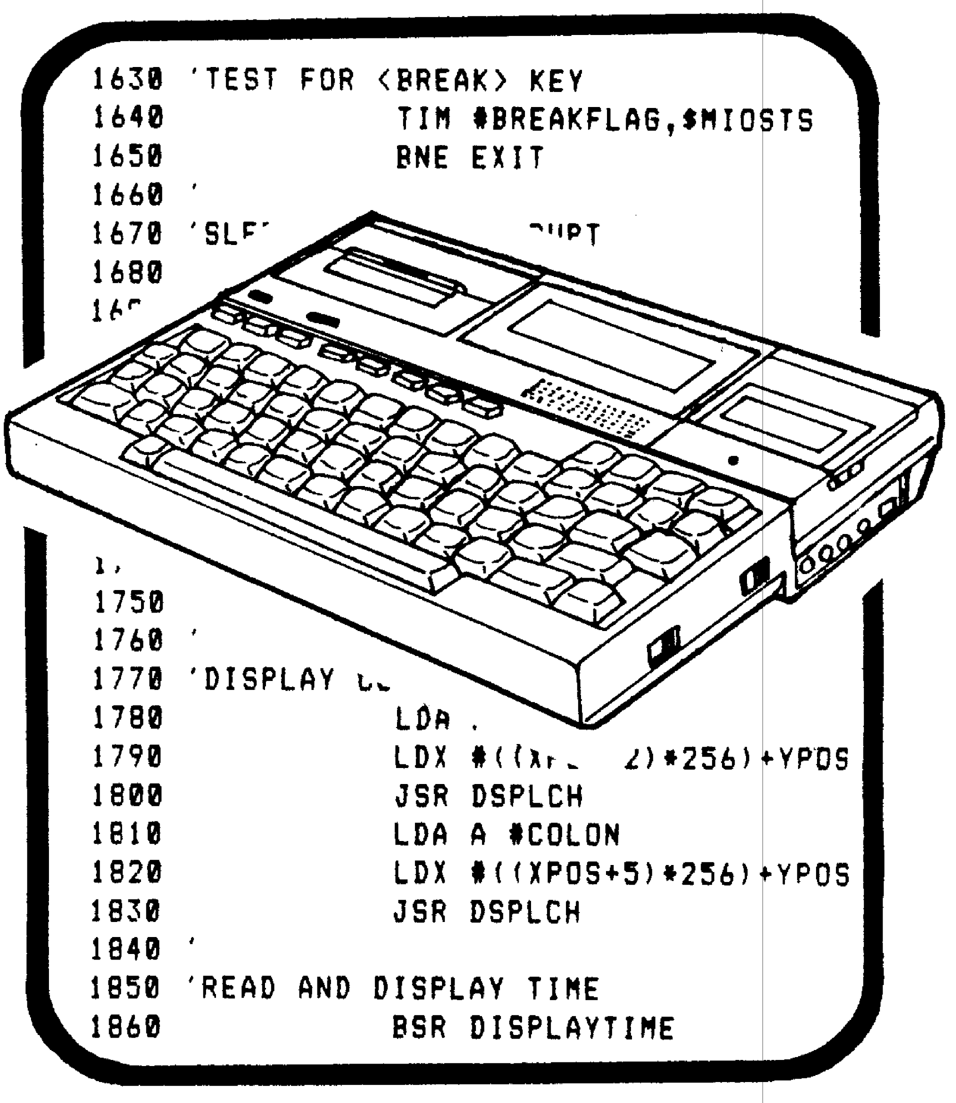
HX-20 ASSEMBLER



REFERENCE MANUAL

HX-20 ASSEMBLER

REFERENCE MANUAL

HX-20 Assembler Reference Manual:
   First Edition (C) J.M. Wald October 1985
   Second Edition (C) J.M. Wald July 1987

Assembler versions:
   1.1c and 1.1d (C) J.M. Wald August 1985
   1.2c and 1.2d (C) J.M. Wald June 1986
   2.0r (C) J.M. Wald December 1986
   2.1r (C) J.M. Wald January 1987
   2.2r (C) J.M. Wald February 1987
   2.3c, 2.3d and 2.3r (C) J.M. Wald May 1987

71 May Tree Close,
Badger Farm,
Winchester,
SO22 4JF
United Kingdom

National: Winchester (0962) 52644
International: +44 962 52644

Contents

Assembler is an extended BASIC module that allows you to include HD6301 assembly language programs as in-line code within BASIC programs. You can also use Assembler as a conventional assembler for programs written entirely in assembly language. Assembler provides the following features:

- Assembler is written in machine code and is linked into the Epson operating system. The area below MEMSET remains free for use by other machine code routines
- The ability to include assembly language directly in a BASIC program. The assembly language code is assembled when you run the BASIC program
- Program editing using the standard full screen BASIC editor
- Full implementation of both local and global labels. Global labels can be used to access routines defined in a separate source file. You can also include labels in numeric expressions
- Expressions can include any BASIC operator or function, including user defined functions
- Object code is written to memory and can be saved in a file using a SAVEM command
- Production of a listing file using any BASIC device

This manual is intended for use by programmers who are already familiar with the HD6301 family of processors.

The following manual is a useful source of additional information:

HX-20 Technical Reference Manual Epson, H8294018-0 Y202990006

A help and information service is provided by writing to:

Julian Wald,
71 May Tree Close,
Badger Farm,
Winchester S022 4JF

Tel: Winchester (0962) 52644

enclosing a stamped addressed envelope.

The assembler allows you to include an assembly language source program in a BASIC program. The *ASM* command (see Chapter 3) is used within a program to switch between assembly language and BASIC. The source program is entered using the BASIC editor and line numbering facilities. All the standard BASIC commands, statements and functions are available for use in the assembler. This means that you can use any function or operator in an expression that is used as part of an assembler statement. The source program is assembled by RUNning the BASIC program that contains the source program.

