6 Assembling and Disassembling

6.1 The Assembler

The built-in 6502 assembler in BASIC is a very useful tool, allowing both large and small machine code routines to be written easily. Being a part of BASIC itself, it is very easy to use BASIC variables and functions, conditional assembly (with some sections of the assembly code in IF..THEN statements), or macros (assembly sections in a GOSUB or FN/PROC).

The assembler is written very efficiently, and in total only occupies just over 1K of the 16K BASIC ROM.

The assembler mnemonics in the ROM are stored in a compressed format to save space. Only the least significant 5 bits of each mnemonic character are used, so that the whole mnemonic can be compressed into 15 bits of a 2-byte number. This also mearvs that both upper case or lower case mnemonics will be recognised (or a mixture of the two). Fig 6.1 shows how the characters are packed.

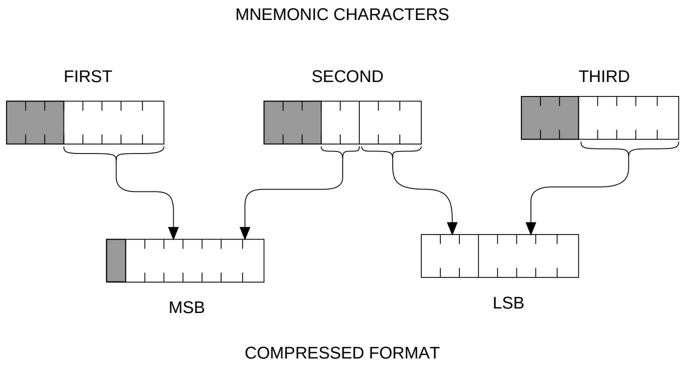

Figure 6.1. − Mnemonic compression.

A further byte is used for each mnemonic, to hold the ‘base value of the opcode. For instructions which can only have one addressing mode (such as the instructions which employ implied or relative addressing), this is the actual opcode used; for other intructions, this base value is modified by the actual addressing mode used.

The mnemonic and base opcode are stored as follows:

BASIC1	**BASIC2**	
&843B+M	&8450+M	MSB mnemonic
&8474+M	&848A+M	LSB mnemonic
&84AD+M	&84C4+M	base opcode

where M is the mnemonic number. Table 6.1 shows the mnemonic and base opcode value for each mnemonic number, as stored in the ROM table. Note that the directives OPT and EQU are stored the same as mnemonics, but they need no base opcode. The EQU directive is not implemented in BASIC1

By comparing this table with fig 6.2, it can be seen that the mnemonics are grouped together with others which allow the same addressing modes. The assembler has a different section of machine code which is used for each of the different groups of mnemonics, to decide which addressing modes to allow. Section 1.5 gives these mnemonic groups.

Table 6.1 − Assembler Mnemonics

No.	Mnemonic	Base	No.	Mnemonic	Base
&01	BRK	&00	&0F	RTI	&40
&02	CLC	&18	&10	RTS	&60
&03	CLD	&D8	&11	SEC	&38
&04	CLI	&58	&12	SED	&F8
&05	CLV	&B8	&13	SEI	&78
&06	DEX	&CA	&14	TAX	&AA
&07	DEY	&88	&15	TAY	&A8
&08	INX	&E8	&16	TSX	&BA
&09	INY	&C8	&17	TXA	&8A
&0A	NOP	&EA	&18	TXS	&9A
&0B	PHA	&48	&19	TYA	&98
&0C	PHP	&08	&1A	BCC	&90
80D	PLA	&68	&1B	BCS	&B0
80E	PLP	&28	&1C	BEQ	&F0

