND "JSR DECR" WOULD INCREMENT OR DECREMENT THE DISPLAY. IF THE PROGRAM APPENDED IS ALSO ENTERED THE DISPLAY WILL INCREASE OR DECREASE RAPIDLY IF "UP" OF "DOWN" KEYS ARE DEPRESSED. THIS WILL BE STOPPED BY ANY HEX KEY. IT WILL INCREMENT BY THE INDICATED AMOUNT IF KEYS 1–F ARE DEPRESSED AND WILL IGNORE ALL OTHER KEYS.
YOU SHOULD PARTICULARLY NOTICE THAT A JSR DISPLAY RETURNS WITH THE CARRY BIT CLEAR AND THE ACCUMULATOR HOLDING THE VALUE OF THE KEY PRESSED FOR THE NUMERICAL KEYS, AND THE CARRY BIT SET AND THE VALUES Ø–7 IN THE ACCUMULATOR FOR THE CONTROL KEYS. IF MEMORY LOCATION ØE, WHICH IS DEDICATED TO THE MONITOR AND SHOULD NOT NORMALLY BE USED IN PROGRAMS, HAS THE MOST SIGNIFICANT BIT CLEAR THEN JSR DISPLAY WILL SCAN ONLY ONCE, IF IT IS SET IT WILL WAIT FOR A KEY TO BE DEPRESSED BEFORE RETURNING TO THE PROGRAM. IT IS A GOOD IDEA TO LOAD IT WITH 'IF' IF YOU WISH TO USE THIS FACILITY AS OTHER VALUES MAY CAUSE YOU DIFFERENT PROBLEMS. AGAIN SEE THE REST OF THIS MANUAL IF YOU REALLY WISH TO UNDERSTAND THE PROCESS.

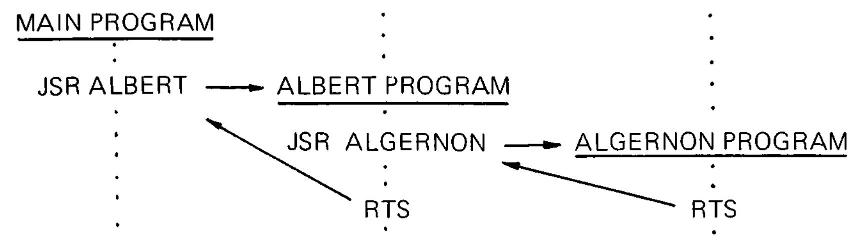
```
ADDR  HEX        LABEL    INSTRUCTION       COMMENTS
      CODE
027C  69  00              ADC #00
027E  46  2A              LSR ANAL + 2
0280  69  00              ADC #00
0282  46  2B              LSR ANAL + 3
0284  69  00              ADC #00
0286  4A                  LSRA
0287  B0  BF              BCS ONEOFF        - NOT A GOOD MOVE
0289  C6  1F              DEC COUNT
028B  D0  E8              BNE CONT          - KEEP CHECKING THE MOVE
028D  A2  03              LDX #03           - GOOD MOVE, TRANSFER TO
                                              ACTUAL STACKS
028F  B5  24     BAT      LDAZX POSS
0291  95  20              STAZX STACK
0293  CA                  DEX
0294  10  F9              BPL BAT
0296  4C  00  02          JMP HUMMOV        - OPPONENT.
0299  A9  00     DSPGAP   LDA #00
029B  A2  07              LDX #07
029D  95  10     CLEAR    STAZX D           - CLEAR THE DISPLAY FIRST
029F  CA                  DEX
02A0  10  FB              BPL CLEAR
02A2  D8                  CLD               - CLEAR DECIMAL MODE
02A3  A2  04              LDX #04           - DISPLAY STACKS
02A5  A0  01 07           LDY #01 07
02A7  B5  1F     AROUND   LDAZX STACK -1
02A9  20  7A  FE          JSR HEXTD
02AC  C8 88               INY DEY
02AD  C8 88               INY DEY
02AE  CA                  DEX
02AF  D0  F6              BNE AROUND
02B1  60                  RTS                  # 1F
02B2
```

2

THE DUCKSHOOT GAME IS A SPEED TEST: YOU HAVE TO SHOOT THE FLYING DUCKS. THEY SUCCESSIVELY ENTER FROM THE RIGHT AND FLY TOWARDS THE LEFT AT A SET SPEED. YOU SHOOT A DUCK BY PRESSING ITS CURRENT POSITION ON THE KEYBOARD. THE LEFT MOST DISPLAY IS 0, THE RIGHTMOST DISPLAY IS 7. WHEN A DUCK IS HIT IT DIES. THE GAME MAY BE RESTARTED WITH ANY HEX DIGIT KEY

DUCK SHOOT

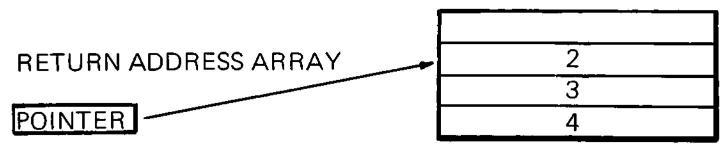
```
ADDR  HEX        LABEL    INSTRUCTION       COMMENTS
      CODE
0200  A9  1F     BEGIN    LDA #1F           - SINGLE SCAN DISPLAY ROUTINE
0202  85  0E              STA Z 0E
0204  A9  00              LDA #00           - CLEAR THE DISPLAY
0206  A2  07              LDX #07
0208  86  20              STX Z 20
020A  95  10     CLEAR    STA Z X 10
020C  CA                  DEX
020D  10  FB              BPL CLEAR
020F  A9  00     REMOVE   LDA #00           - TAKE THE OLD DUCK OFF
0211  A6  20              LDX Z 20
```



WE CAN'T JUST SAY THAT PC IS TO BE SAVED IN LOCATION, SAY, L & M – WE WOULDN'T GET BACK FROM ALBERT SINCE THE CALL TO ALGERNON WOULD HAVE DESTROYED THE NECESSARY INFORMATION IN L & M. (IT IS WORTH NOTING HERE THAT L & M COULD BE "CALLED" –2 "CALLED" –1. THEN A CALL TO ALBERT AS A SUBROUTINE WOULD STORE THE RETURN ADDRESS JUST BEFORE THE START OF ALBERT ALLOWING NESTED SUBROUTINES AS ABOVE. A PROBLEM IS THAT THIS DOES NOT WORK WITH READ ONLY MEMORY, LIKE THE MONITOR).

3.2 THE STACK POINTER

WE NEED SOMETHING WHICH WILL DECIDE WHAT L & M ARE TO BE, DEPENDING ON WHICH SUBROUTINE WE ARE IN. AN OBVIOUS CHOICE IS TO USE AN ARRAY OF MEMORY LOCATIONS, AND A VARIABLE WHICH POINTS TO THE CURRENT LOCATION OF L & M, EACH TIME WE DO A JSR WE STEP UP THE POINTER & EACH TIME WE DO AN RTS WE STEP IT DOWN.



WITH ACORN WE'LL NEED TWO BYTES FOR EACH RETURN ADDRESS. THIS IS NO TROUBLE, WE JUST INCREMENT & DECREMENT THE POINTER TWICE. THE WHOLE PROCESS IS CARRIED OUT BY THE PROCESSOR AUTOMATICALLY ON EACH JSR & RTS, THE POINTER IS CALLED THE STACK POINTER AND IS A SPECIAL 8 BIT REGISTER INSIDE THE PROCESSOR. THE ARRAY IS USUALLY CALLED A STACK SINCE IT CAN ALSO BE USED TO STORE THINGS OTHER THAN RETURN ADDRESSES. THE ACTUAL STACK RUNS FROM 01FF DOWN TO 0100, AND IT STARTS AT THE TOP: AN EMPTY STACK HAS STACK POINTER AT FF. A BYTE IS PUT ON THE STACK AND THE POINTER IS DECREMENTED TO POINT AT THE NEXT LOCATION; THE POINTER IS INCREMENTED AND A BYTE LOADED FROM THE STACK IN THE REVERSE OPERATION. NO CHECK IS MADE FOR THE 00 TO FF DECREMENT INDICATING AN OVERFLOWED STACK, SO PROGRAMS THAT REQUIRE MORE THAN 256 BYTES OF STACK SPACE WILL MYSTERIOUSLY FAIL. SINCE THIS IS 128 CONSECUTIVE JSR'S, THE PROBLEM WON'T BE ENCOUNTERED VERY OFTEN...

NOW THE PROCESSOR STATUS REGISTER CAN BE PUSHED ONTO THE STACK:

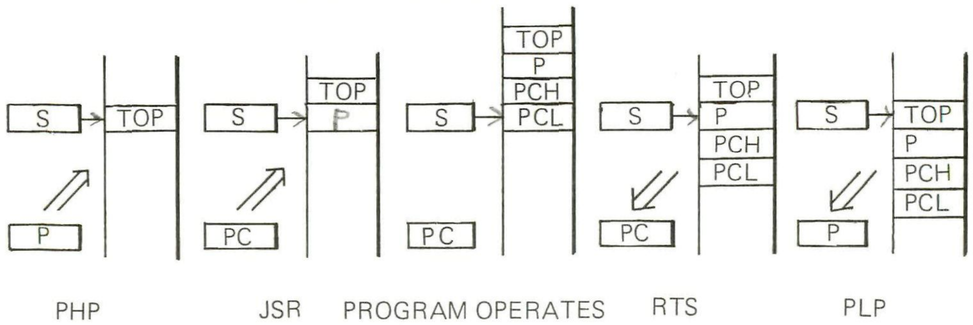
PLP 28 "PULL P"

PHP Ø8 "PUSH P"

AND SO WE MAY SAVE IT BEFORE A SUBROUTINE CALL AND RECOVER IT AFTERWARDS

```
PHP
JSR . . . .
PLP
```

THE SEQUENCE OF STACK OPERATIONS IS



SO WE HAVE NOW MANAGED TO SAVE THE CARRY FLAG, USE FE6Ø, AND REGAIN THE CARRY FLAG. WE WISH TO WRITE IT OUT, SO IT WOULD HAVE BEEN BETTER TO WRITE.

```
PHP
JSR FE6Ø
PLA          PULL BYTE FROM STACK INTO A
```

SINCE THIS GIVES THE CARRY FLAG IN A, AS THE LEAST SIGNIFICANT BIT, TO GET RID OF THE REST OF THE BITS OF THE RECOVERED STATUS REGISTER, WE CAN SIMPLY AND # Ø1. NOW A CONTAINS Ø OR 1 DEPENDING ON THE CARRY FROM ORIGINAL SUM. OUR PROGRAM NOW IS

```
SED                   SET UP FOR DECIMAL ADD
CLC
LDA Ƶ 21              DO IT
ADC Ƶ 2Ø
PHP                   SAVE CARRY
JSR FE6Ø              WRITE OUT TWO DIGITS
                      ON DISPLAYS 6 & 7
PLA
AND #Ø1               A = Ø (NO CARRY FROM SUM)
                          OR A = 1 (CARRY FROM SUM)
```

NOW ALL WE NEED TO DO IS WRITE OUT THE ACCUMULATOR ON DISPLAY NO.5. THE WAY WE WROTE OUT THE FIRST TWO DIGITS OF THE RESULT WAS TO USE A MONITOR SUBROUTINE WHICH DID JUST THAT. YOU'VE PROBABLY NOTICED THAT THE MONITOR ONLY PUTS A DOT ON DISPLAY 5 (THE 3RD

ADDR	HEX CODE			LABEL	INSTRUCTION	COMMENTS
Ø21Ø	29	7F			AND #7F	
Ø212	95	11			STAƵX D + 1	
Ø214	E8				INX	– MOVE FORWARD
Ø215	E8				INX	
Ø216	EØ	Ø7			CPX #Ø7	– END OF STACKS?
Ø218	9Ø	E9			BCC SHIFTPT	
Ø21A	A2	ØØ			LDX #ØØ	
Ø21C	FØ	E5			BEQ SHIFTPT	
Ø21E	A8			MINUS	TAY	
Ø21F	FØ	E8			BEQ CHEAT	– PREVENT ZERO FROM BEING USED
Ø221	8A				TXA	
Ø222	4A				LSRA	– ADDRESS OF REQUIRED STACK
Ø223	AA				TAX	
Ø224	38				SEC	
Ø225	B5	2Ø			LDAƵX STACK	– DO THE PLAYER'S MOVE
Ø227	E5	ØD			SBC KEY	
Ø229	95	2Ø			STAƵX STACK	
Ø22B	2Ø	99	Ø2	COMMOV	JSR DSPGAP	– SHOW STACKS
Ø22E	84	ØE			STY REPEAT	
Ø23Ø	A2	ØØ			LDX #ØØ	
Ø232	2Ø	ØC	FE	WAIT	JSR DISPLAY	– THINKING TIME
Ø235	CA				DEX	
Ø236	DØ	FA			BNE WAIT	
Ø238	CA				DEX	
Ø239	86	ØE			STX REPEAT	– CLEAR REPEAT STATUS
Ø23B	AØ	Ø3			LDY #Ø3	
Ø23D	A2	Ø3		NEXTS	LDX #Ø3	– TRANSFER STACK TO POSS
Ø23F	B5	2Ø		BLOCK	LDAƵX STACK	– POSS REPRESENTS THE POSSIBLE COMPUTER
Ø241	95	24			STAƵX POSS	– MOVES
Ø243	CA				DEX	
Ø244	1Ø	F9			BPL BLOCK	
Ø246	A2	Ø3		ONEOFF	LDX #Ø3	– TRANSFER POSS TO ANAL
Ø248	B5	24		BRICK	LDA2X POSS	– ANAL REPRESENTS THE MOVE BEING
Ø24A	95	28			STA2X ANAL	– ANALYSED
Ø24C	CA				DEX	
Ø24D	1Ø	F9			BPL BRICK	
Ø24F	A2	Ø3			LDX #Ø3	
Ø251	B9	24	ØØ		LDA, Y POSS	
Ø254	38				SEC	
Ø255	E9	ØØ			SBC #Ø1	
Ø257	99	24	ØØ		STA, Y POSS	– POSS CONTAINS POSSIBLE MOVE
Ø25A	99	28	ØØ		STA, Y ANAL	– ANAL CONTAINS POSSIBLE MOVE
Ø25D	BØ	12			BCS CHECK	
Ø25F	88				DEY	
Ø26Ø	1Ø	DB			BPL NEXTS	– TRY ALL STACKS
Ø262	B5	2Ø		TRY	LDAƵX STACK	– CHECK IF STACK EMPTY
Ø264	FØ	Ø5			BEQ EMPTY	
Ø266	D6	2Ø			DECƵX STACK	– MAKE DESPERATE MOVE
Ø268	4C	ØØ	Ø2		JMP HUMMOV	
Ø26B	CA			EMPTY	DEX	
Ø26C	1Ø	F4			BPL TRY	
Ø26E	4C	Ø4	FF		JMP RESTART	– LOST.
Ø271	A9	Ø4		CHECK	LDA #Ø4	
Ø273	85	1F			STA COUNT	
Ø275	A9	ØØ		CONT	LDA #ØØ	– EVALUATE MOVE
Ø277	46	28			LSR ANAL	
Ø279	2A				ROLA	
Ø27A	46	29			LSR ANAL + 1	

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
Ø2Ø7	EØ Ø7		CPX #Ø7	
Ø2Ø9	DØ F7		BNE LOOP	– KEEP GOING
Ø2ØB	85 17		STA ≥ D + 7	– NEW DATA
Ø2ØD	6Ø		RTS.	
Ø2ØE				

GAMES PROGRAMS
1

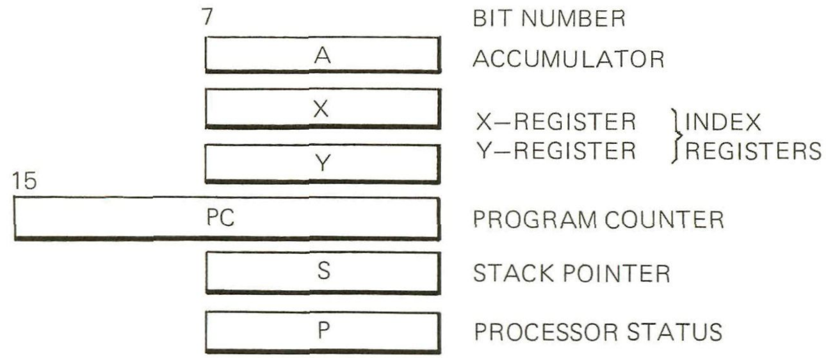
NIM IS A TRADITIONAL GAME IN WHICH THE PLAYERS ALTERNATIVELY REMOVE STICKS, OR COINS, OR WHATEVER FROM ONE OF SEVERAL STACKS. THE ONLY RULES ARE THAT YOU MUST TAKE AT LEAST ONE PIECE PER MOVE AND THAT YOU CAN ONLY REMOVE PIECES FROM ONE STACK PER MOVE. THERE IS A WELL-DEFINED STRATEGY FOR OPTIMAL PLAY BUT THIS DOES NOT GUARANTEE A WIN UNLESS THE OPPONENT MAKES A MISTAKE OR THE INITIAL SITUATION IS AGAINST HIM. THE COMPUTER PLAYS WELL BUT, WITH LUCK, CAN BE BEATEN. THE WINNER IS THE PLAYER WHO REMOVES THE LAST PIECE
IN THIS VERSION OF THE GAME THERE ARE FOUR STACKS OF FROM Ø–F PIECES. YOU MUST ENTER THE SIZE OF YOUR STACKS IN MEMORY LOCATIONS 2Ø–23 BEFORE STARTING THE GAME. THE GAME STARTS AT ~~ØØ2F~~ 0200 AND YOUR MOVE OR Ø18Ø AND THE COMPUTER'S MOVE. ON RUNNING, THE DISPLAY WILL SHOW A· B C D WHERE A,B,C,D ARE THE CONTENTS OF THE STACKS. ANY CONTROL KEY WILL MOVE THE POINTER (FULL STOP) AROUND THE STACKS. WHEN IT POINTS TO THE STACK FROM WHICH YOU WISH TO REMOVE PIECES PRESS THE KEY CORRESPONDING TO THE NUMBER YOU WISH TO REMOVE. ZERO IS ILLEGAL AND WILL NOT BE ALLOWED. IF YOU SUBTRACT MORE PIECES THAN ARE IN THE STACK THE GAME WILL GET VERY CONFUSED.
AFTER REMOVAL OF PIECES THE DISPLAY WILL SHOW THE CURRENT SITUATION AND THE COMPUTER WILL MAKE ITS MOVE.
CONTINUE UNTIL SOMEONE (SOMETHING?) WINS.
YOU MIGHT LIKE TO TRY AND WRITE SUBROUTINES TO PRINT MESSAGES ON THE DISPLAY IN THE EVENT OF EITHER A HUMAN OR COMPUTER VICTORY. A CHECK WOULD HAVE TO BE INSERTED TO DECIDE A COMPUTER WIN BUT THE JUMP FOR A HUMAN WIN IS ALREADY THERE UNDER THE MNEMONIC JMP MESSAGE, THOUGH THE CODE IN FACT JUMPS TO THE HUMAN MOVE.