Appendix 4 contains a program that displays a clock on the LCD.

2.1 Assembler passes

In order to assemble a program, the assembler normally reads the source code in two passes. This can be performed by enclosing the source code in a *FOR..NEXT* loop, where the loop control variable is used by the *ASM* command (see Chapter 3) to specify the pass number. The pass numbers are as follows:

1 This is used for the first pass. The assembler defines labels and checks for syntax errors

2 This is used for the second pass in single file assembly. The assembler may redefine labels, generate object code or generate a listing file. The assembler always checks for errors during this pass

3 This pass replaces pass 1 for the first source file in multiple file assembly

4 This pass replaces pass 2 for the first source file in multiple file assembly

5 This pass replaces pass 1 for the second and subsequent source files in multiple file assembly

6 This pass replaces pass 2 for the second and subsequent source files in multiple file assembly

For example,

```
10 MEMSET &HB00:FOR I=1 TO 2
15  ASM I
20  ORG &HA40          ;FIRST LINE OF SOURCE CODE
        :
80      :              ;LAST LINE OF SOURCE CODE
90  ASM OFF
95 NEXT I
```

causes the assembler to read the source code twice using passes 1 and 2.

2.2 Assembler statements

An assembler statement has the following form:

[<label>] [|<assembler command>|] [<comment>]
|<instruction>|

where:

<label> is either a local or a global label (see section 2.4)

<assembler command> is one of the assembler commands described in Chapter 3

<instruction> is an HD6301 assembly language instruction described in Chapter 4 and Appendix 2

<comment> is a comment, or remark, prefixed by a semicolon (;). Note that a comment is terminated by the end of the line or by a colon (:). Comments differ from BASIC remarks in that comments can be included within multiple statement lines

The assembler allows more than one statement on a line, using a colon (:) as a separator.

2.3 Location counter

The location counter is a predefined integer variable that indicates the memory address of the current assembler instruction or command. The location counter contains the actual address of the instruction unless a non-zero offset is specified (see section 2.3.1).

The location counter is assigned an initial value in an *ORG* command (see Chapter 3) and is updated automatically by the assembler.

The value of the location counter can be used in an expression (see section 2.5.1) and is normally represented by *. The value of the location counter is often used in an expression to produce a position independent program.

For example,

STARTOFFS EQU START-*

calculates the 16-bit displacement to location START, and assigns the displacement to the constant STARTOFFS.

2.3.1 Offset

The offset is a predefined integer variable that is added to the value of the location counter to obtain the actual memory address of the current assembler instruction or command.

The offset is assigned a value explicitly in an *ORG* command (see Chapter 3), or modified implicitly by a *TBL* command.

An offset is used either to force relocation in conjunction with a relocation table (see Chapter 5), or to assemble a program at a location not normally available for user programs. For example, you may need to produce a relocatable program, or a program that is to be run in ROM or below &HA40.

If you specify a non-zero offset, the assembler uses the offset to relocate all references to locations in relocatable instructions (see Appendix 2). For example, if the offset is &H1000, the instruction

LDX #1000

will be assembled as

LDX #2000

Note that the assembler does not relocate references to locations below the <lowest address limit> or above the <highest address limit> (see *LMT* command in Chapter 3).

The value of the offset can be used in an expression (see section 2.5.1) and is normally represented by ^.

2.4 Labels

A label is a symbol, or name, that represents either an address or an item of data. A label is assigned a value either explicitly using an *EQU* command (see Chapter 3), or implicitly by labelling an assembler instruction or command. For example,

COLON EQU "\:"

assigns the ASCII value of the colon character to the label COLON.

Similarly,

```
START ORG &HA40
```

assigns the value &HA40 to the label START. Note that COLON represents an item of data, whereas START represents an address.

A label is represented by a string of up to 16 alphanumeric characters, including an underscore (_). Note that the first character must be alphabetic. In addition, a label must not start with a reserved word, nor contain a reserved word immediately following an underscore. For example,

```
START
PRNTER_SET
SWITCH1ON
```

are valid labels, whereas

```
PRINTER_OFF
_EXPRESSION
LETTER
```

are invalid labels.

Labels are often used to represent the destination of branch and jump instructions. In other words, labels perform a similar function to line numbers in *GOTO* and *GOSUB* instructions in BASIC. Labels are also used to represent memory locations, and as data constants.

2.4.1 Local labels

A local label is held as an integer variable that contains the value of the label. A label is always an integer variable even if it is not declared with a trailing percent sign (%). A local label must be declared either in the first column of a statement, or with a leading full stop (.). For example,

```
10 LOOP    LDA A #4     ;DECLARED ON FIRST COLUMN
10    .LOOP    LDA A #4 ;DECLARED WITH LEADING FULL STOP
10 LOOP%   LDA A #4     ;DECLARED ON FIRST COLUMN WITH PERCENT
```

are all valid declarations of label LOOP.

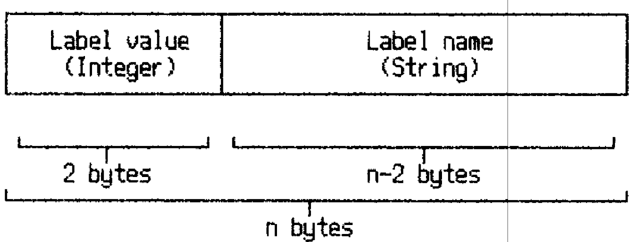
You can usually include the value of a local label in an expression in the same way as you would use any other integer variable in BASIC. However, there are restrictions on the use in expressions of local labels that contain underscore characters (see section 2.7).