No.	Mnemonic	Base	No.	Mnemonic	Base
&1D	BMI	&30	&2C	ROR	&66
&1E	BNE	&D0	&2D	DEC	&C6
&1F	BPL	&10	&2E	INC	&E6
&20	BVC	&50	&2F	CPX	&E0
&21	BVS	&70	&30	CPY	&C0
&22	AND	&21	&31	BIT	&20
&23	EOR	&41	&32	JMP	&4C
&24	ORA	&01	&33	JSR	&20
&25	ADC	&61	&34	LDX	&A2
&26	CMP	&C1	&35	LDY	&A0
&27	LDA	&A1	&36	STA	&81
&28	SBC	&E1	&37	STX	&86
&29	ASL	&06	&38	STY	&84
&2A	LSR	&46	&39	OPT	---
&2B	ROL	&26	&3A	EQU	---

MSD \ LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	BRK Implied 1 7	ORA (IND, X) 2 6				ORA ZP 2 3	ASL ZP 2 5		PHP Implied 1 3	ORA IMM 2 2	ASL Accum 1 2			ORA ABS 3 4	ASL ABS 3 6		0
1	BPL Relative 2 7	ORA (IND, Y) 2 5*				ORA ZP, X 2 4	ASL ZP, X 2 6		CLC Implied 1 2	ORA ABS, Y 3 4*				ORA ABS, X 3 4*	ASL ABS, X 3 7		1
2	JSR Absolute 3 6	AND (IND, X) 2 6			BIT ZP 2 3	AND ZP 2 3	ROL ZP 2 5		PLP Implied 1 4	AND IMM 2 2	ROL Accum 1 2		BIT ABS 3 4	AND ABS 3 4	ROL ABS 3 6		2
3	BMI Relative 2 2**	AND (IND), Y 2 5*				AND ZP, X 2 4	ROL ZP, X 2 6		SEC Implied 1 2	AND ABS, Y 3 4*				AND ABS, X 3 4*	ROL ABS, X 3 7		3
4	RTI Implied 1 6	EOR (IND, X) 2 6				EOR ZP 2 3	LSR ZP 2 5		PHA Implied 1 3	EOR IMM 2 2	LSR Accum 1 2		JMP ABS 3 3	EOR ABS 3 4	LSR ABS 3 6		4
5	BVC Relative 2 2**	EOR (IND), Y 2 5*				EOR ZP, X 2 4	LSR ZP, X 2 6		CLI Implied 1 2	EOR ABS, Y 3 4*				EOR ABS, X 3 4*	LSR ABS, X 3 7		5
6	RTS Implied 1 6	ADC (IND, X) 2 6				ADC ZP 2 3	ROR ZP 2 5		SED Implied 1 4	ADC IMM 2 2	ROR Accum 1 2		JMP ABS 3 5	ADC ABS 3 4	ROR ABS 3 6		6
7	BVS Relative 2 2**	ADC (IND), Y 2 5*				ADC ZP, X 2 4	ROR ZP, X 2 6		SEI Implied 1 2	ADC ABS, Y 3 4*				ADC ABS, X 3 4*	ROR ABS, X 3 7		7
8		STA (IND, X) 2 6			STY ZP 2 3	STA ZP 2 3	STX ZP 2 3		DEY Implied 1 2		TXA Implied 1 2		STY ABS 3 4	STA ABS 3 4	STX ABS 3 4		8
9	BCC Relative 2 2**	STA (IND), Y 2 6			STY ZP, X 2 4	STA ZP, X 2 4	STX ZP, Y 2 4		TYA Implied 1 2	STA ABS, Y 3 5	TXS Implied 1 2			STA ABS, X 3 5			9
A	LDY IMM 2 2	LDA (IND, X) 2 6	LDX IMM 2 2		LDY ZP 2 3	LDA ZP 2 3	LDX ZP 2 3		TAY Implied 1 2	LDA IMM 2 2	TAX Implied 1 2		LDY ABS 3 4	LDA ABS 3 4	LDX ABS 3 4		A
B	BCS Relative 2 2**	LDA (IND), Y 2 5*			LDY ZP, X 2 4	LDA ZP, X 2 4	LDX ZP, Y 2 4		CLV Implied 1 2	LDA ABS, Y 3 4*	TSX Implied 1 2		LDY ABS, X 3 4*	LDA ABS, X 3 4*	LDX ABS, Y 3 4*		B
C	CPY IMM 2 2	CMP (IND, X) 2 6			CPY ZP 2 3	CMP ZP 2 3	DEC ZP 2 3		INY Implied 1 2	CMP IMM 2 2	DEX Implied 1 2		CPY ABS 3 4	CMP ABS 3 4	DEC ABS 3 6		C
D	BNE Relative 2 2**	CMP (IND), Y 2 5*				CMP ZP, X 2 4	DEC ZP, X 2 5		CLD Implied 1 2	CMP ABS, Y 3 4*				CMP ABS, X 3 4*	DEC ABS, X 3 7		D
E	CPX IMM 2 2	SBC (IND, X) 2 6			CPX ZP 2 3	SBC ZP 2 3	INC ZP 2 5		INX Implied 1 2	SBC IMM 2 2	NOP Implied 1 2		CPX ABS 3 4	SBC ABS 3 4	INC ABS 3 6		E
F	BEQ Relative 2 2**	SBC (IND), Y 2 5*				SBC ZP, X 2 4	INC ZP, X 2 6		SED Implied 1 2	SBC ABS, Y 3 4*				SBC ABS, X 3 4*	INC ABS, X 3 7		F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