NIM — NOT RELOCATABLE

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
				– CLEAR DECIMAL
Ø2ØØ	2Ø 99 Ø2	HUMMOV	JSR DSPGAP	– DISPLAY STACKS
Ø2Ø3	B5 11	SHIFTPT	LD≥X D + 1	– SET DECIMAL POINT ON
Ø2Ø5	Ø9 8Ø		ORA #8Ø	
Ø2Ø7	95 11		STA≥X D + 1	
Ø2Ø9	2Ø ØC FE	CHEAT	JSR DISPLAY	– WAIT FOR INPUT
Ø2ØC	9Ø 1Ø		BCC MINUS	
Ø2ØE	B5 11		LDA≥X D + 1	– REMOVE CURRENT DECIMAL POINT

FROM THE RIGHT) AND SUSPECT THAT IT CAN'T PUT ANYTHING ELSE THERE. THIS IS TRUE, BUT IT DOESN'T MEAN THAT THERE ISN'T A MONITOR SUBROUTINE THAT CAN DO THE JOB. SUCH A SUBROUTINE LIVES AT FE7A. IT IS DESIGNED TO PUT THE LOWEST FOUR BITS OF THE ACCUMULATOR ONTO ANY OF THE DISPLAYS, AS A READABLE CHARACTER. THIS IS JUST WHAT WE NEED – BUT HOW DO WE TELL THE SUBROUTINE WHICH DISPLAY TO USE?

3.3 THE INTERNAL REGISTERS X ANDY.
WELL, BACK TO THE μP. THIS IS WHAT IT LOOKS LIKE INSIDE



TWO NEWCOMERS, YOU'LL NOTICE! X & Y ARE 'INDEX REGISTERS', THEY WILL BE DEALT WITH MORE THOROUGHLY IN A FEW MORE PAGES, BUT WHAT MATTERS NOW IS THE USE FE7A MAKES OF THEM:
I FE7A NEITHER CARES ABOUT, NOR CHANGES X
II FE7A DOESN'T CHANGE Y, BUT THE DISPLAY IT PUTS A ONTO IS CONTROLLED BY Y THAT IS, THE LOWER 4 BITS OF A ARE TRANSFORMED INTO THE CORRECT SEQUENCE OF BITS TO REPRESENT THEIR HEXADECIMAL CHARACTER AS IT SHOULD APPEAR ON THE 7 SEGMENT DISPLAY. THEN THIS IS STORED IN MEMORY TO AWAIT THE SUBROUTINE WHICH ACTUALLY PUTS THINGS ON DISPLAY.
ALTHOUGH FE7A MAKES NO RESTRICTIONS ON THE SIZE OF Y, THE MONITOR SUBROUTINE WHICH DISPLAYS THEM ONLY KNOWS ABOUT THE FIRST 8 (NUMBERED, OF COURSE, Ø–7) OF THEM, IN LINE WITH THE ACTUAL DISPLAY HARDWARE. DISPLAY Ø IS THE LEFTMOST, DISPLAY 7 IS THE RIGHTMOST.
TO KEEP THE MONITOR AS EFFICIENT AS POSSIBLE THE SUBROUTINE AT FE6Ø USES FE7A. IT FOLLOWS THAT IT MUST HAVE LOADED Y WITH 7 & 6, AND SINCE FE7A DOESN'T CHANGE Y, Y IS STILL SET TO THE LAST USED OF THESE WHICH IS 6. SO, INSTEAD OF USING

LDY #Ø5	AØ Ø5	"LOAD Y WITH THE NEXT BYTE" (Ø5 HERE)

WE CAN USE

DEY	88	"DECREMENT (IN HEXADECIMAL) Y BY ONE"

TO SET Y TO 5, THUS SAVING A WHOLE BYTE! (BUT NO TIME, THE TWO INSTRUCTIONS ARE EXECUTED IN THE SAME TIME, 2μS). THE COMPLETE PROGRAM IS

```
ØØ2F F8          SED
ØØ3Ø 18          CLC
ØØ31 A5 21       LDA Ƶ 21
ØØ33 65 2Ø       ADC Ƶ 2Ø
ØØ35 Ø8          PHP
ØØ36 2Ø 6Ø FE    JSR FE6Ø
ØØ39 68          PLA
ØØ3A 29 Ø1       AND # Ø1
ØØ3C 88          DEY
ØØ3D 2Ø 7A FE    JSR FE7A
ØØ4Ø 4C Ø4 FF    JMP FFØ4
```

AND SO, AT LAST, WE FIND THE ANSWER TO $5Ø_{10} + 5Ø_{10}$ IS

K. ØØ2F 1ØØ

PERHAPS WE SHOULD HAVE CLEARED THE DISPLAY? OR MADE IT SHOW THE NUMBERS TO BE ADDED TOGETHER? OR ACTUALLY FETCHED THE TWO NUMBERS USING KEYBOARD AND DISPLAY LIKE THE MONITOR DOES? OR SOME COMBINATION OF THESE?

3.4 MAKING OUR PROGRAM 'FRIENDLY'

USING THE MONITOR SUBROUTINE AT FE88 IT IS EASY TO DO THE THIRD OPTION. FE88 IS THE ROUTINE WHICH FETCHES 4 DIGIT NUMBERS, TERMINATED BY ANY COMMAND KEY, INTO THE TWO BYTES IN ZERO PAGE X & X + 1 [i.e. IF X CONTAINS 2Ø, INTO ØØ2Ø (LOWBYTE = RH PAIR OF NUMBERS) & ØØ21] JUST WHAT WE NEED!

```
ØØ2A    F8          SED
ØØ2B    A2 2Ø       LDX # 2Ø
ØØ2D    2Ø 88 FE    JSR FE 88
ØØ3Ø    18          CLC
ØØ31    A5 21       LDA Z 21
ØØ33    65 2Ø       ADC Z 2Ø
ØØ35    Ø8          PHP
ØØ36    2Ø 6Ø FE    JSR FE6Ø
ØØ39    68          PLA
ØØ3A    29 Ø1       AND # Ø1
ØØ3C    88          DEY
ØØ3D    2Ø 7A FE    JSR FE7A
ØØ4Ø    4C Ø4 FF    JMP FFØ4
```

ONCE AGAIN THE PROGRAM HAS BEEN EXTENDED BACKWARDS SINCE THE GREATER PART OF IT HAS ALREADY BEEN ENTERED (UNLESS YOU'VE SWITCHED OFF AND LOST IT ALL)

RUNNING THIS PROGRAME (GØ,Ø,2,A, k) PRODUCES

K. 5Ø5Ø . (ON THE ASSUMPTION THAT ØØ2Ø & ØØ21 STILL CONTAIN THE 5Ø'S ADDED TOGETHER AS BEFORE)

THE FIRST PROGRAM, TEST, IS TRIVIAL: IT JUST SENDS A PARTICULAR BYTE TO TAPE REPETETIVELY. IT MUST BE STOPPED BY RESET. RECORD A FEW MINUTES OF THIS, THEN LOAD IT USING LOAD. DEVIATIONS FROM THE STATIONARY PATTERN ARE EASY TO SEE. THE SECOND PROGRAM, RETAG, IS RELOCATABLE. IT ACTS JUST LIKE THE MONITOR'S STORE ROUTINE, EXCEPT THAT IT ASKS FOR AN EXTRA ADDRESS. THE DATA WHICH IS STORED IS THAT STARTING AT THIS LAST ADDRESS, IT PRETENDS TO BE SITUATED BETWEEN THE FIRST TWO ADDRESSES. INCORPORATE THE REQUIRED STATE OF ZERO PAGE REGISTERS IN FRONT OF YOUR DATA, THEN 'LOAD AND AUTO RUN' PROGRAMS MAY BE CREATED.

TAPE PROGRAMS **NOT RELOCATABLE**

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
Ø2ØØ	A9 55	TEST	LDA #55	– THE TEST BYTE
Ø2Ø2	2Ø B1 FE		JSR PUTBYTE	– SEND IT
Ø2Ø5	4C ØØ Ø2		JMP TEST	– KEEP SENDING IT
Ø2Ø8	A9 F1	RETAG	LDA #F1	– F. PROMPT
Ø2ØA	85 1Ø		STA D	
Ø2ØC	A2 Ø6		LDX #Ø6	
Ø2ØE	2Ø 88 FE		JSR QDATFET	– FIRST ADDRESS
Ø211	A2 Ø8		LDX #Ø8	
Ø213	86 1Ø		STX D	– PROMPT
Ø215	2Ø 88 FE		JSR QDATFET	– SECOND ADDRESS
Ø218	A9 46		LDA #46	– PROMPT
Ø21A	85 1Ø		STA D	
Ø21C	A2 2Ø		LDX #2Ø	
Ø21E	2Ø 88 FE		JSR QDATFE7	– LAST ADDRESS: ACTUAL DATA START
Ø221	A2 Ø4		LDX #Ø4	
Ø223	B5 Ø5	ADRSS	LDA Ƶ,X Ø5	– SEND FAKE ADDRESSES
Ø225	2Ø B1 FE		JSR PUTBYTE	
Ø228	CA		DEX	
Ø229	DØ F8		BNE ADDRSS	
Ø22B	AØ ØØ	DATAS	LDY #ØØ	
Ø22D	B1 2Ø		LDA (2Ø),Y	– PROPER DATA
Ø22F	E6 2Ø		INC 2Ø	– INCREMENT PROPER DATA COUNTER
Ø231	DØ Ø2		BNE NOINC	
Ø233	E6 21		INC 21	
Ø235	2Ø B1 FE	NOI1NC	JSR PUTBYTE	– SEND DATA
Ø238	2Ø AØ FE		JSR COM16	– CHECK FAKE ADDRESSES FOR END
Ø23B	DØ EE		BNE DATAS	
Ø23D	4C Ø4 FF		JMP RESTART	
Ø24Ø				

THE SCROLL PROGRAM SHIFTS THE WHOLE DISPLAY ONE LEFT, AND ENTERS THE NEW INFORMATION, IN A, ON THE FAR RIGHT.

SCROLL

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
Ø2ØØ	A2 ØØ		LDX #ØØ	– MUST GO FORWARDS
Ø2Ø2	B4 11	LOOP	LDY ƵX D + 1	– PICK-UP DATA ON RIGHT
Ø2Ø4	94 1Ø		STY ƵX D	– & MOVE IT ONE LEFT
Ø2Ø6	E8		INX	

```
Ø25A   ØØ 71 77
Ø25D   5Ø ØØ
Ø25F
```

THE RELOCATOR FIRST FETCHES THE THREE ADDRESSES IT REQUIRES, THE ADDRESSES OF THE START & END OF THE MEMORY SECTION TO BE MOVED, AND THE ADDRESS OF THE START OF THE AREA TO WHICH THE MOVE IS TO TAKE PLACE. THE PROMPTS ARE F., & **t** RESPECTIVELY. AFTER TERMINATING THE LAST ADDRESS, THE MOVE TAKES PLACE. MOVES UP BY LESS THAN THE LENGTH OF THE MATERIAL TO BE USED WILL NOT BE SUCCESSFUL (I.E. **t – F** ., IF POSITIVE, SHOULD BE GREATER THAN **–t**)

RELOCATOR

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
Ø2ØØ	A2 F1		LDX #F1	
Ø2Ø2	86 1Ø		STX Ƶ D	– SET UP FROM PROMPT F.
Ø2Ø4	A2 2Ø		LDX #2Ø	
Ø2Ø6	2Ø 88 FE		JSR QDATFET	– AND GET ADDRESS
Ø2Ø9	A2 46		LDX #46	
Ø2ØB	86 1Ø		STX Ƶ D	– SET UP END PROMPT
Ø2ØD	A2 22		LDX #22	
Ø2ØF	2Ø 88 FE		JSR QDATFET	– AND GET SECOND ADDRESS – MOVE THE DATA BETWEEN THESE ADDRESSES
Ø212	A2 78		LDX #78	
Ø214	86 1Ø		STX Ƶ D	– SET UP TO PROMPT
Ø216	A2 24		LDX #24	
Ø218	2Ø 88 FE		JSR QDATFET	– AND GET BASE ADDRESS – MOVE TO HERE & SUCCESSIVE LOCATIONS
Ø21B	A2 1A		LDX #1A	
Ø21D	A1 Ø6	MOVE	LDA (Ø6,X)	– DO THE MOVE
Ø21F	91 24		STA (24,Y)	
Ø221	C8		INY	– INCREMENT THE TO ADDRESS
Ø222	DØ Ø2		BNE NOINC	
Ø224	E6 25		INC Ƶ 25	
Ø226	2Ø AØ FE	NOINC	JSR COM16	– USE COM16 TO DO THE LIMIT TEST
Ø229	DØ F2		BNE MOVE	
Ø22B	4C Ø4 FF		JMP RESTART	
Ø22D				

YOU SHOULD ENTER THE TWO PAIRS OF NUMBERS YOU WISH ADDED TOGETHER AS IF THEY FORMED AN ADDRESS. TERMINATING YOUR ENTRY WITH k INSTANTLY PRODUCES THE RESULT

```
K.   5Ø5Ø    1ØØ
```

LOOKING BACK OVER THE PROGRAM, AND EXAMINING THE MONITOR LISTING, WILL REVEAL THAT IT TOOK AD_{16} (OR 173_{10}) BYTES OF CODE TO ACHIEVE THIS. THE ACTUAL OPERATION USED 6 BYTES OF CODE (SED; CLC; LDA Z; ADC Z) WHILE THE OTHER 167_{10} ARE THERE 'MERELY' TO DISPLAY THE RESULT & FETCH THE INFORMATION NEATLY (THE CODE CALCULATIONS DO NOT CONSIDER THE 16_{10} BYTES OF CHARACTER FONT OR THE 11_{10} BYTES OF TEMPORARY STORAGE ALSO USED)

CHAPTER 4: THE REMAINDER OF THE INSTRUCTION SET

4.1 BRANCHES

THINKING ABOUT THE FE88 PROGRAM, YOU SHOULD REALIZE THAT IT DOES SOMETHING OF THE FORM

FETCH NEXT KEY
<u>IF</u> KEY IS A COMMAND KEY <u>THEN</u> RETURN

THIS IS A CONDITIONAL TRANSFER OF CONTROL AND REPRESENTS SOME NEW INSTRUCTIONS AND A DIFFERENT WAY OF CHANGING THE PROGRAM COUNTER. AN OPERATION LIKE ADC DOES MORE THAN ADDING TWO BYTES AND THE CARRY FLAG TOGETHER AND OUTPUTTING A CARRY. IT ALSO SETS SOME OF THE OTHER FLAGS IN P:

THE Ƶ FLAG IS SET IF THE RESULTING BYTE WAS ZERO
THE V FLAG IS SET IF THERE WAS A 2'S COMPLEMENT OVERFLOW
THE N FLAG IS SET IF THE RESULT WAS A NEGATIVE 2'S COMPLEMENT NUMBER – I.E. BECOMES BIT 7 OF THE RESULT.

THESE FLAGS ARE ABLE TO CAUSE CONDITIONAL TRANSFER BY USING THE APPROPRIATE ONE OF THE EIGHT 'BRANCH' INSTRUCTIONS. THE MECHANISM EMPLOYED IS TO PERFORM A 2'S COMPLEMENT ADD BETWEEN THE PROGRAM COUNTER AND THE SECOND BYTE OF THE BRANCH INSTRUCTION THUS PERMITTING THE TRANSFER TO BE –128. . . +127 BYTES FROM THE NEXT INSTRUCTION. THIS IS CALLED 'RELATIVE ADDRESSING' AND IS A POSITION INDEPENDENT METHOD OF TRANSFER, THE EIGHT BRANCH INSTRUCTIONS ARE ASSOCIATED TWO TO EACH OF THE C, Ƶ, V & N FLAGS, ONE OF WHICH BRANCHES IF THE FLAG IS SET, THE OTHER BRANCHES IF IT IS CLEAR.

TO CLARIFY THIS LET'S LOOK AT AN EXAMPLE:

```
*+Ø   ┌──BCS Ø3      "BRANCH IF CARRY SET"
*+2   │  SEC         SET CARRY
*+3   ↓ ┌BCS Ø1
*+5   └→↓ CLC        CLEAR CARRY
*+6     └→. . . .
```

(THE ARROWS ARE PUT IN FOR CLARITY)

WE'LL NEED TO CONSIDER THIS PROGRAM BOTH WITH THE CARRY SET & WITH IT CLEAR

I CARRY IS CLEAR

INSTRUCTION I DOES NOT TRANSFER CONTROL SO WE DO INSTRUCTION II, SEC, NOW INSTRUCTION III TRANSFERS CONTROL SINCE THE CARRY IS NOW SET. Ø1 IS ADDED TO THE PC (= * + 5) TO GIVE* + 6 AS THE ADDRESS OF THE NEXT INSTRUCTION.

II CARRY IS SET
INSTRUCTION I TRANSFERS CONTROL. Ø3 IS ADDED TO THE PC (= * + 2) TO GIVE * + 5 AS THE ADDRESS OF THE NEXT INSTRUCTION, INSTRUCTION IV. CLC.

SO IF THE CARRY WAS CLEAR IT IS SET; IF IT WAS SET IT IS CLEARED, SO THE PROGRAM COMPLEMENTS THE CARRY (THERE ARE QUICKER METHODS, INDEED IT CAN BE DONE WITH 3 INSTRUCTIONS IN 4 BYTES).

AND WE CAN GO BACKWARDS:

```
* + Ø    BCS   Ø3    BRANCH IF CARRY SET
* + 2    SEC         SET CARRY
* + 3    BCS   FB    BRANCH IF CARRY SET
* + 5    CLC         CLEAR CARRY
* + 6    . . . .
```

IF THE CARRY IS SET THE PROGRAM IS AS BEFORE IF IT IS CLEARED WE SET IT & BRANCH FB

```
( 2's COMPLEMENT ADD     * + 5      )
(                          FB    +  )
(                        * + Ø      )
```

—BACK TO THE BEGINNING. A RATHER COMPLICATED WAY OF CLEARING THE CARRY.

MOST OF THE NON-BRANCH INSTRUCTIONS WILL CHANGE SOME OF THESE 4 TESTABLE FLAGS, USUALLY THE N & Ƶ FLAGS SINCE THEY CONSTANTLY MONITOR THE STATUS OF OPERANDS SO BRANCHES WILL APPEAR RATHER FREQUENTLY IN PROGRAMS.

4.2 INDEXING

IF YOU WISHED TO CLEAR (SET EACH BYTE TO Ø) A PATCH OF MEMORY, e.g. THE MEMORY USED TO STORE THE DATA WHICH IS TO BE OUTPUT TO THE DISPLAYS, WHICH IS FROM ØØ1Ø TO ØØ17, YOU MIGHT THINK

```
LDA# ØØ     LOAD ACCUMULATOR IMMEDIATE WITH ØØ
STA Ƶ 1Ø    STORE ACCUMULATOR IN ADDRESS ØØ1Ø
STA Ƶ 11    STORE ACCUMULATOR IN ADDRESS ØØ11
STA Ƶ 12    STORE ACCUMULATOR IN ADDRESS ØØ12
   :
STA Ƶ 17    STORE ACCUMULATOR IN ADDRESS ØØ17
```

IS NECESSARY. THIS LOOKS SUFFICIENTLY REGULAR THAT THE COMPUTER SHOULD BE ABLE TO DOT IT. THIS IS WHERE THE INDEX REGISTERS REAPPEAR. WE CAN STORE THE ACCUMULATOR INDEXED BY EITHER INDEX REGISTER

```
STA Ƶ,X     95     "STORE A INDEXED BY X IN ZERO
                    PAGE"
STA Ƶ,X 1Ø
```

THE OFFSET CALCULATOR CALCULATES THE OFFSET TO BE ENTERED AS THE SECOND BYTE OF A BRANCH INSTRUCTION. IT WILL PROMPT WITH XXØØØØXX AND YOU SHOULD ENTER THE ADDRESS OF THE BRANCH INSTRUCTION. AFTER A CONTROL KEY IT WILL PROMPT AGAIN WITH XX1111XX AND YOU SHOULD ENTER THE ADDRESS YOU WISH TO BRANCH TO. THE REPLY WILL BE EITHER "OFFSET XX" WHERE XX IS THE VALUE TO BE ENTERED, OR "TOO FAR" IF THAT IS THE CASE. A CONTROL KEY RESTARTS THE SEQUENCE.