2.4.2 **Global labels**

Global labels allow cross references between source files (see Chapter 6). For example, you may need to assemble a program that has more source code than the available memory in the computer. In this case, the source code must be assembled in small sections. However, local labels are erased by BASIC whenever a new file is loaded into memory, so you will need to use global labels to refer to routines and locations not defined in the current source file. Global labels are stored in a RAM file which must be set up before use (see section 5.1 in *HX-20 BASIC Reference Manual*). Note that each source file must contain the same RAM file specification.

A global label is held as a single record in the RAM file. The format of a record is a two byte integer that represents the value of the label, followed by a string that represents the name of the label.

Figure 2-1
Global label RAM file record structure



The name is truncated or space-filled to fit the record size specified in the *DEFFIL* statement. Note that you should specify a record length of at least three bytes and not more than 18 bytes. The record length that you select will affect the number of significant characters in global label names. For example, a record length of 10 bytes allows global label names where only the first eight characters are significant. The size of the RAM file is the number of global labels multiplied by the record size.

A global label is declared with a leading exclamation mark (!). For example,

```
!CLOCK    ORG &HA40
          !HERE EQU *
```

are both valid declarations of global labels. Note that a reference in a source file to a global label must always include the leading exclamation mark, for example

```
    JSR !CLOCK
```

Global labels can be included in expressions (see section 2.5.1).

2.4.3 Symbol table

The symbol table is the set of all labels currently defined. You can obtain a listing of the symbol table using the *SYM* command (see Chapter 3).

2.5 Numeric expressions

The assembler accepts any numeric expression that conforms to the following syntax:

```
|<operand>                                |
|<monadic operator> <operand>             |
|<operand> <dyadic operator> <operand>    |
|(<operand>)                              |
```

where:

<operand> is one of the operands described in section 2.5.1

<monadic operator> is one of the monadic operators described in section 2.5.2

<dyadic operator> is one of the dyadic operators described in section 2.5.3

2.5.1 Operands

The assembler allows you to use an extensive range of operands. In addition, you can treat an entire BASIC expression as a single operand.

The assembler allows you to use the following operands:

- Numeric constants. The following types of numeric constants are allowed:

  - In the default input base (see the *RAD* command in Chpater 3), and starting with a numeric digit

  - Hexadecimal, preceded by **&H** or by **$**

  - Decimal, preceded by **&D**

  - Octal, preceded by **&** or **&O**

  - Binary, preceded by **&B** or **%**

  - A one or two character string enclosed in double quotation marks, for example **"AB"**. Note that either character in the string may be a metacharacter (see section 2.6)

- Labels. Note that references to udefined labels are currently assigned a value of zero, and are flagged by a U in the flag field of the listing

- The location counter, represented by *

- The offset, represented by ^

- The memory address, that is the location counter plus the offset, represented by @

- The count of the number of global labels defined in the program, represented by !

- A BASIC expression enclosed in parenthesis. A BASIC expression is any numeric expression accepted by the standard HX-20 BASIC interpreter. Note that a BASIC expression is evaluated by the HX-20 BASIC interpreter and not by the assembler. For example, **(614400#\256)** returns the value 2400

- A numeric expression

2.5.2 **Monadic operators**

A monadic operator is an operator that takes a single value as an argument and returns a single value as a result.

The asembler accepts the following monadic operators:

+ Monadic identity

- Negation

EXT Sign extension. This operator extends an 8-bit unsigned number to a 16-bit signed number

BYT Sign reduction. This operator converts a signed 16-bit number to an unsigned 8-bit number

NOT Logical inversion

2.5.3 **Dyadic operators**

A dyadic operator is an operator that takes a pair of values as arguments and returns a single value as a result.

The assembler accepts the following dyadic operators:

+	16-bit signed addition
-	16-bit signed subtraction
*****	16-bit signed multiplication
/ ****	16-bit signed integer division (truncates result towards zero)
MOD	16-bit signed remainder after division
AND	Logical AND
OR	Logical OR
EQU	Logical XOR
IMP	Logical implication
< **<=** **>** **>=** **=** **<>**	Signed comparisons. These return a value of -1 if true and 0 if false
LO **LS** **HI** **HS**	Unsigned comparisons. These return a value of -1 if true and 0 if false
ASR **LSR**	Arithmetic and Logical shift right. The left argument is shifted right by the number of places specified by the magnitude of the right argument. A negative right argument produces a shift left
ROR	Rotate right. The low order byte of the left argument is rotated right through the high order byte by the number of places specified by the magnitude of the right argument. A negative right argument produces a rotate left

2.6 Strings

A string is a collection of one or more characters enclosed in double quotation marks ("). Note that any character in the string may be a special character known as a metacharacter.

Metacharacters allow you to include character codes that fall outside the range for alphanumeric characters. For example, you might find it useful to be able to include control character codes in your program. The metacharacters usually consist of a special symbol followed by an alphanumeric character. The assembler interprets both the special character and the alphanumeric character as one character.