0	0	
0	BRK	— OP Code
	Implied	— Addressing Mode
	1 7	— Instruction Bytes; Machine Cycles

* Add 1 to N if page boundary is crossed.
** Add 1 to N if branch occurs to same page.
add 2 to N if branch occurs to different page.

Figure 6.2 − 6502 op-code matrix.

6.2 The Disassembler

A disassembler is always useful: either for exploring the contents of the ROMs in the machine, or for checking that the machine code that you have just assembled is actually what you wanted (especially if its got lots of conditional assembly in it).

Most disassemblers take up quite a lot of memory. For a start, they usually use a large table to decode the opcodes, with one entry for each of the 256 possible l-byte numbers. Each entry of the table contains 3 bytes of mnemonic characters, and a further byte to give the addressing modes allowed with that particular opcode. This means that the disassembler is 1K long already, without any program to decode the instructions. Also, they are usually written in BASIC, which makes them slow, and even larger.

The disassembler described in this section uses the assembler tables in the ROM, and is written in machine code. When assembled, it is less than 500 bytes long, and so will fit in any 2 spare pages of memory (for example, from &B00 to &CFF, which is otherwise used for the user defined characters and function keys).

To use the disassembler, the resident integer variable D% is set to point to the first instruction to be disassembled (similar to the use of P% by the assembler). Typing ‘CALL start ’ will then disassemble one instruction, and leave D% pointing to the next one to be disassembled. If the variables have been re-set since the program was assembled, ‘CALL &B00 , or wherever the start of it is, will have to be used instead. This could be built in as a new statement, if required (see chapter 7).

To disassemble a length of code, a loop can be used:

```
        REPEAT:CALL &B00:UNTIL FALSE
or:     REPEAT:CALL &B00:UNTIL D%>&BFFF
```

(page mode will have to be used with a loop like this, as it disassembles at about 150 bytesfisecond, depending on the screen mode). In fact, a short program could be used to make the use of it very flexible; but the main advantage of it is that other programs can be loaded and run while the disassembler is still resident. If the user defined characters or function keys need to be used while the disassembler is in memory, PAGE could be moved up by 512 bytes, and it could be assembled there.

The ‘EQU’ directive has not been used in the program, so that it will work on either a BASICI or BASIC2 machine with no modification. PROCsetup (lines 9000 on) checks which version of BASIC is present, and sets up th.e correct ROM table labels before it is assembled.

Operation of the disassembler

The disassembler compares the opcode which is to be disassembled against the ‘base opcode’ of each mnemonic, and calculates the difference between them. If this difference can be made up by the offset of a particular addressing mode, and this addressing mode is allowed with the current mnemonic that it is checking, the mnemonic and addressing mode of that particular opcode have been found.