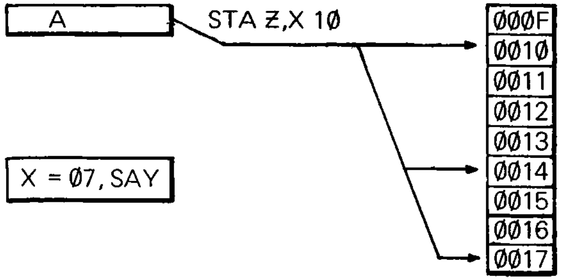
OFFSET CALCULATOR **NOT RELOCATABLE**

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
Ø2ØØ	D8		CLD	
Ø2Ø1	A9 Ø2	AGAIN	LDA #Ø2	
Ø2Ø3	85 21		STA MESSH	– INITIALIZE MESSAGE POINTER
Ø2Ø5	84 22		STY FROMH	– SET UP PROMPT
Ø2Ø7	84 23		STY FROML	
Ø2Ø9	A2 22		LDX #FROML	
Ø2ØB	2Ø 88 FE		JSR QDATFET	– FETCH FIRST ADDRESS
Ø2ØE	A9 11		LDA #11	– SET UP 2ND PROMPT
Ø21Ø	85 24		STA TOL	
Ø212	85 25		STA TOH	
Ø214	A2 24		LDX #TOL	
Ø216	2Ø 88 FE		JSR QDATFET	– FETCH SECOND ADDRESS
Ø219	A5 22		LDA FROML	– OFFSET TO MAKE OVERLENGTH EASY
Ø21B	E9 7E		SBC #7E	– CARRY KNOWN SET BY QDATFET
Ø21D	85 22		STA FROML	
Ø21F	BØ Ø3		BCS HSUB	– DON'T SET THE CARRY AGAIN!
Ø221	C6 23		DEC FROMH	
Ø223	38		SEC	
Ø224	A5 24	HSUB	LDA TOL	– CALCULATE THE LENGTH
Ø226	E5 22		SBC FROML	
Ø228	AA		TAX	
Ø229	A5 25		LDA TOH	
Ø22B	E5 23		SBC FROMH	
Ø22D	DØ ØE		BNE TOOFAR	
Ø22F	A9 51		LDA #51	
Ø231	2Ø 44 Ø2		JSR MESSAGE	– PRINT OUT
Ø232	8A		TXA	
Ø235	49 8Ø		EOR #8Ø	– COMPLEMENT TOP BIT BECAUSE OF THE OFFSET APPLIED
Ø236	2Ø 6Ø FE		JSR RDHEXTD	– PRINT OUT ANSWER, OVER WRITING THE
Ø239	4C Ø4 FF		JMP RESTART	– FINISHED
Ø23C	A9 57	TOOFAR	LDA #57	– WHOOPS
Ø23E	2Ø 44 Ø2		JSR MESSAGE	– TELL THE PROGRAMMER THAT IT'S WRONG
Ø241	4C Ø1 Ø2		JMP AGAIN	– AND GET IT DONE AGAIN
Ø244	85 2Ø	MESSAGE	STA MESSL	– MESSAGE DESCRIBED BY A
Ø246	AØ Ø7		LDY #Ø7	– EIGHT BYTES OF DATA TO DISPLAY
Ø248	B1 2Ø	LOOP	LDA (MESSL), Y	– FETCH THEM
Ø24A	99 1Ø ØØ		STA D, Y	
Ø24D	88		DEY	
Ø24E	1Ø F8		BPL LOOP	
Ø25Ø	6Ø		RTS	
Ø251	5C 71 71			– THE DATA
Ø254	ED 79 78			
Ø257	78 5C 5C			

THE HEXADECIMAL TO DECIMAL CONVERTER PROMPTS WITH XXØØØØXX AND AFTER A CONTROL KEY IS PRESSED WILL PROVIDE AN ANSWER IN THE FORM ????????, AFTER A WAIT!
THE PROGRAM WORKS BY DECREMENTING THE HEX. NUMBER AND INCREMENTING THE DECIMAL NUMBER UNTIL THE HEX. NUMBER REACHES ZERO.
THIS PROGRAM, LIKE THE DECIMAL TO HEX. CONVERTER, WHICH USES VIRTUALLY THE SAME METHOD, ILLUSTRATES THE USE OF THE DECIMAL MODE, AN IMPORTANT FACET OF THIS PROCESSOR.
THEY ALSO PROVIDE AN EXCELLENT DEMONSTRATION OF THE TRADEOFF FREQUENTLY FOUND BETWEEN PROGRAM LENGTH AND SIMPLICITY, AND PROGRAM EXECUTION TIME. THE METHOD USED IS BOTH SHORT AND SIMPLE, BUT CAN TAKE UP TO THREE SECONDS FOR SOME CALCULATIONS. A MUCH LONGER AND MORE COMPLEX (RELATIVELY) PROGRAM COULD HAVE BEEN WRITTEN BASED ON ABCD = A(16*16*16)+B(16*16)+C(16)+D AND WOULD HAVE BEEN VIRTUALLY INSTANTANEOUS.

HEX → DEC

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
Ø2ØØ	84 2Ø		STY Ƶ HEXL	– SET UP ZERO PROMPT
Ø2Ø2	84 21		STY Ƶ HEXH	–
Ø2Ø4	A2 2Ø		LDX #HEXL	
Ø2Ø6	2Ø 88 FE		JSR Q DATFET	– AND FETCH THE DATA
Ø2Ø9	F8		SED	– DECIMAL MODE
Ø2ØA	A2 ØØ		LDX #ØØ	– SET X & Y & DECOUT TO ZERO
Ø2ØC	86 22		STX Ƶ DECOUT	
Ø2ØE	A5 2Ø	DECRHEX	LDA Ƶ HEXL	– TEST FOR ZERO, THEN DECREMENT
Ø21Ø	DØ Ø6		BNE NODEL	
Ø212	A5 21		LDA Ƶ HEXH	
Ø214	FØ 13		BEQ DEAD	– IF HEX NO. IS ZERO, THEN FINISHED
Ø216	C6 21		DEC Ƶ HEXH	
Ø218	C6 2Ø	NODEC	DEC Ƶ HEXL	
Ø21A	18		CLC	}
Ø21B	98		TYA	
Ø21C	69 Ø1		ADC #Ø1	
Ø21E	A8		TAY	
Ø21F	8A		TXA	ADD 1 TO THE DECIMAL NUMBER,
Ø22Ø	69 ØØ		ADC #ØØ	USING X & Y AS TWO BYTE
Ø222	AA		TAX	ACCUMULATOR
Ø223	9Ø E9		BCC DECRHEX	
Ø225	E6 22		INC Ƶ DECOUT	
Ø227	BØ E5		BCS DECRHEX	}
Ø229	84 2Ø	DEAD	STY Ƶ HEXL	– FINISHED, SO STORE X & Y
Ø22B	86 21		STX Ƶ HEXH	
Ø22D	A2 2Ø		LDX #HEXL	
Ø22F	2Ø 64 FE		JSR QHEXTD1	– DISPLAY 4 DIGITS
Ø232	88		DEY	
Ø233	A5 22		LDA Ƶ DECOUT	
Ø235	2Ø 7A FE		JSR HEXTD	– DISPLAY 5 DIGIT
Ø238	4C Ø4 FF		JMP RESTART	
Ø23A				



A IS STORED IN 17 WHICH IS 1Ø THE "BASE ADDRESS" +Ø7 THE "INDEX"

IF WE DO
```
A2 Ø7      LDX # Ø7
95 1Ø      STA Ƶ,X 1Ø
```
THE STORE IS TO LOCATION 17 (=1Ø + X). THE ADDITION IS STRAIGHT-FORWARD BINARY, TRUNCATED TO A LOCATION IN ZERO PAGE SO
```
           LDX # FF
           STA Ƶ,X 1Ø
```
STORES IN LOCATION ØF
WE ALSO HAVE
```
           STA, X     9D       "STORE A INDEXED BY X"
           STA, Y     99       "STORE A INDEXED BY Y"
```
(BUT NO STA Ƶ, Y) WHICH DO NOT NEED TO TRUNCATE THE ADDITION THEY EXPECT A TWO BYTE ADDRESS SO
```
           LDX # FF
           STA, X ØØ1Ø
```
STORES IN LOCATION Ø1ØF
NOW
```
           DEX        CA       "DECREMENT (IN HEX) X BY ONE"
```
SETS THE Ƶ FLAG IF X IS ZERO, & THE N FLAG EQUAL TO BIT 7 OF X.
```
           BPL        1Ø       "BRANCH IF PLUS"
```
TAKES THE BRANCH IF THE N FLAG IS CLEAR I.E. IS SAYING 'NOT NEGATIVE' I.E. PLUS. IT'S EASY TO SEE THAT THE COMBINATION
```
           DEX  <--+
           BPL FD -+
```
DECREMENTS X ONCE, AND, IF THE RESULT WAS POSITIVE (I.E. IN THE RANGE Ø – 7F) IT TAKES THE BRANCH AND DECREMENTS X AGAIN. . . . AND AGAIN UNTIL IT REACHES A NON-POSITIVE NUMBER, WHICH WILL BE FF, WHEN IT DOESN'T TAKE THE BRANCH. IF WE START AT 7 AND EACH TIME AROUND THE LOOP CLEAR THE RELEVANT DISPLAY:

CODE	LABEL	MNEMONICS	COMMENT
A9 ØØ		LDA # ØØ	LOAD ACCUMULATOR IMMEDIATE
A2 Ø7		LDX # Ø7	LOAD X IMMEDIATE
95 1Ø	LOOP:	STA Ƶ, X 1Ø ←	STORE A IN ZERO PAGE INDEXED BY X
CA		DEX	DECREMENT X BY ONE
1Ø FB		BPL LOOP ←	BRANCH IF PLUS TO "LOOP"

SO WE CAN WRITE A VERY SHORT PROGRAM TO CLEAR THE DISPLAY. BY MAKING THE LOOP SLIGHTLY LARGER (WITH THE SAME LENGTH OF PROGRAM)

```
0060 A2 07            LDX #07
0062 B5 48      LOOP: LDA Z, X 48
0064 95 10            STA Z,X 10
0066 CA               DEX
0067 10 F9            BPL LOOP
0069 4C 04 FF         JMP FF04
```

WE CAN, INSTEAD OF CLEARING THE DISPLAY, CAUSE A BLOCK OF MEMORY, 0048 – 004F, TO BE TRANSFERRED TO THE DISPLAY. THE PROGRAM IS POSITION INDEPENDENT SO YOU CAN WRITE IT INTO MEMORY ANYWHERE... EXCEPT LOCATIONS 0010 – 0017. IF YOU PUT THE PROGRAM IN 0048 .... IT WILL FUNCTION PERFECTLY BUT YOU WON'T BE ABLE TO CHANGE THE DATA WHICH IS MOVED, SINCE THIS IS THE PROGRAM. YOU CAN TRY THE PROGRAM USING THIS DATA

```
0048     00 77 58 5C 50 54 00 00
```

OR YOU COULD CONSTRUCT YOUR OWN DATA, USING APPENDIX A.

THE INDEXING MECHANISM SHOWN ABOVE IS ONLY CAPABLE OF DEALING WITH 256 (CONSECUTIVE) BYTES, STARTING AT A GIVEN ADDRESS. THUS

```
A9 00                 LDA #00        LOAD A IMMEDIATE WITH "00"
A8                    TAY            TRANSFER A TO Y
18              LOOP: CLC            CLEAR CARRY
79 00 FE              ADC, Y FE00    ADD WITH CARRY INDEXED BY Y
C8                    INY            INCREMENT Y
D0 F9                 BNE LOOP       BRANCH IF NOT EQUAL
20 60 FE              JSR FE60       JUMP SUBROUTINE
4C 04 FF              JMP FF04       JUMP
```

COMPUTES THE LOWEST BYTE OF THE 256 BYTE ADDITION. (NOTE THAT, SINCE Y IS ZERO WHEN YOU LEAVE THE MONITOR BY THE GO FUNCTION, THE INITIALISATION OF A & Y CAN BE ACCOMPLISHED BY TYA INSTEAD OF LDA # 00, TAY) HOW COULD THIS BE DONE FOR ALL 65536 MEMORY BYTES? CLEARLY IT IS POSSIBLE TO HAVE AN ADC, Y FOR EACH PAGE:

```
98                    TYA
18              LOOP: CLC
79 00 00              ADC, Y 0000   }
18                    CLC           }
79 00 FF              ADC, Y 0100   }      CLC
:                     :             }  256 ADC, Y INSTRUCTION PAIRS
18                    CLC           }
79 00 FF              ADC, Y FF00   }
C8                    INY
F0 03                 BEQ END
4C ? ?                JMP LOOP
20 60 FE END          JSR FE60
4C 04 FF              JMP FF04
```

IN ORDER TO SHORTEN THIS PROGRAM WE WILL INTRODUCE THE CONCEPT OF "INDIRECTION".

```
ADDR  HEX         LABEL    INSTRUCTION     COMMENTS
      CODE
0240  A2 24                LDX #24         – SET UP X
0242  20 66 FE             JSR QHEXTD2     – PUT NEXT 4 OUT
0245  4C 04 FF             JMP RESTART     – DISPLAY RESULT
0247
```

SYSTEM

THE DECIMAL TO HEX CONVERTER WILL PROMPT WITH 0XXXX FOR THE FIRST DIGIT OF THE 5 DIGIT DECIMAL NUMBER. THEN X0000. FOR THE LAST FOUR DIGITS OF THE DECIMAL NUMBER. CLEARLY ANYTHING OVER 65535 WILL GIVE THE REMAINDER WHEN DIVIDED BY 10000 HEX. TO ENTER THIS NUMBER YOU WOULD KEY 6, CONTROL KEY, 5535, CONTROL KEY, AND FFFF WILL APPEAR ON THE DISPLAY (AFTER A SLIGHT DELAY!)

THE PROGRAM WORKS BY A PROCESS OF DECREMENTING THE DECIMAL NUMBER AND THEN INCREMENTING THE HEX. NUMBER.

DEC→HEX

```
0200  98                   TYA             – CLEAR A
0201  85 20                STA Z DECL      – CLEAR NO
0203  85 21                STA Z DECH
0205  A2 20                LDX #DECC
0207  85 22       AGAIN    STA Z DECVH     }
0209  20 7A FE             JSR HEXTD       } – FETCH THE FIRST DIGIT
020C  20 0C FE             JSR DISPLAY     }
020F  90 F6                BCC AGAIN       }
0211  20 88 FE             JSR QDATFET     – AND THEN THE LAST FOUR DIGITS
0214  F8                   SED             – DECIMAL MODE
0215  84 10                STY Z D         – CLEAR LEFT DISPLAY
0217  A6 21                LDX Z DECH      – X & Y AS DOUBLE ACCUMULATOR
0219  98                   TYA
021A  85 21                STA Z DECH      – CLEAR AREA FOR RESULT
021C  A4 20                LDY Z DECL
021E  85 20                STA Z DECL
0220  38          NEXT     SEC             }
0221  98          ALSO     TYA             }
0222  E9 01                SBC #01         } – DO A DECIMAL SUBTRACT, DOUBLE
                                           }   BYTE
0224  A8                   TAY             }
0225  8A                   TXA             }
0226  E9 00                SBC #00         }
0228  AA                   TAX
0229  B0 04                BCS NODEC
022B  C6 22                DEC Z DECVH     – LAST OF THE DECIMAL SUBTRACT,
                                             TO DO 5 DIGITS
022D  30 09                BMI RESULT      – IF MINUS THEN FINISHED
022F  E6 20       NODEC    INC Z DECL      – DOUBLE HEX INCREMENT
0231  D0 ED                BNE NEXT
0233  E6 21                INC Z DECH
0235  38                   SEC             } – CREATE BRANCH ALWAYS, BUT
0236  B0 E9                BCS ALSO        }   DON'T BOTHER TO SET THE CARRY
                                               TWICE
0238  A2 20       RESULT   LDX #20
023A  20 64 FE             JSR QHEXTD      – DISPLAY RESULT
023D  4C 04 FF             JMP RESTART
023F
```

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
Ø2Ø8	A2 2Ø		LDX #2Ø	
Ø2ØA	2Ø 88 FE		JSR QDATFET	– FETCH THE NUMBERS
Ø2ØD	98		TYA	– CLEARS A
Ø2ØE	AØ Ø8		LDY #Ø8	– LOOP COUNTER
Ø21Ø	66 2Ø	LOOP	ROR Ƶ 2Ø	– SHIFT MULIPLIER (AND HIGH BYTE OF RESULT)
Ø212	9Ø Ø3		BCC NAD	– NO ADD IF NO BIT
Ø214	18		CLC	
Ø215	65 21		ADC Ƶ 21	– ADD MULTIPLICAND INTO LOW BYTE OF RESULT
Ø217	6A	NAD	ROR A	– AND SHIFT LOW BYTE OF RESULT
Ø218	88		DEY	
Ø219	DØ F5		BNE LOOP	
Ø213	85 21		STA Ƶ 21	– PUT IN LOW BYTE
Ø21D	66 2Ø		ROR Ƶ 2Ø	– FINAL JUSTIFICATION SHIFT
Ø21F	2Ø 64 FE		JSR QHEXTD	– DISPLAY ANSWER
Ø222	2Ø 64 FF		JMP RESTART	
Ø224				

DOUBLE BYTE MULTIPLY

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
Ø2ØØ	D8		CLD	– BINARY ONLY
Ø2Ø1	84 2Ø		STY Ƶ 2Ø MPIER	– FORM PROMPT FOR THE ZERO INPUT
Ø2Ø3	84 21		STY Ƶ 21	
Ø2Ø5	A9 11		LDA #11	
Ø2Ø7	85 22		STA Ƶ 22 MPICAND	– FORM PROMPT FOR THE FIRST INPUT
Ø2Ø9	85 23		STA Ƶ 23	
Ø2ØB	A2 2Ø		LDX #2Ø	
Ø2ØD	2Ø 88 FE		JSR QDATFET	– FETCH ZERO INPUT
Ø21Ø	A2 22		LDX #22	
Ø212	2Ø 88 FE		JSR QDATFET	– AND FIRST INPUT
Ø215	84 24		STY Ƶ 24	– CLEAR WORKING SPACE
Ø277	84 25		STY Ƶ 25	
Ø219	AØ 1Ø		LDY #1Ø	– LOOP COUNT INITIALISATION
Ø21B	66 23	LOOP	ROR Ƶ 23	– TWO BYTE SHIFT RIGHT
Ø21D	66 22		ROR Ƶ 22	
Ø21F	9Ø ØD		BCC NAD	– NO ADD IF THE O/P BIT ISN'T A ONE
Ø221	18		CLC	
Ø222	A5 2Ø		LDA Ƶ 2Ø	– TWO BYTE ADD
Ø224	65 24		ADC Ƶ 24	
Ø226	85 24		STA Ƶ 24	
Ø228	A5 21		LDA Ƶ 21	
Ø22A	65 25		ADC Ƶ 25	
Ø22C	85 25		STA Ƶ 25	– NO CARRY OUT OF THE ADD
Ø22E	66 25	NAD	ROR Ƶ 25	– SHIFT AGAIN
Ø23Ø	66 24		ROR Ƶ 24	
Ø232	88		DEY	
Ø233	DØ E6		BNE LOOP	– GO ROUND LOOP 16 TIMES
Ø235	66 23		ROR Ƶ 23	– FINAL SHIFT ON RESULT
Ø237	66 22		ROR Ƶ 22	
Ø239	AØ Ø6		LDY #Ø6	– SET UP POSITION
Ø23B	2Ø 66 FE		JSR QHEXTD2	– X ALREADY POINTING AT CORRECT LOCATIONS – PUT 4 HEX OUT
Ø23E	AØ Ø2		LDY #Ø2	– NEXT POSITION

4.3.INDIRECTION:

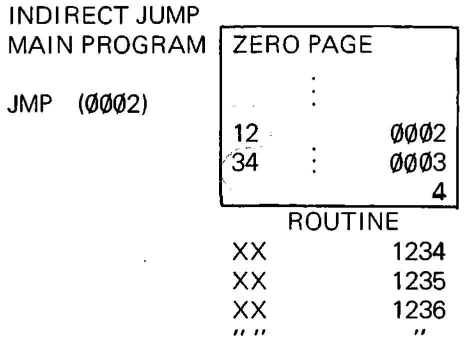
YOU'LL NOTICE THAT THE PROGRAM IS NOT POSITION INDEPENDENT: THE ADDRESS OF THE CLC INSTRUCTION MUST BE WRITTEN INTO THE PROGRAM. THIS IS ANOTHER DISADVANTAGE OF THIS METHOD: (THERE IS AN ADVANTAGE: THIS PROGRAM IS VERY FAST, TAKING ONLY 6μS PER BYTE). THE INSTRUCTION REQUIRED MUST HAVE A 16 BIT UNFIXED ADDRESS AND THIS CAN ONLY GO IN ONE PLACE : MEMORY. A LIMITATION IS THAT GENERALLY IT CAN ONLY BE IN ZERO PAGE MEMORY. THE CONCEPT IS KNOWN AS INDIRECTION. THE MOST DIRECT VERSION OF THIS IS THE INDIRECT JUMP.