The five types of metacharacters are:

' A single apostrophe (') represents the double quotation mark character ("). The double quotation mark is normally not allowed in a string as it is used to delimit the start and end of the string

^<char> A character preceded by a caret (^) represents the character with an ASCII code 64 less than the ASCII code of the specified character. For example,

"^M^J"

is a string containing the control codes for carriage return (ASCII code 13) and line feed (ASCII code 10)

%<char> A character preceded by a percent sign (%) represents the character with an ASCII code 64 greater than the ASCII code of the specified character. For example,

"%0"

is a string containing the code for a lower case letter P (p)

|<char> A character preceded by a vertical bar (|) represents the character with an ASCII code 128 greater than the ASCII code of the specified character. In other words the vertical bar sets the most significant bit in the specified character code to one. For example,

"|J"

is a string containing a single letter J that has the most significant bit set to one

\<char> The reverse solidus (\) enables you to include a symbol that normally has a special meaning. The reverse solidus is used to include the metacharacter symbols, the colon, and the HX-20 graphic symbols in a string. For example,

"This string contains a '\^' character"

is a string containing:

This string contains a "^" character

Note that you can change the double quotation marks into apostrophes by preceding each apostrophe by a reverse solidus as follows:

"This string contains a \'\^\' character"

2.7 Accessing Assembler operands from BASIC

To include assembler operands, for example global labels, in a BASIC expression you must enclose the operand in parenthesis. For example,

(*)
(!CLOCK)

represent the value of the location counter and the value of the global label !CLOCK. Note that you must precede a string by a plus sign (+) and a local label by a full stop (.) if you enclose either type of operand in parenthesis.

For example,

(+"A")
(.START)

represent the values of the string "A" and the local label START respectively.

Note that you can refer to a local label as an integer variable in an expression. However, you cannot refer to a local label that contains an underscore character except by enclosing the name, prefixed by a full stop, in parenthesis.

2.8 Object code

The assembler places the object code in memory, if you have enabled the production of object code. Object code production is enabled in passes 2, 4 and 6 using the *OBJ* command (see Chapter 3). You can disable the production of object code using the *NOB* command.

You must use the MEMSET command (see Chapter 3 and section 5.3.1 in *HX-20 BASIC Reference Manual*) to reserve enough memory for the object code produced by the assembler.

You might find it useful to be able to reserve exactly enough memory, and no more, for the object code. To do this the following routine can be used:

```
100 DEFINT A-Z:I=1:GOSUB 200:MEMSET (0)
110 DEFINT A-Z:FOR I=1 TO 2:GOSUB 200:NEXT I
120 EXEC START          'EXECUTE PROGRAM
130 END
140 '-----------------------------------------
200       ASM I
210 START ORG &HA40     ;START OF PROGRAM
220       OBJ           ;ENABLE OBJECT CODE
           :            ;LINES OF SOURCE CODE
980       ASM OFF
990 RETURN
```

In this routine line 100 partially assembles the code using pass 1. This ensures that the address of the last location used plus one is known. The MEMSET command is given this value as the first available location for BASIC, thus ensuring that no memory is wasted. However, MEMSET has the side effect of erasing all variables and declarations, so the assembler is re-run using both pass 1 and pass 2. Note that you can use a similar routine to reserve memory in multiple file assembly (see Chapter 6).

2.9 Listing file

The assembler allows you to request a listing file when the program is assembled. A listing file is produced only during passes 2, 4 and 6. The listing file contains a formatted copy of the source program and object code. You can specify that the listing file is divided into pages and specify a page title. The page title includes the date and time of assembly.

The format of each page is given in the following sections.

2.9.1 Page heading

The assembler prints a heading at the top of every page of the listing file. The format of a page heading is as follows:

<Page-number> <Date> <Time> <Title>

where:

<Page-number> is the number of the current page. Note that the page number is given as a five digit number with leading zeroes, starting with page 00001. The page number is reset to 00001 at the start of pass 2, but retains its current value at the start of pass 4 and 6

<Date> is the current date and is obtained from the HX-20 internal clock. The format of the date is *mm/dd/yy* where *mm* is the month, *dd* is the day, and *yy* is the year in the century

<Time> is the current time and is obtained from the HX-20 internal clock. The format of the time is *hh/mm/ss* where *hh* is the hour in the 24 hour clock, *mm* is the minute, and *ss* is the second

<Title> is either blank or contains the user-specifed title. The title is specified by the *TTL* command (see Chapter 3)

The heading is separated from the remainder of the page by a blank line. Note that the page heading is truncated if the page width specified in the *FMT* command (see Chapter 3) is too small.

2.9.2 Page body

The format of each line in the page body is as follows:

<Line-number> <Flag> <Address> <Object-code> <Label> <Source-statement>

where:

<Line-number> is the BASIC line number of the current statement, given as a five digit number with leading zeroes

<Flag> is either blank or contains one of the following:

U The line contains a reference to an undefined label

J A JMP or JSR instruction is within the range for a branch instruction, so a branch instruction can be used instead

B The range of a branch instruction is within 16 bytes of the maximum allowed. Note that a J flag may change to a B flag where appropriate

D The label has been defined more than once

<Address> is either blank or contains the hexadecimal value of the location counter (see section 2.3). However, if the current instruction is an *EQU* command (see Chapter 3), <Address> contains the hexadecimal value assigned to the label. Note that if the address is relocated, the address is flagged by a plus sign (+)

<Object-code> is either blank or contains the hexadecimal representation of the object code. Note that any object code that is relocated is listed in its unrelocated form and is flagged by a plus sign (+)

<Label> is either blank or contains the name of the label defined in the current source statement.

<Source-statement> is a copy of the current source statement, excluding any label defined in the current source statement.

Note that a line is truncated if the page width specified in a *FMT* command (see Chapter 3) is too small.

2.10 **Memory locations used by Assembler**

The assembler uses locations &H60 to &H6F, &H2C6 to &H2CF and the area above RAMADR as a temporary work area. If you need to use any of these areas, you must ensure that you save any important contents before you use the Assembler.

This chapter provides you with full details of the assembler commands available. The commands are described in alphabetic order and are in the same format as Chapter 3 and Chapter 4 of *HX-20 BASIC Reference Manual*.

ASM

FORMAT **ASM** ¦<numeric expression>¦
¦CONT¦
¦OFF¦

PURPOSE To start, continue or terminate assembly

EXAMPLE **ASM I**

REMARKS The ASM command is used to switch between BASIC and the assembler.