For example, if the value of the opcode was &31, this would be matched with the mnemonic ‘AND’ (base opcode &21) and the addressing mode ‘(IND), Y’ (offset &10). The base opcodes for each mnemonic are stored in the ROM tables, but the disassembler must contain the tables of allowed addressing modes for each group of instructions, and also the extent of each group. These tables are not in the ROM as the assembler does the addressing mode decoding in machine code rather than using tab les.

The main opcode matching loop is from lines 1460−1760.

If the opcode is not matched with anything in the table, ‘???’ is printed out (for an unrecognised mnemonic). Note that ‘JMP (IND)’ has to be tested for separately (line 1190) as it does not fit into the pattern with the rest of them.

The allowed addressing mode offsets for each group are:

Addressing					Offset				
mode-grp.		00	04	08	0C	10	14	18	1C
0	&00−&21	X							
1	&22−&28	0	1	2	3	4	5	6	7
2	&29−&2C	1	A	3		5		7	
3	&2D−&2E	1		3		5		7	
4	&2F−&30	#	1		3				
5	&31		1		3				
6	&32−&33	3							
7	&34−&35	#	1		3		5		7
8	&36	0	1		3	4	5	6	7
9	&37−&38	1		3		5			

These possible offsets are held in the bit table ‘msktab’ in the program (lines 3490−3590). The number of the lowest mnemonic in each group is held in the table ‘grptab’ (lines 3600−3710).

The symbols in the table (X, #, A, 1 to 7) represent the possible addressing modes. Note that they don’t all line up: the addressing mode decode part of the program has to line up all these to get the correct addressing mode. The symbols represent:

X	either relative or implied
#	IMM (same as 2, but different pattern)
0	(IND,X)
1	ZP
2	IMM
3	ABS
4	(IND),Y
5	ZP,X
6	ABS,Y
7	ABS,X (,Y if LDX or STX)

The rest of the program handles the decoding and printing of the addressing mode characters and data. For most of the groups this is not too difficult, as the addressing mode corresponds directly with the offset from the base address; however, some others need to be shifted by an extra offset to ‘line up’ with the others. This shifting is done by lines 1810−2060.