6C Ø2 ØØ JMP (ØØØ2)

THIS IS THE ONE VERSION OF INDIRECTION THAT DOESN'T NEED TO REFER TO ZERO PAGE MEMORY. WHAT HAPPENS IS THIS:

TIME, μS	ADDRESS BUS	DATA BUS	R/W	
Ø	PC	6C	1	JUMP INDIRECT
1	PC+1	Ø2	1	
2	PC+2	ØØ	1	
3	ØØØ2	V	1	LOWER BYTE
4	ØØØ3	U	1	HIGHER BYTE
5	UV	OPCODE	1	OLD 6C COMPLETED

THE MONITOR USES A JUMP INDIRECT FOR THE GO FUNCTION, HAVING BUILT THE ADDRESS IN ØØØ2 & ØØØ3 : A JUMP INDIRECT VIA ØØØ2 & ØØØ3, ASSUMING THAT THESE LOCATIONS HAVEN'T BEEN ALTERED, WILL THUS RETURN TO THE START OF THE PROGRAM – WITHOUT KNOWING WHERE IT HAD BEEN ENTERED INTO MEMORY AT THE TIME OF WRITING.



WELL, THAT WAS SIMPLE INDIRECTION. NOW WE'LL MOVE ONTO THE MORE COMPLICATED MODES OF INDIRECTION. HAVING FETCHED THE ADDRESS OUT OF MEMORY WITH THE INDIRECTION STAGE, WE CAN INDEX IT. THIS IS CALLED POST-INDEXED INDIRECTION. WITH THE 65XX SERIES OF MICRO-PROCESSORS YOU MAY ONLY

- I INDEX IN THIS MODE WITH THE Y INDEX REGISTER
- II USE ZERO PAGE MEMORY

TIME, µS	ADDRESS BUS	DATA BUS	R/W	
Ø	PC	B1	1	LDA (I),Y
1	PC+1	I	1	
2	ØØI	J	1	
3	ØØI+1	K	1	
4	KJ+Y	DATA	1	(AN EXTRA µS IS NEEDED IF J+Y
5	PC+2	OPCODE	1	RESULTS IN A CARRY)

THIS IS THE MODE OF ADDRESSING NEEDED TO SOLVE THE 65536 BYTE ADDITION PROBLEM. MEANWHILE WHAT ABOUT THE X REGISTER AND INDIRECTION? HERE WE HAVE PRE-INDEXED INDIRECTION

TIME, µS	ADDRESS BUS	DATA BUS	R/W	
Ø	PC	A1	1	LDA (I,X)
1	PC+1	I	1	
2	ØØI	DATA, DISCARDED	1	
3	ØØI+X	J	1	NO CARRY TO HIGH ORDER BYTE
4	ØØI+X+1	K	1	
5	KJ	DATA	1	PUT I N A
6	PC+2	OP CODE	1	

THIS IS THE OPPOSITE TO POST-INDEXED ... HERE THE INDEXING SWITCHES BETWEEN DIFFERENT INDIRECTION LOCATIONS. THE EFFECTS OF THESE TWO INDEXING MODES ARE ONLY THE SAME IN THE TRIVIAL CASE OF ZERO INDEXES. HERE IS THE SOLUTION TO THE 65536 BYTE ADDITION:

```
98               TYA           —ZERO Y & A
85 2Ø            STA Z 2Ø     }
85 21            STA Z 21     } SET UP INDIRECT LOCATIONS
18        LOOP   CLC
71 2Ø            ADC (2Ø), Y
C8               INY
DØ FA            BNE LOOP
E6 21            INC Z 21
DØ F6            BNE LOOP
2Ø 6Ø FE         JSR FE6Ø
4C Ø4 FF         JMP FFØ4
```

THE PROGRAM IS, ONCE AGAIN, POSITION INDEPENDENT. IT IS, AS IMPLIED IN THE FIRST SOLUTION, SLOW : 12µS PER BYTE. THIS IS MAINLY DUE TO THE SMALL SIZE OF THE LOOP : THE 3µS 'NEARLY ALWAYS TAKEN' BRANCH IS TAKING A DISPROPORTIONATE AMOUNT OF TIME, IN THE FIRST SOLUTION THE EQUIVALENT 5µS BRANCH AND JUMP COMBINATION OCCURS ONLY EVERY 256 BYTES AND IS THUS IGNORED IN THE TIME CALCULATIONS.
THE INSTRUCTION INC Z 21 HAS AN OBVIOUS FUNCTION : INCREMENT (IN HEXADECIMAL) LOCATION ØØ21. IT ACTS JUST LIKE INX OR INY — BUT IT TAKES 5µS INSTEAD OF 2µS.

4.4 READ-MODIFY WRITE INSTRUCTIONS

THERE ARE COMPANION INSTRUCTIONS TO INC Ƶ THAT CAN DIRECTLY ALTER MEMORY CONTENTS, THESE ARE CALLED READ-MODIFY-WRITE INSTRUCTIONS, THE NEXT OF WHICH IS THE OBVIOUS DEC INSTRUCTION.

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
Ø217	A4 2Ø		LDY Ƶ 2Ø	— USE Y & X AS DOUBLE ACCUMULATOR
Ø219	A6 21		LDX Ƶ 21	
Ø21B	38		SUB SEC	
Ø21C	98		TYA	
Ø21D	E5 22		SBC Ƶ 22	— SUBTRACT THE DIVISOR
Ø21F	A8		TAY	
Ø22Ø	8A		TXA	
Ø221	E9 ØØ		SBC #ØØ	
Ø223	AA		TAX	
Ø224	9Ø 1Ø		BCC RESULT	— IF NEGATIVE THEN FINISHED
Ø226	84 23		STY Ƶ 23	— ELSE UPDATE THE REMAINDER
Ø228	A5 24		LDA Ƶ 24	
Ø22A	69 ØØ		ADC #ØØ	
Ø22C	85 24		STA Ƶ 24	— AND ADD ONE TO THE RESULT (CARRY WAS SET ON INPUT).
Ø22E	A5 25		LDA Ƶ 25	
Ø23Ø	69 ØØ		ADC #ØØ	
Ø232	85 25		STA Ƶ 25	
Ø234	9Ø E5		BCC SUB	— NO CARRY IS POSSIBLE (USUALLY)
Ø236	A2 24	RESULT	LDX #24	
Ø238	2Ø 64 FE		JSR QHEXTDI	— DISPLAY RESULT
Ø23B	A5 23		LDA Ƶ 23	
Ø23D	2Ø 6Ø FE		JSR RDHEXTD	— AND REMAINDER
Ø24Ø	4C Ø4 FF		JMP RESTART	
Ø242				

THE TWO MULTIPLY ROUTINES ARE FOR SINGLE AND DOUBLE BYTE BINARY MULTIPLICATION. THE FIRST PROMPTS XXØØ11XX AND THE TWO NUMBERS TO BE MULTIPLIED SHOULD BE ENTERED SEQUENTIALLY. (E.G. 1234 WOULD GIVE 12 X 34). THE SECOND PROMPTS XXØØØØXX FOLLOWED BY XX1111XX FOR THE TWO NUMBERS. ANSWERS ARE, AS USUAL, DISPLAYED AFTER A CONTROL KEY HAS BEEN PRESSED.
BOTH ARE BASED ON AN EQUIVALENT TO THE NORMAL METHOD OF LONG MULTIPLICATION.

```
E.G.          11Ø1Ø
              ØØ11Ø
         ØØØØØØØØØ     — (Ø X 2⁴) X 11010
          ØØØØØØØØ     — (Ø X 2³) X 11010
           11Ø1ØØØ     — (1 X 2²) X 11010
            11Ø1ØØ     — (1 X 2)  X 11010
            ØØØØØØ     — (Ø X 2⁰) X 11010
          1ØØ111ØØ
```

SINGLE BYTE MULTIPLY

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
Ø2ØØ	D8		CLD	
Ø2Ø2	84 2Ø		STY Ƶ 2Ø	— SET UP PROMPT FOR ZERO — MULTIPLIER
Ø2Ø4	A9 11		LDA #11	
Ø2Ø6	85 21		STA Ƶ 21	— PROMPT FOR FIRST — MULTIPLICAND

```
ADDR  HEX      LABEL     INSTRUCTION      COMMENTS
      CODE
020F  84 23              STY Z SUBL       - SUBTRACT 0001 AT START
0211  A4 20              LDY Z SQL        - USE Y & X AS DOUBLE SIZED
                                            ACCUMULATOR
0213  A6 21              LDX Z SQH
0215  38       NXTSUB    SEC              )
0216  98                 TYA              |
0217  E5 23              SBC Z SUBL       |
0219  A8                 TAY              > - SUBTRACT SUB FROM X & Y
021A  8A                 TXA              |
021B  E5 24              SBC Z SUBH       |
021D  AA                 TAX              )
021E  90 14              BCC RESULT       - IF NEGATIVE THEN STOP
0220  A9 00              LDA# 00          - NOT FINISHED YET. INCREMENT
                                            ROOT
0222  65 22              ADC Z ROOT
0224  85 22              STA Z ROOT
0226  A5 23              LDA Z SUBL       - INCREMENT SUB
0228  69 02              ADC #02
022A  85 23              STA Z SUBL
022C  A5 24              LDA Z SUBH
022E  69 00              ADC #00
0230  85 24              STA Z SUBH
0232  90 E1              BCC NXTSUB       - THERE CAN BE NO CARRY:
                                            BRANCH ALWAYS
0234  A5 22    RESULT    LDA Z ROOT
0236  20 60 FE           JSR DHEXTD       - DISPLAY ANSWER
0239  4C 04 FF           JMP RESTART
024B
```

THE DIVIDE ROUTINE WILL CALCULATE THE INTEGER RESULT AND REMAINDER OF A FOUR DIGIT NUMBER DIVIDED BY A TWO DIGIT NUMBER. BY ENTERING CLD (FOR HEX.) OR SED (FOR DECIMAL) EITHER BASE MAY BE USED, SINCE IT WORKS BY SUBTRACTING THE DIVISOR SUCCESSIVELY FROM THE DIVIDEND. THE ROUTINE PROMPTS WITH XX0000XX FOR THE DIVIDEND AND THEN XXXX11XX FOR THE DIVISOR. THE ANSWER WILL APPEAR IN THE FORM ABCD.EF WHERE ABCD IS THE INTEGER RESULT AND EF IS THE REMAINDER.

DIVIDER

```
ADDR  HEX        LABEL     INSTRUCTION              COMMENTS
      CODE
0200  D8 OR F8             CLD OR SED               - BINARY (DECIMAL) OPERATION
0201  84 20                STY Z 20  DIVIDED        - CLEAR DIVIDEND - PROMPT FOR
0203  84 21                STY Z 21                   NUMBER
0205  A9 11                LDA #11                  - PROMPT FOR SECOND NUMBER
0207  85 22                STA Z 22  DIVISOR
0209  A2 20                LDX #20
020B  20 88 FE             JSR QDATFET              - FETCH DIVIDEND
020E  A2 22                LDX #22
0210  20 88 FE             JSR QDATFET              - FETCH DIVISOR
0213  84 24                STY Z 24  RESULT         - CLEAR RESULT
0215  84 25                STY Z 25
```

THE OTHER FOUR ARE NEW, THEY ARE SHIFTS AND ROTATES. LET'S USE ASL AS AN EXAMPLE

```
0070  A9 55     LDA #55    LOAD A IMMEDIATE WITH 55
  72  0A        ASLA       ARITHMETIC SHIFT LEFT
  73  20 60 FE  JSR FE60   JUMP TO SUBROUTINE
  76  4C 04 FF  JMP FF04   JUMP
```

THE RESULT OF RUNNING THIS PROGRAM IS AA ON THE DISPLAY. EACH BIT IN THE ACCUMULATOR HAS BEEN SHIFTED ONE BIT LEFT.

```
            C            A
BEFORE     [1]       [01010101]
AFTER      [0]  <--  [10101010]  <--  0
```

ROLA, ROTATE LEFT ACCUMULATOR, (2A) WILL HAVE THE SAME EFFECT, EXCEPT THAT THE RIGHT INPUT 0 IS REPLACED BY C, IN THIS CASE 1, SO THE RESULT IS AB.

LSRA, LOGICAL SHIFT RIGHT ACCUMULATOR (4A)

```
            C
BEFORE     [1]    0      [01010101]      2A
AFTER      [1]     -->   [10101010]
```

RORA, ROTATE RIGHT ACCUMULATOR (6A) WILL REPLACE THE LEFT INPUT 0 WITH C TO GIVE AA

ALL THESE INSTRUCTIONS MAY BE USED DIRECTLY ON MEMORY LIKE INC Z.

4.5 MISCELLANEOUS REMAINING INSTRUCTIONS

THERE ARE A FEW INSTRUCTIONS LEFT, WHICH WILL HAVE TO BE DEALT WITH PIECE-MEAL:

BRK 00 : THE MICROPROCESSOR HAS TWO INTERRUPTS, AS EXPLAINED IN THE HARDWARE SECTION, AND THE INSTRUCTION SIMULATES AN IRQ, FIRST SETTING THE B FLAG IN THE STATUS REGISTER. THE RETURN AFTER A BREAK WILL BE AT THE NEXT BUT ONE BYTE

BIT 2C : A COMBINATION OF TWO INSTRUCTIONS

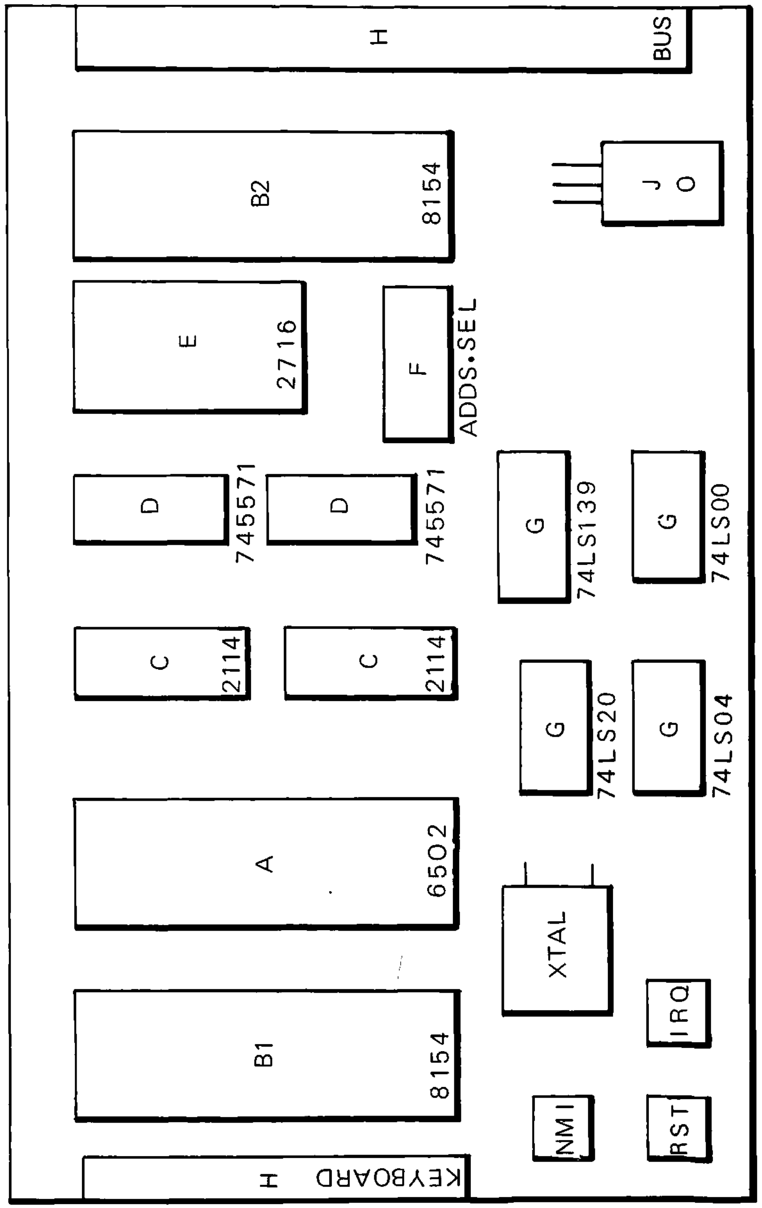
- I READ MEMORY BITS 6 & 7 INTO THE OVERFLOW & NEGATIVE FLAGS
- II LOGICAL AND ACCUMULATOR AND MEMORY, A ZERO RESULT SETTING THE Z FLAG. THE RESULT IS NOT LOADED INTO THE ACCUMULATOR. THE INSTRUCTION IS USUALLY USED TO TEST THE STATUS OF PERIPHERAL DEVICES, WITHOUT UPSETTING A,X OR Y.

RTI, RTS 40, 60 BOTH INSTRUCTIONS PULL THE PROGRAM COUNTER FROM THE STACK, RTI FIRST PULLS THE PROCESSOR STATUS FROM THE STACK.

CHAPTER 5: ACORN HARDWARE

5.1 CHIP LAYOUT AND BUS

BEFORE PLUNGING DEEPER INTO SOFTWARE WE'LL TAKE A REST AND LOOK AT THE HARDWARE. WE'LL START WITH THE CPU BOARD



SECTION 5.1

ROUTINES BY ENTERING THE SECTION OF PROGRAM FROM THE TITLE LABEL (E.G. DIVIDE) TO THE RESULT LABEL AND SUBSTITUTING THE LINE

6Ø RESULT RTS .

ALL ARE RELOCATABLE.