To enter the assembler from BASIC use the ASM command followed by a numeric expression to specify the current pass. The pass number is usually specified as the control variable in a *FOR..NEXT* loop which encloses the code to be assembled.

The ASM OFF command is used to terminate assembly and return to BASIC.

The ASM CONT command is used to re-enter the assembler after an ASM OFF command.

See also OBJ, LST and section 2.1 Assembler passes

EQU

FORMAT **<label> EQU <numeric expression>**

PURPOSE To assign a specified value to a label

EXAMPLE **SNSCOM EQU &HFF19**

REMARKS The EQU command assigns the value specified by <numeric expression> to <label>. Note that a MO error occurs if no label is specified. Note also that a DD error occurs if a label is re-defined in RDF N mode.

See also RDF, ORG and section 2.4 Labels

FCB

FORMAT **FCB |<numeric expression>|[,|<numeric expression>|....]**
|<string> | |<string> |

PURPOSE To fill contiguous bytes with the specified data

EXAMPLE
```
FCB 10,23,5,8
FCB "A Message",13,10,0
FCB "Another Message^M^J^@"
```

REMARKS The FCB command fills contiguous bytes with the given data. The data can include either numbers or strings, or any combination of the two types of data.

Numeric expressions must evaluate to zero or a positive number less than 256, otherwise a FC error occurs.

Strings must be enclosed in double quotation marks (") and may include metacharacters. For example, "^M^J^@" is a string containing a carriage return followed by a line feed and null character.

See also FDB, RMB, RDB, section 2.5 Numeric expressions and section 2.6 Strings

FDB

FORMAT **FDB <numeric expression>[,<numeric expression>....]**

PURPOSE To fill contiguous double bytes with the specified data

EXAMPLE
```
FDB 10,23,5,8
FDB SNSCOM,*-START,FETCH-HERE  ;LABEL VALUES
```

REMARKS The FDB command fills contiguous double bytes with the given data. This command is particularly useful for setting up a table of addresses, as in the second example above.

See also FCB, RMB, RDB and section 2.5 Numeric expressions

FMT

FORMAT **FMT [<page length>][,[<page width>][,[<start col>][,[<header string>][,<trailer string>]]]**

PURPOSE To specify the page size

EXAMPLE

```
FMT 60,80
FMT ,,,"^[M","^L"
FMT 255,32,1
```

REMARKS The page length is set to <page length>, or to the default if none is given. The default page length is 60 lines or the last value given. Note that a page length of 255 lines is interpreted as an infinite page length.

The page width is set to <page width>, or to the default if none is given. The default page width is 80 characters or the last value given. Any value greater than 128 is truncated to 128. If the device width is smaller than the line width, the line is truncated to fit the device. The value of <page width> must be at least 5.

The start column is set to <start col>, or to the default if none is given. The default start column is 1 or the last value given. Any value greater than 128 is truncated to 128. If the page width plus the start column exceeds 128, the page width is truncated.

The header string sent at the start of every page is set to <header string>, or to the default value if none is given. The default header string is "" or the last value given.

The trailer string sent at the end of every page is set to <trailer string>, or to the default value if none is given. The default trailer string is "" or the last value given.

See also LST

LMT

FORMAT **LMT [<lowest address limit>][,<highest address limit>]**

PURPOSE To specify the lowest and highest addresses used in relocation

EXAMPLE **LMT &HA40,@-1**

REMARKS The LMT command is used to specify the lowest and highest addresses to be relocated when using a non-zero offset in an *ORG* command. At the start of a program, after an *ASM* command using pass 1 or 2, the lowest address limit is &HA40, and the highest address limit is &H7FFF.

If you specify a new <lowest address limit> but no new <highest address limit>, the current value of <highest address limit> is retained. Similarly, if you specify a new <highest address limit> but no new <lowest address limit> the current value of <lowest address limit> is retained.

See also ORG, TBL, Section 2.3 Location Counter and Chapter 5 Relocatable programs

LST

FORMAT **LST [¦[#]<BASIC file number>¦]**
¦<device descriptor> ¦

PURPOSE To generate a listing file during pass 2, 4 and 6

EXAMPLE **LST "68N1B"**

REMARKS The LST command enables the production of a listing file.

The listing file is sent to the specified device or BASIC file, or the default if none is given. The default is the RS232 port or the last file or device specified. The file descriptor is either "I" for the microprinter, or "BLPSC" for the RS232 port. BLPSC are the parameters used in the BASIC OPEN "COM0:" statement

See also FMT, NOL, PAG, TTL and section 2.9 Listing file. Refer also to the OPEN and CLOSE statements in *HX-20 BASIC Reference Manual*, and to section 5.2 *Sequential files* in the same manual.

MEM

FORMAT **MEM <lowest memory limit>[,<highest memory limit>]**

PURPOSE To specify the range of memory allowed for object code

EXAMPLE **MEM $800,$7FFF**

REMARKS The MEM command specifies the lowest and highest memory addresses allowed for object code, changing the previosly set values or defaults

At the start of assembly, after an *ASM* command using pass 1 or 2 the lowest memory limit is &HA40 and the highest address limit is MEMSET-1.

If you specify a new <lowest memory limit> but no <highest memory limit> the current value of <highest memory limit> is retined. Similarly, if you specify a <highest memory limit> but no <lowest memory limit> the current value of <lowest memory limit> is retained.

See also LMT

NOB

FORMAT **NOB**

PURPOSE To disable the production of object code during pass 2, 4 or 6

EXAMPLE **NOB**

REMARKS The NOB command disables the production of object code during pass 2, 4 or 6. This command is particularly useful if you only want to assemble part of a file. Note that a NOB command is not necessary in pass 2, 4 or 6 if no *OBJ* command has been given, as the assembler assumes a NOB command and does not produce object code.