The more complex addressing modes are printed using a bit mask table (lines 3800 to 3882) to decide which characters to print. The simpIer addressing modes are printed by a separate part of the routine.

```
  10 REM Machine code disassembler
  15 REM using assembler ROM tables
  20 REM
  25 REM      M D Plumbley 1984
  30 REM
  99
 100 PROCsetup            :REM Set up ROM entry points
 590
 595 REM *** Allocate workspace ***
 600 worksp = &0070
 605 grpmsk = worksp      :REM Bit mask of allwed modes
 610 ytemp = worksp+1     :REM Temp for addr mode group
 615 mdstor = worksp+2    :REM Store for addressing mode
 620 opcode = worksp+3    :REM Opcode read in from memory
 625 data = worksp+4      :REM The 2 bytes after the opcode
 630 addr = worksp+6      :REM Copy of address in D%
 635 mnem = worksp+8      :REM Mnemonic construction area
 640 xtemp = worksp+10    :REM Temp for mnemonic number
 645 lastch = worksp+11   :REM Last char of mnemonic
 650 nbytes = worksp+12   :REM Number of instruction bytes
 655 chrmsk = worksp+13   :REM Addr mode character mask
 690
 700 count = &1E
 799
 900 start% = &0B00       :REM User defined charlkey area
 905
 910 FOR opt% = 0 TO 3 STEP 3
 920 P% = start%
 950 [    OPT opt%
1000 .disass
1010     LDA &410         \Get address from D%, and put it
1020     STA addr         \ in the workspace
1030     LDA &411
1040     STA addr+1
1045
1050     LDY #2           \Transfer the opcode and 2 data
1060 .txbyte              \ bytes to be disassembted
1070     LDA (addr),Y
1080     STA opcode,Y
1090     DEY
1100     BPL txbyte
1105
1110     LDA addr+1       \Print the address and the opcode
1120     JSR phex
1130     LDA addr
```

```
1140     JSR phexsp
1150     JSR pspace
1160     LDA opcode
1170     JSR phexsp
1180
1190     LDA opcode       \If we have a JMP (XXXX), then
1200     CMP #&6C         \ set the mnemonic to "JMP"
1210     BNE mtchop       \ (mnemonic number &32),and the
1220     LDX #&32         \ addressing mode to 8.
1230     STX xtemp        \ Otherwise, attempt to match the
1240     LDA #8           \ opcode with the table
1250     STA mdstor
1260     JMP domode
1270
1280 .nomtch
1290     JSR tbmnem       \If we get here, no match was
1300     LDY #3           \ found, so print a "???",
1310     LDA #ASC"?"      \ and go on to add 1 to D%
1320 .pqloop              \ before finishing
1330     JSR pchar
1340     DEY
1350     BNE pqloop
1360     JMP add1
1370
1380 .tbmnem
1390     LDY #16          \Print spaces until we get to
1400 .tbloop              \ the 16th cotumn. This lines
1410     JSR pspace       \ up all the mnemonics.
1420     CPY count
1430     BCS tbloop
1440     RTS
1450
1451     \ ** Main opcode matching routine **
1452
1460 .mtchop              \Go through all the mnemonics,
1470     LDX #&39         \ and try to match one to the
1480     LDY #&0A         \ opcode.
1485
1490 .nextop
1500     DEX              \If ws have tried all the
1510     BEQ nomtch       \ mnemonics, it is invatid.
1515
1520     TXA              \Check to see if we are now in
1530     CMP grptab,Y     \ a new mnemonic group.
1540     BCS samgrp
1550     DEY
1560     LDA msktab,Y
1570     STA grpmsk
1575
1580 .samgrp
1590     LDA opcode       \The opcode can only have this
1600     SEC              \ mnemonic if is a positive
```

```
1610     SBC opbase,X     \ offset from the "baseopcode"
1620     BCC nextop       \ of it. Also, the offset must
1630     LSRA             \ be divisible by 4, and must be
1640     BCS nextop       \ &1C or less (&1C=4*7)
1650     LSRA
1660     BCS nextop
1670     CMP #8
1680     BCS nextop
1685
1690     STA mdstor       \Check to see if this addr mode
1700     STY ytemp        \ is attowed with this mnemonic.