SYSTEM PROGRAMS

THESE PROGRAMS ARE ALL SHORT ROUTINES WHICH CAN PROVE USEFUL TIME SAVERS AT THE DEVELOPMENT AND INPUT STAGES OF PROGRAM WRITING.
IT MAY BE FOUND USEFUL TO KEEP COPIES OF THEM ON TAPE AND TO HAVE THEM IN THE ACORN AND BESIDE YOU WHILE DEVELOPING PROGRAMS. BRANCH CALCULATIONS IN PARTICULAR ARE A FERTILE SOURCE OF ERRORS AND TIME WASTING IN ANY HAND ASSEMBLED PROGRAM.
THE RELOCATOR WILL MOVE PROGRAMS AROUND MEMORY FOR YOU. A GODSEND TO ANYONE WHO FINDS THEMSELVES WITH THE NEED TO REENTER LARGE CHUNKS OF CODE MANUALLY.

MISCELLANEOUS

THIS IS A SELECTION OF PROGRAMS AND ROUTINES INCLUDED BECAUSE THEY ARE INTERESTING, ELEGANT OR IMPORTANT. THEY SHOW SOME OF OF THE THINGS THAT CAN BE DONE WITH THE SYSTEM, WHICH MAY BE MORE THAN YOU IMAGINE. WE HAVE, FOR INSTANCE, RUN A CHESS GAME IN THE 1K SYSTEM.
IN PARTICULAR THE METRONOME AND COUNTER PROGRAMS ARE INTENDED TO DEMONSTRATE SOME OF THE USES OF THE KEYBOARD. IN ORDER TO UNDERSTAND WHAT IS GOING ON WITH THESE PROGRAMS YOU WOULD BE WELL ADVISED TO STUDY THE MONITOR LISTING AND PART 1 OF THIS MANUAL.

MATHEMATICAL

THE SQUARE ROOT PROGRAM WILL CALCULATE EITHER DECIMAL OR HEXA-DECIMAL SQUARE ROOTS ACCORDING AS CLD (FOR HEX) OR SED (FOR DECIMAL) IS ENTERED AS THE FIRST LINE. IN EITHER CASE THE PROMPT WILL BE XXØØØØXX . THE SQUARE SHOULD BE ENTERED, A CONTROL KEY PRESSED AND THE ROOT WILL APPEAR ON THE DISPLAY.
THE PROGRAM IS BASED ON THE EQUALITY

((N+1)∗(N+1))–(N∗N)=(2∗N)+1

AND SUCCESIVELY SUBTRACTS 1,3,7,9 ETC. FROM THE SQUARE. WHEN THE RESULT OF A SUBTRACTION GOES NEGATIVE THE NUMBER OF SUBTRACTIONS DONE TO DATE IS THE ROOT.

HEX/DEC SQ ROOT. **RELOCATABLE**

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
Ø2ØØ	F8 OR D8		SED (OR CLD)	– SET DECIMAL (BINARY) OPERATING
Ø2Ø1	84 21		STY Ƶ SQH	– CLEAR SQUARE TO PROMPT
Ø2Ø3	84 2Ø		STY Ƶ SQL	
Ø2Ø5	A2 2Ø		LDX #SQL	
Ø2Ø7	2Ø 88 FE		JSR QDATFET	– FETCH THE NO. WHOSE ROOT IS TO BE FOUND
Ø2ØA	84 24		STY Ƶ SUBH	– CLEAR HIGH PART OF SUBTRACTING NO.
Ø2ØC	84 22		STY Ƶ ROOT	– CLEAR ROOT
Ø2ØE	C8		INY	

PART 2
APPLICATION PROGRAMS

MATHEMATICAL

1. SQUARE ROOT (HEX. OR DECIMAL)
2. DIVIDE (HEX. OR DECIMAL)
3. SINGLE BYTE MULTIPLY
4. DOUBLE BYTE MULTIPLY

SYSTEM

1. DECIMAL TO HEX.
2. HEX. TO DECIMAL
3. BRANCH OFFSET CALCULATOR
4. RELOCATOR
5. TAPE USE PROGRAMS
6. SCROLL

GAMES

1. NIM
2. DUCK SHOOT

MISCELLANEOUS

1. COUNTER
2. KEYBOARD COUNTER ROUTINE
3. METRONOME
4. EIGHT QUEENS PROBLEM

GENERAL

THESE APPLICATIONS PROGRAMS ARE INTENDED TO DEMONSTRATE SOME OF THE CAPABILITIES OF THE SYSTEM AND OF THE PROCESSOR. THEY HAVE BEEN DESIGNED FOR CLARITY AND SIMPLICITY AND IN MANY CASES ARE NOT OPTIMAL EITHER IN TERMS OF LENGTH OF PROGRAM OR OF EXECUTION TIME. THEY ARE INTENDED SIMPLY TO GIVE YOU A FEEL FOR THE SYSTEM AND SOMEWHERE TO START OFF FROM.

ALL PROGRAMS MARKED RELOCATABLE CAN BE ENTERED ANYWHERE IN AVAILABLE MEMORY, SUBJECT TO NOT IMPINGING IN VARIABLE STORAGE SPACE FOR EITHER THE PROGRAM OR MONITOR AND NOT USING SPACE NEEDED BY THE STACK. (FOR STACK USAGE SEE RELEVANT SECTIONS OF PART 1 OF THIS MANUAL.)

AS FAR AS HAS PROVED POSSIBLE THE CONVENTION OF A XX ØØØØ XX PROMPT FOR THE FIRST NUMBER TO BE ENTERED AND XX 1111 XX FOR THE SECOND HAS BEEN OBSERVED IN THESE PROGRAMS. AFTER ENTERING A NUMBER CHECK THAT IT IS CORRECT AND THEN PRESS A CONTROL KEY (ANY ONE WILL DO) TO PROGRESS THROUGH THE PROGRAM.

YOU SHOULD NOW BE READY TO TYPE IN THE PROGRAMS AND RUN THEM, BOTH TO ASSURE YOURSELF THAT THE SYSTEM IS OPERABLE AND TO FAMILIARISE YOURSELF WITH ITS OPERATION.

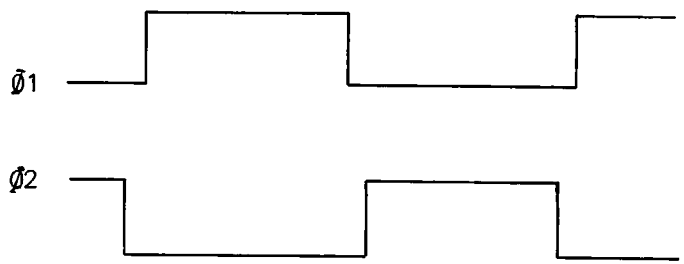
THROUGHOUT THESE NOTES X INDICATES AN UNDEFINED/UNIMPORTANT CHARACTER.

MOST OF THE PROGRAMS WERE WRITTEN BY MARK I'ANSON, THANK YOU MARK I.

MATHEMATICAL PROGRAMS

ALL THESE ROUTINES RESET THEMSELVES WHEN A CONTROL KEY IS PRESSED AFTER THE NUMBER HAS BEEN OBTAINED. THEY MAY ALL BE USED AS SUB

THE OBVIOUS IMPORTANT DEVICE HERE IS A, THE MICROPROCESSOR. THIS IS WHERE A,X,Y,P,S,PC LIVE. FROM HERE COME THE COMMANDS TO RUN EVERYTHING ELSE. THERE ARE TWO PRIMARY BUSSES, CONSISTING OF PARALLEL PATHS OF BINARY DATA, THE BIGGEST BUS IS THE ADDRESS BUS. THIS CONSISTS OF 16 LINES TO TRANSFER THE ADDRESS GENERATED BY THE PROCESSOR TO THE ADDRESS INPUTS OF ALL OTHER SYSTEM CHIPS. THIS BUS IS UNIDIRECTIONAL : ONLY THE PROCESSOR (IN A NORMAL SYSTEM) GENERATES ADDRESSES, AND IT HAS 2^{16} STATES (=65536,) THE SECOND BUS IS THE DATA BUS. THIS IS 8 BI-DIRECTIONAL LINES, ALLOWING A SINGLE WORD/BYTE TO BE TRANSFERRED EITHER FROM THE PROCESSOR TO MEMORY – A WRITE, OR FROM MEMORY TO PROCESSOR – A READ. THE REMAINING BUS IS THE CONTROL BUS, ITS MEMBERS HAVE NO PARTICULAR RELATIONSHIP WITH EACH OTHER, BUT THEY ARE ALL SUPERVISORY SIGNALS FOR THE SYSTEM. THE FIRST CONTROL SIGNAL IS THE R/W LINE. THIS SPECIFIES THE TYPE OF DATA TRANSFER THAT THE PROCESSOR WISHES TO MAKE: WHEN THE R/W LINE IS HIGH (LOGIC ONE; > 2.4 V DC) THE PROCESSOR IS READING WHEN THE R/W LINE IS LOW (LOGIC ZERO < Ø.8 V DC) THE PROCESSOR IS WRITING. THE NEXT CONTROL LINES ARE THE SYSTEM CLOCK, WHICH CONTROLS THE TIMING OF ALL DATA TRANSFERS. THE PROCESSOR, WITH HELP FROM 1/6 OF A TTL IC, GENERATES THE SYTEM CLOCK AS TWO NON-OVERLAPPING SQUARE WAVES, PHASE ONE (Ø1) & PHASE TWO (Ø2)



DURING Ø1 THE ADDRESS BUS AND THE R/W LINE CHANGE, AT THE END OF, OR DURING, Ø2 THE DATA IS TRANSFERRED. OTHER CONTROL SIGNALS ALSO CHANGE AT TIMES SPECIFIED WITH RESPECT TO THE SYSTEM CLOCK, E.G. THE SYNC SIGNAL : THIS GOES HIGH DURING Ø1 WHEN THE PROCESSOR IS FETCHING AN INSTRUCTION, AND RETURNS LOW WITH THE TRAILING EDGE OF Ø2.

5.2 RESET. INTERRUPT REQUEST AND NON-MASKABLE INTERRUPT

ANOTHER CONTROL LINE IS RESET. THIS IS GENERATED BY SUITABLE HARDWARE (IN THE ACORN THE CORNER SWITCH ON THE CPU BOARD, AND THE RESET SWITCH ON THE KEYBOARD,) AND CAUSES ALL PARTS OF THE SYSTEM TO BE RESET TO A SAFE, KNOWN STATE. IN THE PROCESSOR'S CASE RESET INITIALIZES THE PROGRAM COUNTER TO THE CONTENTS OF ADDRESSES FFFC AND FFFD WHICH, FOR ACORN, CONTAIN THE ADDRESS FEF3. EXECUTION OF THE ACORN MONITOR STARTS THERE. PERIPHERAL DEVICES SHOULD BE SET TO THEIR LEAST DANGEROUS STATE BY RESET, E.G. REMOVE INTERRUPT CAPABILITY, SET ALL PROGRAMMABLE INPUT/OUTPUT LINES TO INPUTS.

THE TWO PUSH BUTTONS ON THE CPU BOARD ON EITHER SIDE OF THE RESET BUTTON ARE INTERRUPT BUTTONS. THE IDEA OF AN INTERRUPT IS TO PULL THE PROCESSOR AWAY FROM IT'S CURRENT TASK, LET IT BRIEFLY DO SOMETHING IMPORTANT AND THEN RETURN TO IT'S TASK AS IF NOTHING HAD HAPPENED. THE 65Ø2 HAS TWO DISTINCT INTERRUPT CAPABILITIES

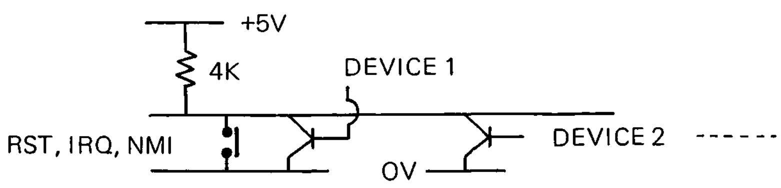
IRQ

WITH AN INTERRUPT REQUEST, IRQ, THE PROCESSOR HAS THE OPTION OF IGNORING IT. AN IRQ IS ONLY GRANTED IF THE FLAG I (INTERRUPT DISABLE) IN THE PROCESSOR STATUS REGISTER IS Ø. THE PROCESSOR THEN PUSHES PC & P & THEN SETS I TO 1. (THE STATE OF THE IRQ LINE IS CHECKED BETWEEN INSTRUCTIONS . . . IF IT REMAINS LOW, WE DON'T WANT ANOTHER INTERRUPT). THEN THE PROCESSOR LOADS PC FROM LOCATIONS FFFE & FFFF AND CONTINUES. NOTE THAT AN RTI RETURNS THE ORIGINAL P, WHICH HAD THE I FLAG Ø.

NMI

WITH A NON-MASKABLE INTERRUPT, NMI, THE PROCESSOR HAS NO OPTIONS; WHEN THE LINE HAS BEEN LOW FOR AT LEAST TWO CLOCK CYCLES, THE PROCESSOR WILL FINISH ITS CURRENT INSTRUCTION, SAVE ITS STATUS & PC, SET I HIGH AND FETCH A NEW PC FROM FFFA & FFFB. TO AVOID RECOGNISING ANOTHER INTERRUPT NMI IS EDGE-SENSITIVE: NO FURTHER INTERRUPTS ARE RECOGNISED UNTIL NMI HAS RETURNED HIGH. SINCE NMI SETS I HIGH, IRQ WILL NOT SUCCEED DURING THE NORMAL OPERATION OF AN NMI PROGRAM, BUT NMI WILL BE ABLE TO TAKE CONTROL DURING EXECUTION OF AN IRQ PROGRAM; IT HAS A HIGHER PRIORITY.

IRQ, NMI, & RESET ARE OPEN-COLLECTOR LINES ON THE CPU BOARD: MANY INTERRUPTING/RESETTING DEVICES MAY BE CONNECTED.