See also OBJ

NOL

FORMAT **NOL**

PURPOSE To disable the production of the listing file during pass 2, 4 or 6

EXAMPLE **NOL**

REMARKS The NOL command disables the production of the listing file enabled by a *LST* command. This command is particularly useful if you only require part of the file to be printed out, for example when you are using a routine for which you already have a listing. Note that a NOL command is not necessary if no *LST* command has been given, as the assembler assumes a NOL command and does not produce a listing file. To re-enable the production of the listing file use the *LST* command.

See also LST

OBJ

FORMAT **OBJ**

PURPOSE To enable the production of object code during pass 2, 4 or 6

EXAMPLE **OBJ**

REMARKS The OBJ command enables the production of object code during pass 2, 4 or 6 of the assembler. The object code produced is stored in memory at the address calculated by adding the value of the location counter to the offset value given in an *ORG* command. Note that if you specify a non-zero offset, the assembler automatically relocates the object code. Any relocated references in the object code are flagged by a + sign in the listing file.

See also NOB, ORG, LST, section 2.8 Object code and section 2.9 Listing file

ORG

FORMAT **ORG [<origin>][,<offset>]**

PURPOSE To specify the value of the location counter and the offset

EXAMPLE **ORG &HA40**

REMARKS At the start of the program, after an *ASM* command using pass 1 or 2, both the location counter and the offset are zero. The ORG command allows you to specify a new value for the location counter (<origin>) and offset (<offset>). You must specify <origin> and <offset> as numeric expressions.

If you specify a new <origin> but no new <offset>, the current value of the offset is retained. Similarly, if you specify a new <offset> but no new <origin>, the current value of the location counter is retained.

Note that if you define a label in the same statement as an ORG command, the label is assigned the new value of the location counter and not the location of the ORG command. In other words, the label is assigned a value after the ORG command is executed.

See also Sections 2.3 Location counter, 2.3.1 Offset and 2.4 Labels

PAG

FORMAT **PAG**

PURPOSE To start a new page in the listing file

EXAMPLE **PAG**

REMARKS The PAG command forces a new page in the listing file. The command takes effect at the point at which the command is given. Note that this command is ignored if an infinite page length has been selected using the *LST* command.

See also LST and TTL

RAD

FORMAT **RAD ¦<input base>[,<output base>]¦**
¦,<output base>   ¦

PURPOSE To specify the input and output numeric base

EXAMPLE **RAD &D10**

REMARKS The RAD command sets the default input and output bases. The defaults are initially &D10 and &H10 respectively. The output base is used only in the HD6301 Debugger.

See also Section 2.5.1 Operands

RDB

FORMAT **RDB <numeric expression>**

PURPOSE To reserve a specified number of double bytes of storage

EXAMPLE **RDB 250**

REMARKS The RDB command advances the location counter by the number of double bytes specified by <numeric expression>. Note that the reserved space is not initialised to any particular value.

See also RMB, FCB and FDB

RDF

FORMAT **RDF |Y|**
|N|

PURPOSE To enable or disable the re-definition of labels

EXAMPLE **RDF N**

REMARKS The RDF command allows you to re-define the values of labels within a program without causing a DD error. The Y parameter enables the re-definition of labels, and the N parameter disables the re-definition of labels.

The example below shows a situation in which the RDF Y command is useful:

```
          RDF Y                ;ENABLE RE-DEFINITION
SIZE      EQU TBLEND-TBLSTART  ;TABLE SIZE
          RDF N                ;DISABLE RE-DEFINITION
           :
TBLSTART  RMB 100              ;RESERVE TABLE SPACE
TBLEND    EQU *                ;END OF TABLE + 1
```

Normally the assembler assigns values to labels during pass 1, 3 or 5. The RDF Y command forces the assembler to re-calculate the value of SIZE during pass 2, 4 or 6 when the values of TBLSTART and TBLEND are known. The RDF N command is used to ensure that no other labels are re-defined. Note that you must use RDF Y and RDF N commands in pairs.

The assembler assumes a RDF N command if no RDF command is given.

See also EQU and section 2.4 Labels

RMB

FORMAT **RMB <numeric expression>**

PURPOSE To reserve a specified number of bytes of storage

EXAMPLE **RMB 500**

REMARKS The RMB command advances the location counter by the number of bytes specified by <numeric expression>. Note that the reserved space is not initialised to any particular value.

See also RDB, FCB and FDB

SYM

FORMAT **SYM [|L|]**
|G|

PURPOSE To produce a symbol table listing

EXAMPLE **SYM G**

REMARKS The SYM command produces a symbol table listing in the listing file at the point at which the SYM command is given. The format of the table is one label per line together with its current value (address) in hexadecimal. The *L* parameter produces a symbol table for local labels only, and the *G* parameter for global labels only. If no parameter is given, the symbol table contains both local and global labels. Note that the labels are listed in alphabetical order of initial letter only; global labels first, followed by local labels.

See also Section 2.4 Labels and section 2.4.3 Symbol table

TBL

FORMAT **TBL [<numeric expression>]**

PURPOSE To enable production of a relocation table

EXAMPLE **TBL &H6000**

REMARKS The TBL command is used to enable production of a link table. The link table is used to relocate programs. The object code is assembled with an origin of <numeric expression>, regardless of the value of <origin> and <offset>. The object code is placed in memory at <origin>+<offset>. The TBL command has no effect if <offset> is zero. Only references between <lowest address limit> and <highest address limit> are relocated. The value of <offset> is altered by the TBL command to <old offset>+(<highest address limit>-<lowest address limit>)\8+1. This alteration is made to enable the correct amount of memory to be reserved for the link table. The <origin> is NOT affected.