1710     TAY              \ If it isn't, go back to check
1720     LDA bittab,Y     \ for another mnemonic.
1730     LDY ytemp        \ "grpmsk" holds the allowed
1740     AND grpmsk       \ addr modes for this mnemonic.
1750     BEQ nextop
1755
1760     STX xtemp        \Success!! - so save the mnemonic
1762                      \ number
1765
1770     LDY ytemp        \If the mode group is 0, it is
1790     TYA              \ either implied or relative
1800     BEQ imprel
1805
1810     LDA #&10         \If the group masksuggests that
1820 .trymsk              \ the mnemonic doesn'tallou
1830     BIT grpmsk       \ absotute addressing, w. have to
1840     BNE mskok        \ atter the addressing mode untit
1850     INC mdstor       \ it does. (The "BPL" will always
1860     LSR grpmsk       \ work after a "LSR".)
1870     BPL trymsk
1875
1880 .mskok               \When we get here, the mask and
1890     LDA grpmsk       \ addr mode offset is OK.
1900     AND #&08         \ Homever, if the addr mode is 0
1910     BNE modeok       \ and (indir),Y is not attowed,
1920     LDA mdstor       \ then it is really immediate
1930     BNE modeok       \ addressing, which should be
1940     LDA #2           \ addr mode 2
1950     STA mdstor
1955
1960 .modeok              \When we get here, the only thing
1970     CPY #2           \ left to test for is accumulator
1980     BNE domode       \ addressing. If the "allowed
1990     TYA              \ mode" group is 2, and the addr
2000     CMP mdstor       \ mode is also 2, then print the
2010     BNE domode       \ mnemonic, followed by an "A",
2020     JSR pmnem        \ and go to add 1 to D% before
2030     LDA #ASC"A"      \ finishing. Otherwise, go to
2040     JSR pchar        \ "domode".
2050 .jadd1
2060     JMP add1
```

```
2065
2070 .imprel              \If we get here, the addressing
2080     LDX xtemp        \ mode is either retative or
2090     CPX #&1A         \ imptied.
2100     BCS rel
2105
2110     JSR pmnem        \If it is imptied, print the
2120     JMP add1         \ mnemonic, and add 1 to D%
2125
2130 .rel                 \If it is relative, we have 1
2140     LDA data         \ extra data byte to print out
2150     JSR phexsp       \ before the mnemonic.
2160     JSR pmnem
2165
2170     LDA #0           \The absolute addr has to be
2180     STA data+1       \ calculated from the offset.
2190     LDA data         \ First extend the sign of the
2200     BPL nodec        \ offset byte into 2 bytes
2210     DEC data+1
2215
2220 .nodec               \Then add this 2-byte offset to
2230     SEC              \ D%, adding another 2 with it.
2240     ADC &410         \ One extra is added by setting
2250     STA data         \ the carry before the addition,
2260     LDA &411         \ the other is added by
2270     ADC data+1       \ incrementing the address
2280     STA data+1       \ afterwards.
2290     INC data
2300     BNE nopage
2310     INC data+1
2315
2320 .nopage              \Finally, print the absotute
2330     JSR pabs         \ address, and add 2 to D% before
2340     JMP add2         \ leaving.
2350
2355     \ ** Print the mnemonic ***
2360 .pmnem
2370     LDX xtemp        \First, get the number of the
2380     JSR tbmnem       \ mnemonic, and get the LSB and
2390     LDA lsbmn,X      \ MSB of the compressed mnemonic.
2400     ASLA             \ The shifts are to get the bits
2410     STA mnem         \ ready for the first 5 bits to
2420     LDA msbmn,X      \ be shifted out.
2430     ROLA
2440     STA mnem+1
2445
2450     LDX #3           \This is the main loop which
2460 .mcloop              \ shifts 3 characters out of
2470     LDA #0           \ the 2-byte compressed mnemonic.
2480     LDY #5           \ 5 bits at a time are shifted