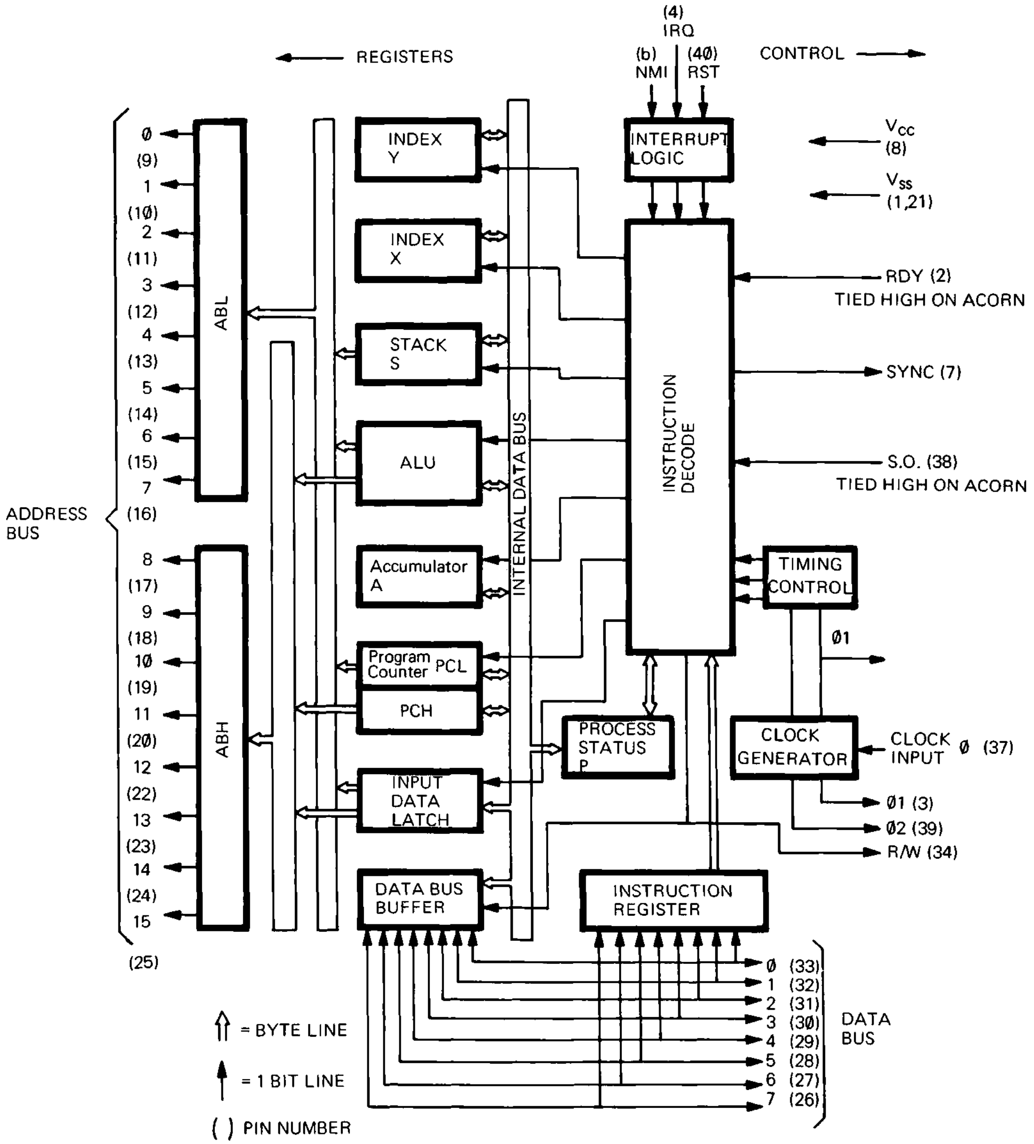



```
FF B3  85 ØA          BREAK   STA Z RØ      - WHEN THE IRQ/BREAK VECTOR
                                              POINTS HERE THEN DISPLAY
                                              DISPLAY EVERYTHING - SAVE A
   B5  86 ØB                  STX Z R1      - SAVE X
   B7  84 ØC                  STY Z R2      - SAVE Y
   B9  68                     PLA           - GET P OFF STACK
   BA  48                     PHA           - PUT IT BACK FOR FUTURE USE
   BB  85 ØD                  STA Z R3      - STORE Q IN REGISTER 3
   BD  A2 ØD                  LDX #R3       - SET X TO POINT AT REGISTERS
                                              3→Ø FOR QUAD
   BF  A9 FF                  LDA # FF      - KILL POSSIBILITY OF DISPLAY
                                              BEING ON SINGLE SCAN
   C1  85 ØE                  STA Z REPEAT
   C3  2Ø ØØ FE               JSR QUAD      - USE QUAD TO WRITE OUT A X Y P
   C6  BA                     TSX           - GET STACK POINTER
   C7  86 13                  STX Z R7
   C9  C8                     INY           - Y ZERO SINE QUAD ENDED WITH
                                              DISPLAY SO THIS FORMS Ø1
   CA  84 12                  STY Z R6
   CC  D8                     CLD           - CLEAR DECIMAL MODE FOR BINARY
                                              SUBTRACT - DOESN'T AFFECT
                                              USER SINCE P IS STACKED
   CD  BD Ø2 Ø1               LDA, X Ø1Ø2   - GET PCL OFF STACK
   DØ  38                     SEC
   D1  E5 1F                  SBC Z RECAL   - CORRECT IT BY AMOUNT IN RECAL
   D3  9D Ø2 Ø1               STA, X Ø1Ø2   - PUT IT BACK ON STACK
   D6  85 11                  STA Z R5      - AND STORE IT FOR QUAD
   D8  BD Ø3 Ø1               LDA, X Ø1Ø3   - PCH OFF STACK
   DB  E9 ØØ                  SBC #ØØ       - REST OF TWO BYTE SUBTRACTION
   DD  9D Ø3 Ø1               STA, X Ø1Ø3   - PUT IT BACK ON STACK
   EØ  85 1Ø                  STA Z R4      - AND STORE IT FOR QUAD
   E2  A2 13                  LDX #R7       - POINT X AT THESE REGISTERS -
                                              QUAD WILL DESTROY THEM
   E4  2Ø ØØ FE               JSR QUAD      - QUAD WRITES OUT PC SP
FF E7  4C Ø7 FF               JMP RE-ENTER  - AND THE WHOLE SHEBARG STARTS
                                              OVER AGAIN
FF EA  3F Ø6 5B 4F    FONT    'Ø' '1' '2' '3'  - 7 SEGMENT FORMS OF THE HEX
                                              DIGITS
   EE  66 6D 7D Ø7            '4' '5' '6' '7'
   F2  7F 6F 77 7C            '8' '9' 'A' 'b'
   F6  58 5E 79 71            'c' 'd' 'E' 'F'
FF FA  AD FF          NMIVEC  NMI           - POINT TO THE ADDED INDIRECTION
FF FC  F3 FE          RSTVEC  RESET         - POINT TO THE RESET ENTRY POINT
FF FE  BØ FF          IRQVEC  IRQ           - POINT TO THE ADDED INDIRECTION
```

```
FF45  2Ø  5E  FE  "MODIFY"  JSR MHEXTD     – DISPLAY THE MEMORY
  48  2Ø  ØC  FE            JSR DISPLAY    – AND GET KEY
  4B  BØ  BC                BCS SEARCH     – IF NOT HEX DO OVER
  4D  A1  ØØ                LDA (ØØ, X)    – HEX SO GET OLD INFO
  4F  ØA                    ASL A
  5Ø  ØA                    ASL A
  51  ØA                    ASL A
  52  ØA                    ASL A          – MOVED ALONG
  53  Ø5  19                ORA Ƶ KEY      – AND PUT IN NEW INFO
  55  81  ØØ                STA (ØØ, X)    – AND PUT IT BACK
  57  4C  45  FF            JMP "MODIFY"   – THEN KEEP DOING IT
FF5A  DØ  Ø3      N1        BNE N2         – MUST BE 4 OR 6 AS 2 IS
FF5C  6C  Ø2  ØØ  "GO"      JMP (GAP)      – THE VERY SIMPLE GO
FF5F  EØ  Ø4      N2        CPX #Ø4        – IS IT 4 OR 6?
  61  FØ  36                BEQ POINT      – WELL IT'S NOT 4
  63  A2  Ø8      "STORE"   LDX #Ø8        – SO IT MUST BE 6 – X NOW POINTS
                                             TO TAP
  65  86  1Ø                STX Ƶ D        – GIVE PROMPT
  67  2Ø  88  FE            JSR QDATFET    – AND GET 2ND STORE INFO
FF6A  A2  Ø4                LDX #Ø4        – LOOP COUNT
FF6C  B5  Ø5                LDA Ƶ,X Ø5     – SEND ADDRESSES TO TAPE
  6E  2Ø  B1  FE            JSR PUTBYTE
  71  CA                    DEX
FF72  DØ  F8                BNE ADDRESS    – X NEATLY ZEROED ON EXIT
FF74  A1  Ø6      DATAS     LDA (Ø6, X)    – DATA SEND – GET INFO FROM
                                             MEMORY
  76  2Ø  31  FE            JSR PUTBYTE    – AND SEND IT TO TAPE
  79  2Ø  AØ  FE            JSR COM16      – SEE IF PRINTED
  7C  DØ  F6                BNE DATAS      – NO
  7E  FØ  2A                BEQ WAYOUT     – YES
FF8Ø  A2  Ø4      "LOAD"    LDX #Ø4
FF82  2Ø  DD  FE  ADDRSL    JSR GETBYTE    – RESCUE ADDRESSES FROM TAPE
  85  95  Ø5                STA Ƶ,X Ø5     – PUT THEM IN FAP & TAP, THOUGH
                                             IT COULD BE ELSEWHERE
  87  CA                    DEX
  88  DØ  F8                BNE ADDRSL     – X NEATLY SERVED AGAIN
FF8A  2Ø  DD  FE  DATAL     JSR GETBYTE    – GET DATA FROM TAPE
  8D  81  Ø6                STA (Ø6, X)    – AND STORE IT IN MEMORY
  8F  8D  21  ØE            STA 1PIB       – AND ON THE DISPLAY SO IT CAN BE
                                             SEEN
  92  2Ø  AØ  FE            JSR COM16      – SEE IF FINISHED
  95  DØ  F3                BNE DATAL      – NO
  97  FØ  11                BEQ WAYOUT     – YES
FF99  A1  ØØ      "POINT"   LDA (ØØ, X)    – SET/CLEAR BREAK POINT – GET
                                             DATA FROM ADDRESSED MEMORY
  9B  FØ  Ø6                BEQ SET        – IF ZERO BREAK POINT HAS
                                             ALREADY BEEN SET = MUST CLEAR
                                             IT
  9D  85  18                STA Ƶ P        – NOT ZERO SO SAVE THE
                                             INFORMATION
  9F  A9  ØØ                LDA #ØØ        – AND PUT IN A BREAK POINT
  A1  FØ  Ø2                BEQ OUT
FFA3  A5  18      SET       LDA Ƶ P        – WAS SET SO GET OLD
                                             INFORMATION BACK
FFA5  81  ØØ      OUT       STA (ØØ, X)    – INSERT BREAK POINT OR OLD
                                             INFORMATION
  A7  2Ø  5E  FE            JSR MHEXTD     – NOW READ IT OUT AGAIN TO
                                             REVEAL ROM
FFAA  4C  Ø4  FF  WAYOUT    JMP RESTART    – GO BACK & DO IT ALL OVER AGAIN
FFAD  6C  1C  ØØ  NMI       JMP (USERNMI)  – INDIRECTION ON NMI
FFBØ  6C  1E  ØØ  IRQ       JMP (USERIRQ)  – INDIRECTION ON IRQ
```

TO DECIDE WHICH DEVICE CAUSED AN INTERRUPT THE PROCESSOR CHECKS A STATUS REGISTER OF EACH DEVICE, USING THE BIT INSTRUCTION TO TEST BIT 7 OF THE DEVICE. AFTER EXECUTING THE PROGRAM REQUIRED FOR A PARTICULAR DEVICE THE PROCESSOR RESETS THE DEVICE'S INTERRUPT BEFORE EXECUTING ITS RTI. IF THE INTERRUPT LINE IS STILL LOW (IRQ) OR MAKES ANOTHER NMI THE WHOLE THING IS REPEATED. THIS PRIORITIES THE INTERRUPTS IN SOFTWARE.

5.3 65Ø2 INTERNAL ARCHITECTURE



5.4 PROMS, EPROM, RAM, RAM I/O

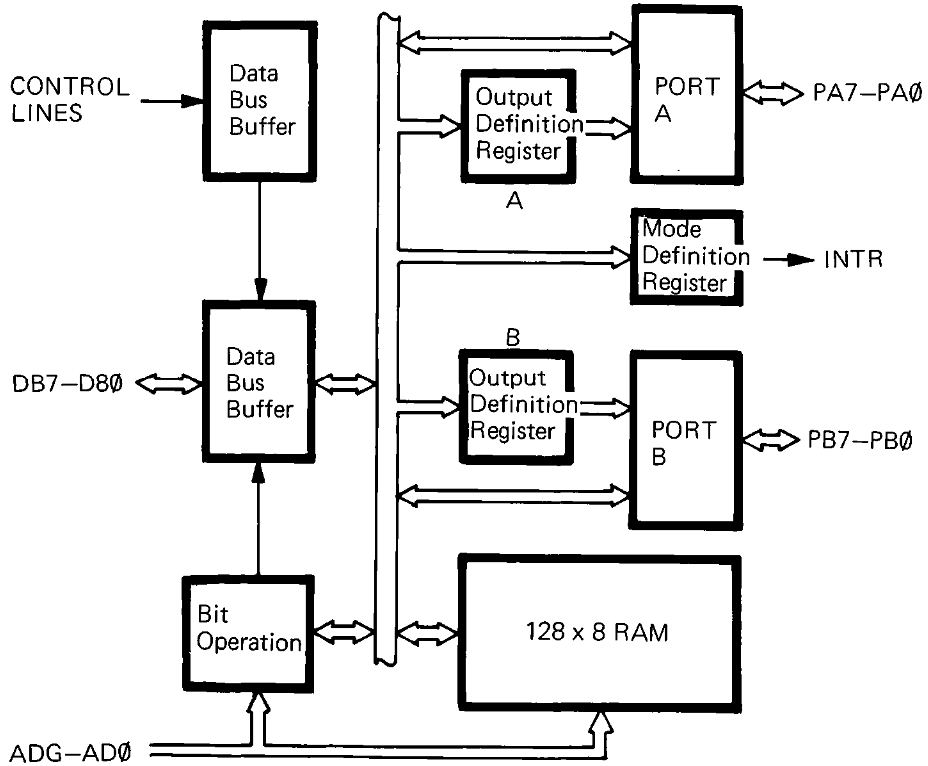
THE NEXT THINGS CONNECTED TO THE CPU ARE DEVICES D. THESE ARE PROMS: PROGRAMMABLE READ ONLY MEMORYS. EACH CONTAINS 512 X 4 BITS OF INFORMATION WHICH HAS BEEN FIXED AS HALF OF THE ACORN MONITOR. SHORT OF CATASTROPHIC DESTRUCTION THERE IS NO WAY TO MAKE A 'HIGH' PART OF THE MEMORY 'LOW', BUT 'LOW' PARTS CAN BE PROGRAMMED 'HIGH' BY PASSING EXCESS CURRENT THROUGH A FUSE AND DESTROYING IT. IN NORMAL ACORN OPERATION THESE TWO DEVICES WILL BE ENABLED BY ANY ADDRESS IN THE RANGE F800 TO FFFF: THEY THUS OCCUR IN THE MEMORY FOUR SEPARATE TIMES, MORE ON THIS ANON.
AKIN TO D, IS DEVICE E. THIS IS NOT PART OF THE KIT, BUT IS INTENDED TO BE A 2048 X 8 EPROM: ERASEABLE PROGRAMMABLE READ ONLY MEMORY. LIKE THE PROM, THE EPROM CAN BE PROGRAMMED ALTHOUGH FUSES ARE NOT BLOWN BUT CHARGE IS STORED ON THE GATE OF A FIELD EFFECT TRANSISTOR (F.E.T.). THIS CHARGE CAN ONLY LEAK AWAY SLOWLY – ABOUT TEN YEARS, UNLESS THE GATE IS EXPOSED TO ULTRA-VIOLET LIGHT WHICH HAS ENOUGH ENERGY TO SET THE DEVICE BACK TO IT'S STANDBY STATE. (IF YOU MAKE ONE PROGRAM MISTAKE THE WHOLE DEVICE MUST BE ERASED TO ALLOW YOU TO CORRECT THE MISTAKE. STILL, IT'S BETTER THAN NOT BEING ABLE TO CORRECT A MISTAKE AS WITH THE PROM). AN ENABLE SIGNAL IS PROVIDED BETWEEN F000 & F7FF FOR THIS DEVICE, OR ELSE IT MAY BE PROGRAMMED WITH A LARGER MONITOR AND ENABLED BY THE F800 – FFFF SIGNAL. SMALLER (1024 X 8 or 512 X 8) EPROMS MAY ALSO BE FITTED IN SOCKET E, BUT THESE OLDER DEVICES USUALLY REQUIRE ADDITIONAL POWER SUPPLIES, AND TWO MODIFICATIONS TO THE CIRCUIT BOARD ARE REQUIRED TO ALLOW THIS.
THE LAST TYPE OF MEMORY ON THE CPU BOARD IS TYPE C. THIS IS A STATIC READ/WRITE MEMORY: INFORMATION CAN BE CREATED AND DESTROYED BY THE MICROPROCESSOR ITSELF, BUT ALL IS LOST WHEN THE POWER IS REMOVED. TOGETHER WITH THE DYNAMIC VERSION, THIS TYPE OF DEVICE HAS RECEIVED THE NAME RANDOM ACCESS MEMORY R.A.M., ALTHOUGH THEY ARE NO MORE RANDOM THAN P.R.O.M.S. OR E.P.R.O.M.S. DEVICES C ARE 1024 X 4 RAMS, TWO ARE REQUIRED LIKE THE TWO PROMS TO BUILD UP A WHOLE BYTE, AND THEY ARE ENABLED BY ADDRESSES IN THE RANGE 0000 TO 03FF. THEY THUS CONTAIN ZERO PAGE & PAGE 1, THE STACK PAGE, AS WELL AS TWO FURTHER PAGES.
THE ENABLE SIGNALS FOR ALL I.C.S. ON THE CPU BOARD ARE PROVIDED BY THE LOGIC I.C.'S G. THESE I.C.S. DECODE CERTAIN RANGES OF ADDRESSES FROM THE ADDRESS BUS BY RECOGNISING A PATTERN ON HIGH ADDRESS LINES, E.G. FOR THE SIGNAL TO THE TWO RAM'S THE TOP 6 (A15–A10) ADDRESS LINES MUST BE LOW (LOGIC ZERO).THE SIGNALS ARE ALL BROUGHT TO THE SOCKET F, WHERE LINKS CAN BE MADE (OR A D.I.L. HEADER USED) TO TAKE THE ENABLE SIGNALS AWAY TO THE CHOSEN DEVICES THUS MANY DIFFERENT SYSTEM CONFIGURATIONS CAN BE USED, FROM JUST THE TWO P.R.O.M.S AND DEVICE B1, THROUGH TO BOTH C'S, B2 & E OR ANY COMBINATION.

DEVICES B HAVE TWO FUNCTIONS. IN THE FIRST PLACE EACH CONTAINS A 128 X 8 RAM, BRINGING THE CPU BOARD UP TO 1280 BYTES OF R.A.M. SECONDLY EACH HAS THE FACILITIES FOR MAKING TWO WORDS OF MEMORY

```
   EA   0E 20 0E             ASL 1PIA        – GET SAMPLE AUTO CARRY
   ED   6A                   ROR A           – AND AUTO A
   EE   88                   DEY
   EF   D0 F6                BNE INPUT       – KEEP GOING
FE F1   F0 DA                BEQ WAIT        – USE WAIT TO GET OUT ONTO THE
                                               THE SHOP BIT HIGH
FE F3   A2 FF      RESET     LDX #FF         – MAIN PROGRAM
   F5   0A                   TXS             – INITIALIZE STACK
   F6   8E 23 0E             STX 1BDDR       – AND B DATA DIRECTION REGISTER
   F9   86 0E                STX Ƶ REPEAT    – MULTI-SCAN DISPLAY MODE
FE FB   A0 80      INIT      LDY #80         – THE FAMILIAR DOT ON THE
                                               DISPLAY
   FD   A2 09                LDX #09         – ALL EIGHT DISPLAYS AND
                                               INITIALIZE EXEC
   FF   94 0E      ROUND     STY Ƶ, X REPEAT – Y USED FOR AMUSEMENT
FF01    CA                   DEX
FF 02   D0 FB                BNE ROUND       – X ZERO ON EXIT, SO UP & DOWN
                                               IMMEDIATELY VALID
FF 04   20 0C FE   RESTART   JSR DISPLAY     – MARK RETURN TO MONITOR POINT
                                               DISPLAY DISPLAY & GET KEY
FF 07   90 F2      RE-ENTER  BCC INIT        – HEX KEY GETS THE DOTS BACK
FF 09   29 07      SEARCH    AND #07         – REMOVE ANY STRAY BITS
                                               (EFFECTIVELY SUBTRACT 10)
   0B   C9 04                CMP #04
   0D   90 25                BCC FETADD      – KEYS OF VALUE LESS THAN 4
                                               NEED AN ADDRESS
   0F   F0 6F                BEQ LOAD        – KEY 4 IS THE LOAD KEY
   11   C9 06                CMP #06
   13   F0 09                BEQ "UP"        – KEY 6 IS UP
   15   B0 0F                BCS "DOWN"      – & KEY 7 IS DOWN
FF 17   A5 0A      "RETURN"  LDA Ƶ R0        – MUST BE KEY 5 – GET A BACK
   19   A6 0B                LDX Ƶ R1        – GET X BACK
   1B   A4 0C                LDY Ƶ R2        – GET Y BACK
   1D   40                   RTI             – GET P & PC BACK & CONTINUE
                                               FROM WHERE YOU WERE
FF 1E   F6 00      "UP"      INC Ƶ,X 00      – 16 BIT INDEXED INCREMENT
   20   D0 0C                BNE ENTERM
   22   F6 01                INC Ƶ,X 01
   24   B0 08                BCS ENTERM      – A BRANCH ALWAYS : THE CARRY
                                               WAS SET BY THE FF11 COMPARE
FF 26   B5 00      "DOWN"    LDA Ƶ,X 00      – 16 BIT INDEXED DECREMENT
   28   D0 02                BNE NODEC
   2A   D6 01                DEC Ƶ,X 01
FF 2C   D6 00      NODEC     DEC Ƶ,X 00
FF 2E   20 64 FF   ENTERM    JSR QHEXTD1     – NOW DISPLAY THE VALUE
   31   4C 45 FF             JMP "MODIFY"    – AND GET INTO THE MODIFY
                                               SECTION
FF 34   84 16      FETADD    STY Ƶ D+6       – CLEAR DISPLAYS 6
   36   84 17                STY Ƶ D+7         & 7 – Y WAS ZERO ON EXIT FROM
                                               DISPLAY
   38   0A                   ASL A           – DOUBLE A
FF 39   AA                   TAX             – THE ZERO PAGE ADDRESSES MAP,
                                               GAP, PAP & FAP
   3A   49 F7                EOR #F7         – FIX UP DIGIT 0 COMMAND SYMBOL
   3C   85 10                STA Ƶ D
   3E   20 88 FE             JSR QDATFET     – FETCH THE ADDRESS, AUTO MAP,
                                               GAP, PAP OR FAP
   41   E0 02                CPX #02         – CHECK X TO FIND OUT WHICH
                                               COMMAND WE'RE DOING
   43   B0 15                BCS NI          – MUST BE 2, 4 OR 6 – AS 0 IS
```

```
  87  60                    RTS
FE88  20  64  FE  QDATFE7   JSR QHEXTD1     – QUAD DATA FETCH – DISPLAY OLD
                                              DATA
  8B  20  ØC  FE            JSR DISPLAY     – GET KEY
  8E  BØ  2Ø                BCS RETURN      – NON HEX RETURN
  9Ø  AØ  Ø4                LDY #Ø4         – LOOP COUNTER
  92  ØA                    ASL A
  93  ØA                    ASL A
  94  ØA                    ASL A
  95  ØA                    ASL A           – DIGIT IN A IN CORRECT PLACE
FE96  ØA          SHIFTIN   ASL A           – MULTI SHIFT TO GET DIGIT INTO
                                              MEMORY
FE97  36  ØØ                ROL Z,X ØØ      – INDEXED
  99  36  Ø1                ROL Z,X Ø1
  9B  88                    DEY
  9C  DØ  F8                BNE SHIFTIN     – KEEP SHIFTING IN
  9E  FØ  E8                BEQ QDATFET     – GO AND DO IT ALL AGAIN
FEAØ  F6  Ø6      COM 16    INC Z,X Ø6      – INCREMENT & COMPARE 16 BIT
                                              NOS – INCREMENT LOWER
  A2  DØ  Ø2                BNE NOINC       – NO HIGH INCREMENT
  A4  F6  Ø7                INC Z,X Ø7
FEA6  B5  Ø6      UDINC     LDA Z,X Ø6      – LOW BYTE EQUALITY TEST
  A8  D5  Ø7                CMP Z,X Ø8
  AA  DØ  Ø4                BNE RETURN      – NO NEED TO DO HIGH BYTE
  AC  B5  Ø7                LDA Z,X Ø7      – HIGH BYTE EQUALITY TEST
  AE  D5  Ø9                CMP Z,X Ø9
FEBØ  6Ø          RETURN    RTS
FEB1  AØ  4Ø      PUTBYTE   LDY #4Ø         – PUT BYTE TO TAPE – CONFIGURE
                                              I/O PORT
  B3  8C  22  ØE            STY 1ADDR
  B6  AØ  Ø7                LDY #Ø7         – LOOP COUNTER
  B8  8C  2Ø  ØE            STY 1PIA        – AND SEND THE START BIT
  BB  6A                    ROR A
  BC  6A                    ROR A           – BACK A UP A COUPLE OF BITS
FEBD  2Ø  CD  FE  AGAIN     JSR WAIT        – WAIT TO SEND OUT RESET BIT
  CØ  6A                    ROR A           – SENDING ORDER IS BIT Ø→BIT 7
  G1  8D  2Ø  ØE            STA 1PIA        – SEND BIT
  C4  88                    DEY
  C5  1Ø  F6                BNE AGAIN       – KEEP GOING
  C7  2Ø  CD  FE            JSR WAIT        – WAIT FOR THAT BIT TO END
  CA  8C  2Ø  ØE            STY 1PIA        – SEND STOP BIT : Y IS FF
FECD  2Ø  DØ  FE  WAIT      JSR ½ WAIT      – 3ØØ BAND WAITING TIME – IN TWO
                                              PARTS
FEDØ  84  1A      ½ WAIT    STY Z TY        – ½ THE WAITING TIME – SAVE Y
  D2  AØ  48                LDY #48         – 72 X 5µS DELAY
  D4  88          WAIT 1    DEY             – PART ONE OF THE WAIT
  D5  DØ  FD                BNE WAIT 1
  DY  88          WAIT 2    DEY             – Y WAS ZERO ON ENTRY – 256 x 5µS
                                              DELAY
  D8  DØ  FD                BNE WAIT 2
  DA  A4  1A                LDY Z TY        – RETRIEVE Y
  DC  6Ø                    RTS
FEDD  AØ  Ø8      GETBYTE   LDY #Ø8         – GET BYTE FROM TAPE – LOAD
                                              COUNTER
FEDF  2C  2Ø  ØE  START     BIT 1PIA        – WAIT FOR 1 →Ø TRANSISITON –
                                              A START BIT
  E2  3Ø  FB                BMI START
  E4  2Ø  DØ  FE            JSR ½ WAIT      – WAIT HALF THE TIME, SO
                                              SAMPLING IN THE CENTRE
FEE7  2Ø  CD  FE  INPUT     JSR WAIT        – FULL WAIT TIME BETWEEN
                                              SAMPLES
```