See also ORG, LMT, TBL, and Chapter 5 Relocatable programs

TTL

FORMAT **TTL <string>**

PURPOSE To set up a title and start a new page on the listing file

EXAMPLE **TTL "Simple Clock Program"**

REMARKS The TTL command sets up the title which is printed at the top of every subsequent page on the listing file, if a listing file has been opened. The title will also include the date and time of the listing. The time given will be the time that the title is set up, and not the time at which the file is actually listed. Thus, if you set up several titles within one listing file, each title will contain a different time.

See also PAG, LST and section 2.6 Strings

The Hitachi HD6301 is an eight-bit processor based on the Motorola MC6800. The HD6301 can access a maximum of 65536 eight-bit memory locations. The Epson HX-20 contains two HD6301 processors in a master and slave relationship. However, the user can program only the master processor. The slave processor is reserved for input-output operations.

4.1 HD6301 Instruction set

There are 81 instructions in HD6301 assembly language (see Appendix 2). Each instruction is represented inside the computer as a unique eight bit binary number known as the op-code. However, a program written as a series of op-codes can be confusing and so each instruction is also represented by a mnemonic. Thus the *no operation* instruction is written as *NOP* and has the op-code &H01. The purpose of an assembler is to convert a program written using mnemonics into the internal op-code representation.

Appendix 2 provides details of the HD6301 instruction set divided into the following six categories:

- Data movement. The data movement instructions move data between two registers, or between a register and memory. This section includes instructions to set or clear flags

- Arithmetic. The arithmetic instructions perform arithmetic operations. This section includes the arithmetic shift instructions

- Logical. The logical instructions perform the logical operations AND, OR and exclusive OR. This section includes the logical shift and rotate instructions

- Comparison and test. The comparison and test instructions compare the contents of a register with another register or memory. This section includes instructions to test the value of individual bits in a register or memory location

- Branch. The branch instructions transfer program control to another part of the program

- Program transfer and miscellaneous. This section contains instructions to transfer program control and control interrupts

4.1.1 HD6301 addressing modes

The location of data used in an instruction is specified using one of the following six addressing modes:

- Immediate. In this mode the data is contained as a constant in the instruction. The immediate addressing mode is represented by a numeric expression preceeded by a hash sign (#), or a pound sign (£) on the English keyboard. For example,

  LDA A #10

  loads accumulator A with the constant value 10

- Direct. In this mode the data is contained in a memory location with an address in the range 0 to 255 (&H0 to &HFF). The direct addressing mode is represented by a numeric expression optionally preceeded by a dollar sign ($). For example,

  STA A &H80

  stores the contents of accumulator A in location &H80

- Indexed. In this mode the data is contained in a memory location with an address specified as a constant positive offset from the contents of the index register. The indexed addressing mode is represented by a numeric expression preceeded by a letter X and an optional plus sign (+). For example,

  ```
  LDX #&H1000
  LDA A X+4
  LDA B X0
  ```

  loads accumulator A with the contents of location &H1004 and accumulator B with the contents of location &H1000

- Extended. In this mode the data is contained in a memory location. The extended addressing mode is represented by a numeric expression. For example,

  JMP &HDFFD

  transfers program control to location &HDFFD. Note that if the extended address is below &H100 the assembler will automatically use direct addressing where possible

- Implied. In this mode the data is contained in a register or memory location implicitly specified by the instruction. For example,

  CLR A
  INC B

  clears accumulator A and increments accumulator B

- Relative. This mode is used to specify the destination of branch instructions. The data is the offset from the address of the branch instruction to the destination address. The assembler does not use relative addressing, but expects an extended mode address to be supplied as the destination of a branch. The assembler converts the supplied extended mode address into the relative offset required by the HD6301. The extended mode address is usually a label. For example,

  BRA START

  transfers program control to label START

A relocatable program is a program that makes no references to any fixed memory locations within the program area, and can therefore be run anywhere in memory. Relocatable programs are particuarly suitable for use on the HX-20 as all application files must be relocatable.

One problem caused by relocatable programs is that all addresses must be ultimately be specified as offsets from a particular location. This location must be calculated. This is difficult as the HD6301 does not allow you to read the contents of the program counter to obtain the current location of the program.
One method of finding the current location of a table is given below:

```
        BSR LABEL1
TABLE    :
         :
LABEL1  PUL X or PUL A
                 PUL B
        ;X or D contains absolute address of TABLE
```

Note that a pseudo subroutine call is used to push the address of TABLE on the stack. This address is then later pulled off the stack. The program continues to run from LABEL which is the first address after the table.

The two locations LABEL and TABLE could be the same location, as follows:

```
        BSR LABEL1
LABEL1  PUL X or PUL A
                 PUL B
        ;X or D contains absolute address of LABEL1
```

In this case the routine is used merely to find the address of LABEL1.

Assembler provides an alternative system to enable you to write relocatable programs. Assembler allows you to write a relocatable program as if it is an absolute program, and use a link table to relocate absolute references at run time. The assembler can automatically produce a link table using the *LMT* and *TBL* commands (see Chapter 3). Note that the format of the link table produce is compatible with that required by the LOAD OPEN command in the Epson TF-20 Disk drive.