2490 .mbloop              \ out into the accumutator, and
2500     ASL mnem         \ they are then ORed with &40 to
```

```
2510     ROL mnem+1       \ turn them into upper case
2520     ROLA             \ letters .
2530     DEY
2540     BNE mbloop
2550     ORA #&40
2560     JSR pchar
2570     DEX
2580     BNE mcloop
2585
2590     STA lastch       \Save the last character printed:
2595                      \ it might be an "X".
2600     JMP pspace       \Print a space, and exit.
2605
2606     \ ** Handle the addressing mode stuff **
2610 .domode
2620     LDY mdstor       \First, get the number of bytes
2630     LDX mdbyts,Y     \ used by this addr mode, and
2640     STX nbytes       \ save it.
2645
2650     DEX              \Print the required number of
2660     BEQ nodata       \ data bytes before the mnemonic.
2670     LDA data
2680     JSR phexsp
2690     DEX
2700     BEQ nodata
2710     LDA data+1
2720     JSR phexsp
2725
2730 .nodata
2740     JSR pmnem        \Print the mnemonic.
2745
2750     LSR mdstor       \If the addr mode was odd, it is
2760     BCS smplmd       \ a simple one, so deal with it
2770
2780     LDY mdstor       \If it was not a simple mode, get
2790     LDA chmstb,Y     \ the mask of characters to be
2800     STA chrmsk       \ printed into "chrmsk".
2805
2810     LDY #6           \Starting at the 7th (0..6) char,
2820 .newchr              \ if the bit shifted out of the
2830     ASL chrmsk       \ mask is set, then print it.
2840     BCC nochr
2850     LDA chtab,Y
2860     JSR pchar
2865
2870 .nochr               \If we have got to the 5th char,
2880     CPY #5           \ the data can be printed (i.e.
2890     BNE nodat        \ the "#" or "(" has been printed
2900     JSR pdata        \ if there was one)
2905
2910 .nodat               \Go round for another character
2920     DEY              \ if we haven't printed them all;
```

```
2930     BPL newchr       \ otheruise add "nbytes" to D%
2940     JMP addn         \ and exit.
2950
2960 .smplmd              \If we get here, the addr mode is
2970     JSR pdata        \ either "zero-page", "absotute",
2980     LSR mdstor       \ "zero-page,X" or "absolute,X".
2990     LSR mdstor       \ Shifting out the 2nd bit from
3000     BCC addn         \ "mdstor" gives whether indexed
3010     LDA #ASC","      \ addressing is required.
3020     JSR pchar
3025
3030     LDA #ASC"X"      \If the last character of the
3040     CMP lastch       \ mnemonic was a "X", then use
3050     BNE px           \ "Y" as the index
3060     LDA #ASC"Y"
3070 .px
3080     JSR pchar        \Print the index character, and
3090     JMP addn         \ add "nbytes" to D%.
3095
3096     \ ** Routines to print the data after the mnemonic **
3110 .pabs                \Print the data as an absotute
3120     LDA #ASC"&"      \ address.
3130     JSR pchar
3140     LDA data+1
3150     JSR phex
3160     LDA data
3170     JMP phex
3175
3180 .pdata               \If the total number of bytes for
3190     LDA nbytes       \ this addressing mode is not 2
3200     CMP#2            \ (i.e. it is 3) then print the
3210     BNE pabs         \ absolute address.
3220 .pzerop
3230     LDA #ASC"&"      \Print the data as a single byte.
3240     JSR pchar
3250     LDA data
3260     JMP phex
3265
3267 \** Exit points; add size to D% and exit ***
3270 .add1                \Add 1 to D%, and then exit
3280     LDA #1
3290     BNE add
3300 .add2                \Add 2 to D%, and then exit
3310     LDA #2
3320     BNE add
3360 .addn                \Add the number of bytes in the
3370     LDA nbytes       \ instruciton to D%, then exit
3375
3380 .add\Add A to D%
3390     CLC              \ (The least significant 2 bytes
3400     ADC &410         \ of D%, are stored in &410 and