(16 BITS) APPEAR IN A USABLE FORM FOR THE OUTSIDE WORLD. THE ACORN MONITOR USES DEVICE B1 TO CONTROL THE DISPLAY, CASSETTE INTERFACE AND KEYBOARD.
EACH ONE OF THE 16 LINES MAY BE PROGRAMMED TO BE AN INPUT OR AN



8154 RAM I/O

OUTPUT DEPENDING ON THE STATE OF INTERNAL CONTROL REGISTERS. ONLY A GENERAL DESCRIPTION OF THE DEVICE IS GIVEN HERE, IN ADDITION TO THE FOLLOWING FUNCTIONS PORT A MAY BE SET TO OPERATE IN A VARIETY OF DIFFERENT HANDSHAKING TRANSFER MODES BY USE OF THE MODE DEFINITION REGISTER. IT SHOULD BE NOTED THAT THESE MODES REQUIRE CONNECTION OF INTERRUPT AND THAT THE INS8154 INTERRUPT LINE IS THE INVERSE OF THAT REQUIRED BY THE PROCESSOR.

THE 16 LINES ARE, AS YOU MIGHT EXPECT, DIVIDED INTO TWO SEPERATE BYTE SECTIONS A & B. A & B BOTH HAVE AN "OUTPUT DEFINITION REGISTER" ASSOCIATED WITH THEM. EACH BIT IN THE O.D.R. DEFINES THE ASSOCIATED BIT IN THE 'PORT' AS EITHER AN INPUT (Ø) OR AN OUPUT (1). THUS, IN THE MONITOR WE WRITE FF TO THE SEGMENT O.D.R. TO USE ALL IT'S LINES AS OUTPUTS, AND 'DISPLAY' WRITES Ø7 TO THE DIGIT DRIVE O.D.R. TO HAVE 3 OUTPUTS AND 5 INPUTS.
NOT ONLY MAY WE READ/WRITE TO THE OUTPUT PORT USING THE PARALLEL READ & WRITE OPERATIONS, BUT WE MAY ALSO READ/WRITE SINGLE BITS:

OPERATION			ADDRESS LOW	R/W
SET	BIT Ø	PORT A	1Ø	W
SET	BIT 7	PORT A	17	W
CLEAR	BIT Ø	PORT A	ØØ	W
CLEAR	BIT 7	PORT A	Ø7	W
READ	BIT Ø	PORT A	ØØ or 1Ø	R
READ	BIT 7	PORT A	Ø7 or 17	R
SET	BIT 1	PORT B	19	W
SET	BIT 6	PORT B	1E	W
CLEAR	BIT 2	PORT B	ØA	W
CLEAR	BIT 5	PORT B	ØD	W
READ	BIT 4	PORT B	ØC or 1C	R
	PORT A		2Ø	R or W
	PORT B		21	R or W
	O.D.R.A.		22	W
	O.D.R.B.		23	W

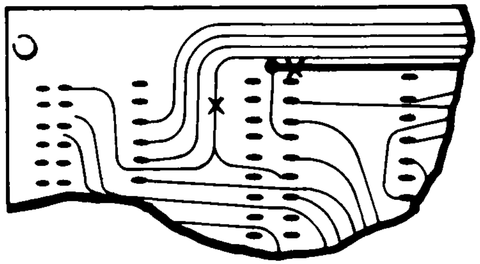
IF YOU READ A SINGLE BIT IT WILL END UP IN BIT 7 OF A BYTE, THUS THE BIT INSTRUCTION WILL ASSIGN IT TO THE TESTABLE N FLAG.
THE INS8154 ALSO CONTAINS A USEFUL 128 BYTES OF RAM. THIS IS CONTINUOUS FROM (ADDRESS LOW) 8Ø TO FF.
DEVICE B1 IS ENABLED FOR ADDRESS HIGH OF ØE, DEVICE B2 IS AT Ø9.

```
FE3E  E4 19        BUTTON    CPX Ƶ COL     – ARE WE ON THE SAME KEY'S
                                             COLUMN?
  4Ø  DØ FØ                  BNE DELAY     – NO
  42  C9 38                  CMP #38       – HAS A KEY ACTUALLY BEEN
                                             PRESSED?
  44  9Ø Ø4                  BCC PRESSED   – YES
  46  A9 8Ø                  LDA #8Ø       – NO, THEN CLEAR THE EXECUTION
                                             STATUS – THE KEY HAS BEEN
                                             PRESSED & RELEASED
  48  DØ E6                  BNE KEYCLEAR  – ALWAYS BRANCH
FE4A  C5 ØF        PRESSED   CMP Ƶ EXEC    – A KEY HAS BEEN PRESSED
  4C  FØ E4                  BEQ DELAY     – BUT IT HAS ALREADY BEEN
                                             EXECUTED
  4E  85 ØF                  STA Ƶ EXEC    – SET IT AS BEING EXECUTED
  5Ø  49 38                  EOR #38       – JIGGERY POKERY TO ENCODE THE
                                             ROW INPUTS TO BINARY
FE52  29 1F        OUTPUT    AND #IF       – ALSO ENSURE THE KEY IN REPEAT
                                             WAS OF REASONABLE SIZE
  54  C9 1Ø                  CMP #1Ø       – A HEX KEY OR NOT? CARRY CLEAR
                                             IF HEX
  56  85 ØD                  STA Ƶ KEY     – PUT THE KEY IN A TEMP LOCATION
                                             FOR FURTHER USE (BY "MODIFY")
  58  A6 1A                  LDX Ƶ TX      – RETRIEVE X
  5A  82 21 ØE               STY 1PIB      – TURN THE SEGMENT DRIVES OFF
  5D  6Ø                     RTS           – AND RETURN
FE5E  AØ ØØ        MHEXTD    LDA (ØØ, X)   – MEMORY HEX TO DISPLAY = GET A
                                             BYTE FROM MEMORY
FE6Ø  AØ Ø6        RDHEXTD   LDY #Ø6       – RIGHT (OF DISPLAY) DOUBLE HEX
                                             TO DISPLAY : SET Y TO RIGHT OF
                                             DISPLAY
  62  DØ ØB                  BNE DHEXTD    – AND USE DHEXTD
FE64  AØ Ø3        QHEXTD1   LDY #Ø3       – QUAD HEX TO DISPLAY 1: SET Y
                                             TO USE POSNS 1,2,3 & 4
FE66  B5 ØØ        QHEXTD2   LDA ƵY ØØ       2: USE ANY Y: GET THE DATA
  68  2Ø 6F FE               JSR DHEXTD    – AND USE DHEXTD
  68  88                     DEY
FE6C  88                     DEY           – HAVING DECREMENTED THE
                                             POSITION
  6D  B5 Ø1                  LDA Ƶ, X Ø1   – GET THE HIGH BYTE OF THE DATA
                                             & USE DHEXTD
FE6F  C8           DHEXTD    INY           – DOUBLE HEX TO DISPLAY : FIRST
                                             HEX ON RIGHTEST POSITION
  7Ø  48                     PHA           – SAVE A
  71  2Ø 7A FE               JSR HEXTD     – USE HEX TO DISPLAY
  74  88                     DEY           – GET Y BACK TO CORRECT
                                             POSITION
  75  68                     PLA           – RETRIEVE A
  76  4A                     LSR A
  77  4A                     LSR A
  78  4A                     LSR A
  79  4A                     LSR A         – ORIENTATED FOR OTHER HEX
                                             DIGIT
FE7A  84 1A        HEXTD     STY Ƶ TY      – HEX TO DISPLAY = SAVE Y
  7C  29 ØF                  AND #ØF       – REMOVE SURPLUS BITS FROM A
  7E  A8                     TAY           – & PUT IT IN 7
  7F  B9 EA FF               LDA, Y FONT   – GET THE 7 SEGMENT FORM
  82  A4 1A                  LDY Ƶ TY      – RETRIEVE Y
  84  99 1Ø ØØ               STA, Y D      – AND POSITION THE 7 SEG FORM ON
                                             THE DISPLAY
```

ACORN MONITOR

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
FE ØØ	AØ Ø6	QUAD	LDY #Ø6	DISPLAY THE 4 BYTES AT X–3,X–2, X–1 & X IN THAT ORDER ON THE DISPLAY
FE Ø2	B5 ØØ	STILL	LDA ƵX ØØ	– GET THE BYTE POINTED TO BY X
Ø4	2Ø 6F FE		JSR DHEXTD	– USE DOUBLE HEX TO DISPLAY ROUTINE
Ø7	CA		DEX	– NEXT X
Ø8	88		DEY	– NEXT Y POSITION
Ø9	88		DEY	
ØA	1Ø F6		BPL STILL	– FALL AUTO DISPLAY WHEN FINISHED –Y POSITION & ALSO LOOP COUNTER
FE ØC	86 1A	DISPLAY	STX Ƶ TX	– SAVE X!!!!
FE ØE	A2 Ø7	RESCAN	LDX #Ø7	– SCAN 8 DIGITS, NO MATTER WHAT
1Ø	8E 22 ØE		STX 1 ADDR	– SET UP DATA DIRECTION REGISTER
FE 13	AØ ØØ	SCAN	LDY #ØØ	– CLEAR Y FOR LATER USE
15	B5 1Ø		LDA Ƶ,X D	– GET DISPLAY DATA FROM THE ZERO PAGE MEMORY
17	8D 21 ØE		STA 1PIB	– & PUT IT ONTO SEGMENTS
1A	8E 2Ø ØE		STX 1PIA	– SET DIGIT DRIVE ON AND THE KEY COLUMNS
1D	AD 2Ø ØE		LDA 1PIA	– GET KEY DIGIT BACK
2Ø	29 3F		AND #3F	– REMOVE SURPLUS TOP BITS
22	24 ØF		BIT Ƶ EXEC	– CHECK STATUS = 'I' MEANS NOT PROCESSING A KEY
24	1Ø 18		BPL BUTTON	– BUT Ø MEANS THAT WE ARE
26	7Ø ØA		BVS DELAY	– THUS CAN BE BLOWN TO AN ESCAPE FROM THE DISPLAY ROUTINE ALTOGETHER ON STATUS CØ AT THE MOMENT IT IGNORES KEYS IF GIVEN THIS STATUS
28	C9 38		CMP #38	– CHECK FOR ALL 1'S ROW INPUT FROM KEYBOARD = SET COPY IOF SO
2A	BØ Ø6		BCS DELAY	– IF ALL 1's THEN NO KEY HAS BEEN PRESSED
2C	86 19		STX Ƶ COL	– STORE THE PRESSED KEY'S COLUMN INFORMATION
2E	A9 4Ø		LDA #4Ø	– SET STATUS TO "WE ARE PROCESSING A KEY"
FE 3Ø	85 ØF	KEY CLEAR	STA Ƶ ECEC	
FE 32	88	DELAY	DEY	– Y WAS ZERO SO HERE IS A 256X5μS DELAY
33	DØ FD		BNE DELAY	– Y WILL BE ZERO ON EXIT
35	CA		DEX	
FE 36	1Ø DB		BPL SCAN	– IF X WAS STILL TVE, CONTINUE THIS SCAN
38	A5 ØE		LDA Ƶ REPEAT	– IF WE SHOULD CONTINUE SCANNING THEN TOP BIT IS SET
3A	3Ø D2		BMI RESCAN	– CONTINUE SCANNING
3C	1Ø 14		BPL OUTPUT	– IF TOP BIT IS ZERO, THEN USE THIS DATA AS THE KEY ITSELF

ALSO ON THE CPU BOARD IS A 5V REGULATOR. THIS PROVIDES THE REGULATED +5V POWER SUPPLY USED BY ALLTHE I.C.S. ON THE BOARD, AND THE KEYBOARD/INTERFACE BOARD WHEN CONNECTED. IF THE 27Ø4 OR 27Ø8 TYPE OF E.P.R.O.M. IS EMPLOYED IN SOCKET E, EXTRA +12 & –5 V POWER SUPPLY LINES ARE REQUIRED, AND TWO TRACKS ON THE P.C.B. NEED CUTTING.



THE TWO CUTS ARE ON THE REAR OF THE MPU BOARD IN THE TOP LEFT HAND CORNER. X's MARK THE SPOTS

(THERE IS NO PROVISION FOR ON-BOARD REGULATORS FOR THESE TWO EXTRA SUPPLIES).
OF COURSE, THE 2716 EPROM NEEDS NO EXTRA SUPPLY LINES, AND IS THE DEVICE THAT THE P.C.B. WAS DESIGNED FOR, IT PLUGS STRAIGHT INTO SOCKET E.
THE CONNECTOR H CARRIES THE ADDRESS BUS, THE DATA BUS, THE CONTROL BUS, POWER SUPPLY LINES AND THE 16 INPUT/OUTPUT LINES FROM B2. THIS WILL PLUG INTO A BACKPLANE WHICH TAKES THE BUSSES TO OTHER ACORN CARDS.

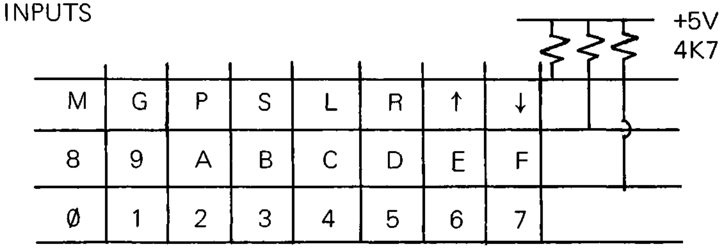
5.5 THE KEYBOARD AND TAPE INTERFACE

AT THE OTHER END OF THE BOARD, CONNECTOR I CARRIES ALL 16 I/O LINES FROM DEVICE B1, AS WELL AS OV, +5V, Ø2 & RESET LINES. WITH THE INTELLIGENT ACORN MONITOR AND THE KEYBOARD BOARD, THE I/O LINES ARE DEDICATED AS FOLLOWS

B1 PORT	BØ-7	OUTPUTS	SEGMENT DRIVES
	AØ-2	OUTPUTS	BINARY ENCODED DIGIT DRIVES
	A3-5	INPUTS	KEYBOARD ROW INPUTS
	A6	OUTPUT	FROM COMPUTER TO CASSETTE
	A7	INPUT	FROM CASSETTE TO COMPUTER

–A COMMENT FOR THOSE INTERESTED: ALTHOUGH THE KEYBOARD ONLY CONSISTS OF 24 KEYS AT PRESENT, IT IS POSSIBLE, WITH A PRIORITY ENCODER ON THE ROW INPUTS, TO USE UP TO 56 KEYS. THE DISPLAY SUBROUTINE WILL COPE CORRECTLY WITH THE UNKNOWN KEYS, EXCEPT THAT, AT THE POINT, OUTPUT, IT THROWS AWAY A SIGNIFICANT BIT OF INFORMATION. HOWEVER, THE ACTUAL KEY VALUE HAS BEEN STORED IN LOCATION ØØØF AND SO CAN BE RECOVERED. THE UNKNOWN KEYS WILL NOT AFFECT THE MONITOR ITSELF, SINCE AT THE POINT SEARCH MORE ITS OF INFORMATION IS THROWN AWAY, LEAVING THE MONITOR WITH A CHOICE OF EIGHT VALUES.

THE SUBROUTINE DISPLAY RUNS THE DISPLAY IN A MULTIPLEXED MANNER, AT THE SAME TIME STROBING AND DEBOUNCING THE MATRIXED KEYBOARD ON THE KEYBOARD BOARD. EACH OF THE EIGHT COLUMNS OF THE 8 X 3 KEYBOARD IS DRIVEN BY ONE OF THE EIGHT DIGIT DRIVER LINES, THE THREE ROW LINES ARE CONNECTED TO DEVICE B1, AND THEY ARE PULLED TO LOGIC ONE BY THE 4K7 RESISTORS. IN CONJUNCTION WITH ITS COLUMN BEING DRIVEN LOW, A CLOSED KEY PRODUCES A LOW ON ONE OF THE ROW INPUTS



ALL THE INTERFACE BETWEEN THE MICROPROCESSOR AND THE KEYBOARD AND DISPLAY IS THUS ACCOMPLISHED BY ONE OCTAL DECODER/DRIVER AND THREE RESISTORS. THE REST OF THE CIRCUITRY ON THE INTERFACE BOARD ALLOWS PROGRAMS TO BE RECORDED ON CASSETTE AT THIRTY BYTES PER SECOND, THE INTERFACE IS SLIGHTLY MORE COMPLICATED THAN THE SINGLE I.C. AND THREE RESISTORS USED ABOVE, IT HAS TWO TASKS.

I CONVERT THE SERIAL STREAM OF INFORMATION PRODUCED BY PUTBYTE INTO TONES SUITABLE FOR AN UNMODIFIED CASSETTE RECORDER TO RECORD.THE FREQUENCIES USED ARE 24Ø3.8 HZ FOR A LOGIC ONE AND 12Ø1.9 HZ FOR A LOGIC ZERO. THE FREQUENCIES ARE PRODUCED BY DIVIDING Ø2, WHICH IS CRYSTAL CONTROLLED AT 1 MHZ, BY 416 OR 832.

II CONVERT THE PLAYED BACK FREQUENCIES INTO A STREAM OF BINARY INFORMATION. THE PLAYBACK IS 'AMPLIFIED' INTO A SQUARE WAVE, AND ITS PERIOD IS COMPARED WITH THE PERIOD OF A REFERENCE DIGITAL MONOSTABLE ON THE CIRCUIT BOARD

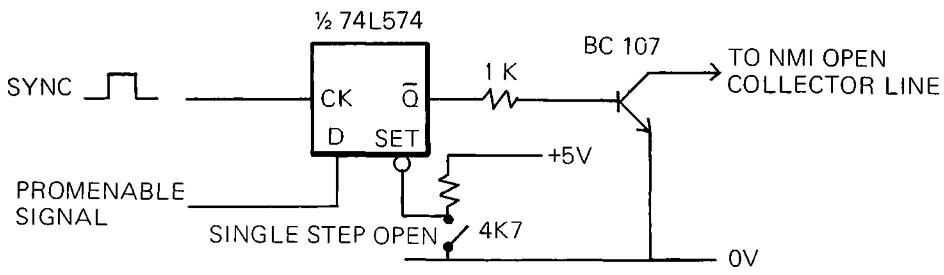
BECAUSE OF THE AMPLIFICATION STAGE, THE OUTPUT FROM A TAPE RECORDER'S 'LINE' OUTPUT, OR THE 'EAR' JACK SOCKET, SHOULD PERFORM SATISFACTORILY EVEN AT MODEST VOLUME LEVEL. HOWEVER THE COMPUTER OUTPUT IS AT QUITE HIGH LEVEL AND SHOULD BE ATTENUATED FOR THE TAPE RECORDER. TO PREVENT NOISE PICK-UP THIS SHOULD BE WITH THE COMPLETE PROCESSOR STATUS RECOVERED. THUS, IF WE FINISH THE PROGRAM

```
Ø2ØD    69 19       ADC #19
Ø2ØF    2Ø 6Ø FE    JSR RDHEXTD
Ø212    4C Ø4 FF    JMP RESTART
Ø215
```

AND PRESS R, THE DISPLAYED ANSWER WILL BE Ø3

6.3 THE SINGLE STEPPING FACILITY

A MORE INTERESTING USE OF THE ROUTINE BREAK AT FFB3 IS IF YOU GENERATE GENERATE A NMI EVERY OPCODE FETCHED NOT IN THE MONITOR, AS DISCUSSED DISCUSSED IN THE HARDWARE SECTION THE SYNC PULSE ISSUED DURING AN OPCODE FETCH IS LESS THAN 1 CYCLE LONG, WHILE NMI REQUIRES AT LEAST 2 CYCLES. A LATCH IS REQUIRED TO STRETCH THE SYNC SIGNAL



AND IT ALSO ONLY PROVIDES AN NMI WHEN NOT IN THE MONITOR. BEFORE EXECUTING A PROGRAM SET THE NMI VECTOR (LOCATIONS ØØ1C & ØØ1D) TO BREAK (FFB3) THE PROGRAM COUNTER RECALCULATION, IN ØØ1B, SHOULD BE ØØ. EACH INSTRUCTION EXECUTED CAUSES THE MONITOR TO DISPLAY THE STATUS OF THE PROCESSOR, PRESSING R CAUSES THE NEXT INSTRUCTION TO BE EXECUTED. YOU MAY USE THE MONITOR TO ALTER A,X,Y (LOCATIONS) ØØØA, B & C) OR P (AT STACK POINTER + 1), BEFORE THE NEXT STEP. IT IS INADVISABLE TO CHANGE PC (STACK POINTER +2 & +3), BUT THIS CAN BE DONE AS WELL. THE SINGLE STEP EXECUTION CAN BE STOPPED IN TWO WAYS

I GROUND NMI LINE/GROUND THE SET INPUT OF THE D FLIP-FLOP

II POINT THE NMI VECTOR AT AN RTI INSTRCTION, SAY THE ONE AT FFID (EXECUTION OF A PROGRAM WILL BE SLOWED DOWN BY A FACTOR OF 5 OR SO DUE TO THE PERSISTENT NMI'S.)

AN IMPORTANT NOTE: THE BREAK ROUTINE SETS THE REPEAT LOCATION TO FF, SO THAT IT, AND THE MONITOR, MAY SAFELY USE THE DISPLAY ROUTINE. IF YOU NEED TO USE SINGLE SCANS AND BREAKS TO THE BREAK ROUTINE, SOME INGENUITY WILL BE REQUIRED, OR SOME DEDICATED BUTTON PUSHING.

NOW THE COMPLETE MONITOR LISTING. THIS IS WRITTEN TO FIT IN THE TWO 512 X 4 PROMS.

AFTER THE ADDRESS IS SET UP, THEN ANY KEY WILL CHANGE THE STATE OF IT'S CONTENTS: IF NOT A BREAK, A BREAK IS INSERTED, THE ORIGINAL DATA IS SAVED IN LOCATION ØØ18. IF A BREAK, THEN THE CONTENTS OF ØØ18 ARE INSERTED. THE RESULTING STATE OF THE LOCATION IS DISPLAYED

P. Ø2ØØ . ØØ-

WE ARE NOW BACK AT FFØ4. BUT ↑ & ↓ NOW OPERATE ON THE P ADDRESS. CONTENTS OF A LOCATION MAY BE CHANGED AS IF THIS WERE M. PRESSING P TWICE WILL INSERT A BREAKPOINT (ONLY A SINGLE LOCATION'S BACK-UP COPY IS RETAINED) AND SEND YOU BACK TO FFØ4. THE M KEY WILL RETURN IT'S MEMORY ADDRESS WHEN PRESSED

NOW THE PROGRAM IS SITTING THERE WITH A BREAK AT Ø2ØØ. EXECUTION OF THIS BREAK WILL CAUSE AN IRQ AND CONTROL IS TRANSFERRED TO THE ADDRESS IN LOCATION ØØIE & ØØ1F: FOR DIAGNOSTICS THIS ADDRESS SHOULD BE FFB3 (THE B3 IN ØØIE & THE FF IN ØØIE) ALSO IF THE PROGRAM COUNTER REQUIRES RESETTING AFTER A BREAK. THE AMOUNT BY WHICH THIS IS DONE, Ø2, SHOULD BE STORED IN LOCATION ØØIB NOW EXECUTING THE BREAK CAUSES THE STATUS OF THE PROCESSOR TO BE DISPLAYED IN THE FOLLOWING FORM

FIRST DISPLAY SET : | A | X | Y | P | (HEX PAIRS OF DATA IN EACH)

SECOND DISPLAY SET: | PC | SP | (TWO BYTES EACH, SECOND SET DISPLAYED AFTER ANY KEY IS PRESSED).

THIS PROGRAM

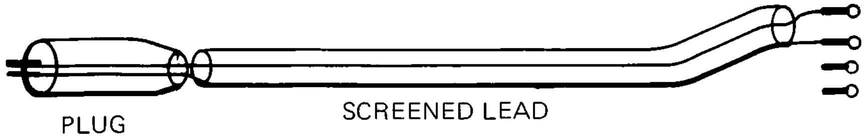
Ø2ØØ	78	SEI	–SET INTERRUPT DISABLE
Ø2Ø1	B8	CLV	–CLEAR OVERFLOW
Ø2Ø2	18	CLC	–CLEAR CARRY
Ø2Ø3	F8	SED	–SET DECIMAL MODE
Ø2Ø4	A9 11	LDA # 11	11
Ø2Ø6	A2 FF	LDX # FF	
Ø2Ø8	AØ33	LDY #33	33
Ø2Ø9	9A	TXS	–INITIALISE STACK
Ø2ØB	AZ 22	LDX #22	22
Ø2ØD	ØØ	BRK	
Ø2ØE			

CAUSES 1 1 2 2 3 3 3 C FOR THE FIRST DISPLAY SET AND

Ø 2 Ø D Ø 1 F C FOR THE SECOND SET.

THE ACTIVE FLAGS ARE THE DECIMAL AND INTERRUPT DISABLE FLAGS, (THE 2 PART OF THE STATUS REGISTER'S 2C IS AN UNUSED FLAG), THE PROGRAM WAS STOPPED AT LOCATION Ø2ØD WITH AN EMPTY STACK (THREE BYTES, PCH, PCL, P, WERE AUTOMATICALLY STACKED BY THE BRK INSTRUCTION). YOU MAY NOW CONTINUE TO WRITE (OR CORRECT) THE PROGRAM, USING THE MONITOR AS USUAL (BUT AVOID PRESSING THE RESET KEY SINCE THE STACKED PCH, PCL & P WILL BE DESTROYED) PRESSING THE R KEY WILL RETURN YOU TO Ø2ØØ TO TRY CONTINUING THE PROGRAM,

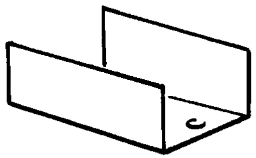
DONE IN THE PLUG CONNECTING TO THE RECORDER



BEST RECORDING RESULTS WITH A LEVEL OF ABOUT TWO-THIRDS MAXIMUM LEVEL. THE VERY CHEAPEST TAPE RECORDERS SOMETIMES USE A DC. ERASE SYSTEM, AND SUBSTANTIALLY POORER RESULTS MAY OCCUR ON RECORDING OVER AN ALREADY RECORDED SECTION OF TAPE. HIGH FREQUENCY RESPONSE IS AT A PREMIUM IN THIS APPLICATION, THE TAPE RECORDER'S HEADS SHOULD BE CLEANED FREQUENTLY, AND, PREFERABLY, DEMAGNETISED EVERY 8–1Ø HOURS. LOW QUALITY TAPES SHOULD BE AVOIDED SINCE THEY OFTEN CAUSE VERY FAST BUILD UP OF DIRT ON THE HEADS. THE SPEED OF THE REPLAYED DATA SHOULD NOT DEVIATE BEYOND ±5% OF THE RECORDED SPEED, SO DON'T USE BATTERIES FOR POWER, (OR C12Ø CASSETTES SINCE THE THINNER, HEAVIER TAPE OFTEN GETS STUCK). CLEAN THE EXPOSED CAPSTAN AND PRESSURE WHEEL WHEN YOU CLEAN THE HEADS: A HEAD CLEANING TAPE MAY NOT MANAGE TO REMOVE OXIDE BUILD-UP FROM THE MECHANISM.

5.6 POWER SUPPLY

THE TWO BOARDS ARE SUPPLIED BY THE 5V REGULATOR ON THE CPU BOARD. IF ALL THE I.C.S. ARE IN PLACE ON THE CPU BOARD, THEN AT LEAST 6ØØ MA IS REQUIRED. PROPER REGULATION IS ENSURED BY NEVER LETTING THE INPUT UNREGULATED SUPPLY DROP BELOW +7V. WHILE THE REGULATOR IS PERFECTLY HAPPY WITH +27V INPUT, IT WILL NEED TO DISSIPATE 13.2W AND WILL GET EXTREMELY HOT. . . AND TURN ITSELF OFF DUE TO THERMAL OVERLOAD, LOSING YOUR NICE PROGRAM IN THE R.A.M. UNLESS AN ADDITIONAL HEAT SINK IS USED, +12V SHOULD BE REGARDED AS AN ABSOLUTE MAXIMUM UNREGULATED INPUT, THE REGULATOR WILL NOT GET SO HOT AS TO TURN ITSELF OFF, BUT YOU MIGHT RECEIVE A BURN IF YOU TOUCH IT.



ADDITIONAL HEATSINK

CHAPTER 6: FIRMWARE

6.1 TAPE STORE AND LOAD

IN THE SOFTWARE SECTION WE USED SOME OF THE FUNCTIONS OF THE ACORN MONITOR TO WRITE AND EXECUTE SOME SIMPLE PROGRAMS WHICH DEMONSTRATED FEATURES OF THE MICROPROCESSOR AND PROGRAMMING. THE MONITOR IS MORE POWERFUL THAN DEMONSTRATED IN THAT SECTION, AND HERE WE'LL EXAMINE IT MORE CLOSELY, AND GIVE A COMPLETE LISTING OF IT. AFTER THE M, G, ↑ AND ↓ KEYS, THE MOST USEFUL KEYS WILL PROBABLY BE S AND L. THESE ENABLE YOU TO STORE AND LOAD PROGRAMS OF ANY SIZE USING CASSETTE TAPE OR A SIMILAR RECORDING MEDIUM. LET'S ASSUME WE WISH TO CREATE A TAPE VERSION OF THE DUCK-SHOOT GAME. THIS WILL HAVE BEEN ENTERED IN MEMORY FROM ADDRESS, SAY, Ø2ØØ TO ADDRESS Ø23F INCLUSIVE. AFTER TESTING THAT THE PROGRAM ACTUALLY DOES WORK, PRESS THE S KEY.

```
F.          XXXX    .
```

THE MONITOR IS PROMPTING YOU TO ENTER THE ADDRESS FROM WHICH YOU WANT TO RECORD. THE DISPLAYED ADDRESS IS EITHER GARBAGE OR THE LAST END ADDRESS USED. ENTER THE ADDRESS, TERMINATING WITH ANY COMMAND KEY

```
F.          Ø2ØØ    .
–           XXXX    .
```

THE MONITOR IS NOW PROMPTING YOU TO ENTER THE END ADDRESS. THIS IS THE ADDRESS OF THE LAST BYTE IN YOUR PROGRAM + 1. THE DISPLAYED ADDRESS IS EITHER GARBAGE OR THE LAST END ADDRESS USED. ENTER THE ADDRESS, BUT DON'T TERMINATE IT YET

```
–           Ø24Ø    .
```

THE SYSTEM IS NOW READY TO SERIALLY OUTPUT THAT SECTION OF MEMORY. YOU SHOULD RECORD A BRIEF VERBAL DESCRIPTION OF THE PROGRAM – "DUCKSHOOT" – AND ALSO THE ADDRESSES (OR ADDRESS OF START AND LENGTH) WHICH THE PROGRAM USES. KEEP A LIST OF WHICH PROGRAMS ARE STORED ON EACH TAPE. NOW CONNECT IN THE COMPUTER AND START RECORDING. AFTER A FEW SECONDS, PRESS ANY COMMAND KEY TO TERMINATE THE ADDRESS ENTRY. THE DISPLAY WILL GO BLANK, WHILE THE PROCESSOR DEVOTES ITSELF TO SENDING THE INFORMATION TO THE TAPE. WHEN THE DISPLAY

```
–           Ø24Ø    .
```

REAPPEARS, YOU MAY STOP THE TAPE-RECORDER: THE RECORDING IS COMPLETE, AND YOU ARE BACK AT FFØ4. ANY HEX KEY HERE WILL BRING BACK THE MONITOR'S DOTS, OR YOU MAY JUST START USING THE MONITOR. THE RECORDING PROCEEDS AT 3Ø BYTES PER SECOND, THIS PROGRAM, AT 68 BYTES (PROGRAM LENGTH + 4 BYTES OF ADDRESS INFORMATION) TOOK ONLY TWO SECONDS TO RECORD.

TO LOAD A PROGRAM FROM THE TAPE YOU SHOULD BE IN A SITUATION WHERE MONITOR COMMANDS ARE ACCEPTED, NOT WHERE YOU ARE ALLOWED ANY KEY TO TERMINATE AN ADDRESS ENTRY. PLAY THE TAPE, AND, WHEN THE 24Ø3.8 HZ LEADER IS HEARD, PRESS THE L KEY. THE DISPLAY WILL BE BLANK UNTIL DATA IS ENCOUNTERED ON TAPE, WHEN EACH BYTE ENTERED WILL BE DISPLAYED AS A SYMBOL ON THE LEFTMOST DIGIT. WHEN THE LAST BYTE HAS BEEN READ THE PREVIOUS DISPLAY WILL RETURN – YOU'RE AT FFØ4 AGAIN. THE ADDRESSES INTO WHICH THE PROGRAM IS LOADED WILL BE THOSE WITH WHICH IT WAS STORED ON TAPE, BUT YOU MAY WISH TO DELIBERATELY AVOID THIS. JUST USING THE MONITOR, THE BEST THAT CAN BE DONE IS TO TREAT THE ENTIRE RECORDING AS DATA AND LOAD ENOUGH OF IT TO FIT BETWEEN TWO ADDRESSES: THE FIRST FOUR BYTES LOADED WILL THUS BE THE ORIGINAL ADDRESSES THE PROCEDURE IS

I SET ADDRESSES ØØØ8 & ØØØ9 TO THE LOW & HIGH BYTE OF THE ADDRESS INTO WHICH YOU WISH TO PUT THE FIRST BYTE.
II SET ADDRESSES ØØØA & ØØØB TO THE LOW & HIGH BYTE OF THE LAST ADDRESS +1 INTO WHICH YOU WANT THE DATA TO BE LOADED.
III SET UP THE GO ADDRESS OF FF8A, START THE PLAYBACK, WHEN YOU HEAR THE 24Ø3.8 HZ LEADER, PRESS ANY KEY TO GO. LOADING WILL OCCUR BETWEEN THE ADDRESSES SPECIFIED.

THE ABOVE PROCEDURE MAY NOT BE SATISFACTORY: IT LOADS THE PROGRAM'S ADDRESSES AS DATA, AND DESTROYS THE DATA IN REGISTERS Ø AND 1 (A & X AFTER A BREAKPOINT) BETTER METHODS ARE GIVEN IN THE SYSTEM SECTION OF THE APPLICATION PROGRAMS
THE LAST COMMENT ON LOAD FROM TAPE IS THAT IT IS POSSIBLE TO CREATE A PROGRAM ON TAPE THAT WILL, WHEN LOADED, SEIZE CONTROL AND EXECUTE ITSELF. THIS IS IDEAL FOR, SAY, A BASIC INTERPRETER: YOU JUST HAVE TO LOAD IT, AND IT AUTOMATICALLY SETS ITSELF RUNNING AND PROMPTS READY. THE IDEA IS TO LOAD THE PROGRAM INTO THE MONITOR'S ZERO PAGE REGISTERS, LOADING THE PROGRAM START ADDRESS INTO GAP AND THE GO KEY (II) INTO REPEAT. CARE MUST BE TAKEN WHEN YOU LOAD INTO FAP AND TAP: YOU MUST BE SURE TO LOAD WHAT'S ALREADY THERE, OR SOMETHING SENSIBLE!

6.2 THE BREAKPOINT AND RESTORE COMMAND

THE FINAL TWO MONITOR FUNCTIONS ARE EMBODIED BY THE KEYS R AND P. YOU MAY ALREADY HAVE DISCOVERED THAT PRESSING R IS DISASTROUS, AND THAT P IS LIKE M, BUT WITH A PENCHANT FOR INSERTING ØØ INTO THE ADDRESS SPECIFIED. WITH THESE KEYS YOU ARE EXPECTED TO DEBUG (A BUG IS ANY SMALL MISTAKE PREVENTING A PROGRAM FROM FUNCTIONING) YOUR PROGRAMS. THE P KEY ALLOWS YOU TO INSERT THE BREAK INSTRUCTION ON TOP OF AN INSTRUCTION AT A POINT WHERE YOU SUSPECT SOMETHING SUSPICIOUS IS HAPPENING, SAY Ø2ØØ:

```
P.          Ø2ØØ    .
```