3410     STA &410         \ &411)
```

```
3420     LDA &411
3430     ADC #0
3440     STA &411
3445
3450     JMP pnewl        \Print a CRLF and exit
3460
3480 \*** Allowed offset table ***
3482 \This tabte gives the allowed addr mode offset for
3484 \ each group of mnemonics. Bit 7 (the top bit) is set
3486 \ if 0 is allowed; bit 6 set if 4 is allowed; etc.
3490 ]:msktab=P%:P%=P%+10
3500 msktab?0 = &80
3510 msktab?1 = &FF
3520 msktab?2 = &EA
3530 msktab?3 = &AA
3540 msktab?4 = &D0
3550 msktab?5 = &50
3560 msktab?6 = &80
3570 msktab?7 = &D5
3580 msktab?8 = &DF
3590 msktab?9 = &A8
3592
3594 REM ** Addressing mode groups **
3596 REM This table contains the starts of the mnemonics
3598 REM which have the same allowed addressing modes
3600 grptab=P%:P%=P%+11
3610 grptab?&0 = &01
3620 grptab?&1 = &22
3630 grptab?&2 = &29
3640 grptab?&3 = &2D
3650 grptab?&4 = &2F
3660 grptab?&5 = &31
3670 grptab?&6 = &32
3680 grptab?&7 = &34
3690 grptab?&8 = &36
3700 grptab?&9 = &37
3710 grptab?&A = &39
3712
3714 REM *** Bit position table ***
3716 REM This table contains the bit position corresponding
3718 REM to each addressing mode
3720 bittab=P%:P%=P%+8
3730 bittab?0 = &80
3740 bittab?1 = &40
3750 bittab?2 = &20
3760 bittab?3 = &10
3770 bittab?4 = &08
3780 bittab?5 = &04
3790 bittab?6 = &02
3800 bittab?7 = &01
3802
3804 REM *** Addr mode character mask table ***
```

```
3806 REM This table gives the characters to be printed for
3808 REM the non-simple addressing modes
3810 chmstb=P%:P%=P%+5
3820 chmstb?0 = &78       :REM "(,X)"
3830 chmstb?1 = &80       :REM "#"
3840 chmstb?2 = &4E       :REM "(),Y"
3850 chmstb?3 = &06       :REM ",Y"
3860 chmstb?4 = &48       :REM "()"
3870 chtab=P%:P%=P%+7
3880 $chtab="Y,)X,(#"
3882
3884 REM *** Addressing mode bytes table ***
3886 REM This table gives the total number of bytes used by
3888 REM a given addressing mode.
3890 mdbyts=P%:P%=P%+9
3900 mdbyts?0 = 2
3910 mdbyts?1 = 2
3920 mdbyts?2 = 2
3930 mdbyts?3 = 3
3940 mdbyts?4 = 2
3950 mdbyts?5 = 2
3960 mdbyts?6 = 3
3970 mdbyts?7 = 3
3980 mdbyts?8 = 3
8000
8010 NEXT
8015 @%=0
8020 PRINT'"Code length =&"~P%-start%
8190
8200 PRINT'''''"** WARNING: Once assembled, the code"
8210 PRINT"generated by this program is not"
8220 PRINT"transferable between different BASICs"
8230 PRINT
8300 PRINT"DO ""CALL &"~disass""" to disassemble 1 line"
8305 PRINT"D% points to code to be disassembled"'
8810 END
8990
8992 REM *** Set up ROM entry points, allowing for ***
8993 REM *** BASIC 1 and BASIC 2.    ***
9000 DEFPROCsetup
9010 basic1$ = "BASIC"+CHR$0+"(C)1981 Acorn"+CHR$&A
9020 basic2$ = "BASIC"+CHR$0+"(C)1982 Acorn"+CHR$&A
9030 IF $&8009=basic1$ THEN PROCset1 :ENDPROC
9040 IF $&8009=basic2$ THEN PROCset2 :ENDPROC
9050 PRINT "NOT BASIC 1 OR 2"
9060 END
9290
9292 REM *** Set up BASIC 1 entry points ***
9300 DEFPROCset1
9310 opbase = &84AD       :REM Opcode base vatue table
9315 lsbmn  = &843B       :REM Tabte of LSB of mnemonic
9320 msbmn  = &8474       :REM Tabte of MSB of mnemonic
```

```
9325 phex   = &8570        :REM Print A as a HEX byte
9330 phexsp = &856A        :REM Print A in HEX, then space
9335 pspace = &B57B        :REM Print a space
9340 pnewl  = &BC42        :REM Print a CRLF
9345 pchar  = &B571        :REM Print char in A
9350 ENDPROC
9490
9492 REM *** Set up BASIC 2 entry points ***
9500 DEFPROCset2
9510 opbase = &84C4        :REM Opcode base vatue tabte
9515 lsbmn  = &8450        :REM Table of LSB of mnemonic
9520 msbmn  = &848A        :REM Table of MSB of mnemonic
9525 phex   = &B545        :REM Print A as a HEX bytes
9530 phexsp = &B562        :REM Print A in HEX, then space
9535 pspace = &B565        :REM Print a space
9540 pnewl  = &BC25        :REM Print a CRLF
9545 pchar  = &B558        :REM Print char in A
9550 ENDPROC
```