A suitable routine to perform the relocation is listed below:

```
ORIGIN    EQU $6000               ;OBJECT CODE ORIGIN
          ORG ORIGIN,$A40-*       ;PLACE OBJECT CODE @ $A40
;
PGMSTRT   EQU $60                 ;ADDR OF START OF PROGRAM
PGMEND    EQU $62                 ;ADDR OF END OF PROGRAM
LNKSTRT   EQU $64                 ;ADDR OF START OF LINK TABLE
;
TEMP1     EQU $68                 ;POINTER TO PROGRAM
TEMP2     EQU $6A                 ;POINTER TO LINK TABLE
;
START     LDS +$4AF               ;INITIALISE STACK
          BSR HERE
HERE      PUL A
          PUL B
          SUB D #(HERE-START)     ;D = START
          STD PGMSTRT
          ADD D #(LNKTBL-PGMSTRT) ;D = LNKTBL
          STD PGMEND
          STD LNKSTRT
          STD TEMP2
          LDX PGMSTRT
.LNKLOOP  STX TEMP1               ;SAVE PROGRAM POINTER
          PSH X
          LDA B #8                ;BIT COUNTER
          LDX TEMP2
          LDA A X+0               ;GET LINK BYTE
          INX
          STX TEMP2               ;UPDATE LINK TABLE POINTER
          PUL X                   ;RESTORE PROGRAM POINTER
.BYTLOOP  CPX PGMEND
          BEQ EXIT                ;TERMINATE LOOP IF ALL DONE
          ROL A                   ;GET LINK BIT INTO (C)
          BCC NONREL              ;IF 0 NO RELOCATION
          PSH B
          PSH A                   ;SAVE LINK BYTE + BIT COUNT
          LDD X+0                 ;GET ADDRESS TO RELOCATE
          SUB D #ORIGIN
          ADD D PGMSTRT
          STD X+0                 ;SAVE RELOCATED ADDRESS
          PUL A
          PUL B                   ;RESTORE LINK BYTE + BIT COUNT
NONREL    INX                     ;NEXT PROGRAM BYTE
          DEC B
          BNE BYTLOOP             ;8 TIMES FOR 1 LINK BYTE
          BRA LNKLOOP             ;NEXT LINK BYTE
EXIT       :                      ;EXECUTE PROGRAM
           :
           :
LNKTBL    LMT START,LNKTBL-1      ;RELOCATION LIMITS
          TBL ORIGIN              ;ENABLE LINK TABLE
                                  ;MEMSET MUST BE SET TO (@)
```

You can assemble a program in several sections if there is insufficient memory in the computer for the complete source code. The source code is divided into several small files, and a header file is written. The header file contains initialisation information and is RUN to assemble the source code. Appendix 5 contains a multiple source file program that continuously displays a clock on the LCD.

Multiple source file assembly operates as follows:

The header file initialises a pass flag to 1. The pass flag indicates the current assembler pass required. The header file also sets up a RAM file and opens a file for the listing file. The header file then RUNs the first source file.

The first source file defines its RAM file to be the same as that set up by the header file. The first file then examines the pass flag. If the pass flag is 1, the source code in the first file is assembled using pass 1 only. Otherwise, the source code in the first file is assembled using pass 3 followed by pass 4. The first source file then RUNs the second source file.

The second source file defines its RAM file to be the same as that set up by the header file. The second file then examines the pass flag. If the pass flag is 1, the source code in the second file is assembled using pass 5 only. Otherwise, the source code in the second file is assembled using pass 5 followed by pass 6. The second file then RUNs the next source file.

The last source file is similar to the second source file. However, once it has assembled its source code, it increments the pass number. If the pass number is now 2, the last source file RUNs the first source file. Otherwise the last source file closes the listing file and assembly is completed.

Note that the third and subsequent source files, if any, are handled in the same way as the second source file. Note also that the second source file may be the last source file.

The following sections contain details of the format of each file required in multiple file assembly.

6.1 Header file

The header file consists of the following program:

```
10 CLEAR 200,8*255+4        'MAX 255 GLOBAL LABELS
20 DEFFIL 2,0:PUT% 0,1      'PASS NUMBER
30 OPEN "O",1,"COM0:"       'LISTING FILE CHANNEL
40 WIDTH "COM0:",80         'LISTING FILE WIDTH
50 T=TAPCNT:PUT% 1,T        'TAPE COUNT VALUE
60 RUN "FILE1.SRC",R
```

6.2 The first source file

The first source file consists of the following program:

```
10 DEFINT A-Z
20 DEFFIL 2,0:GET% 0,P      'GET PASS NUMBER
30 DEFFIL 8,4               'SET UP RAM FILE
40 IF P=1 THEN I=1:GOSUB 100
50 IF P=2 THEN FOR I=3 TO 4:GOSUB 100:NEXT I
60 RUN "FILE2.SRC",R
70 '----------------------------------------------------
100       ASM I
110       ORG &HA40
120       OBJ
130 !START                  ;START OF PROGRAM
           :
           :
           :                ;Mth SOURCE CODE LINE
980       ASM OFF
990 RETURN
```

6.3 The second source file

The second source file consists of the following program:

```
10 DEFINT A-Z
20 DEFFIL 2,0:GET% 0,P      'GET PASS NUMBER
30 DEFFIL 8,4               'SET UP RAM FILE
40 IF P=1 THEN I=5:GOSUB 100
50 IF P=2 THEN FOR I=5 TO 6:GOSUB 100:NEXT I
60 RUN "FILE3.SRC",R
70 '----------------------------------------------------
100       ASM I
           :                ;M+1th SOURCE CODE LINE
           :
           :                ;Nth SOURCE CODE LINE
980       ASM OFF
990 RETURN
```

6.4 The final source file

The final source file consists of the following program:

```
10 DEFINT A-Z
15 DEFFIL 2,0:GET% 0,P     'GET PASS NUMBER
20 DEFFIL 8,4              'SET UP RAM FILE
15 IF P=1 THEN 50
30 FOR I=5 TO 6:GOSUB 100:NEXT I:CLOSE
35 EXEC (!START)           'EXECUTE OBJECT CODE
40 GOTO 80
50 I=5:GOSUB 100
55 DEFFIL 2,0
60 PUT% 0,2                'SET UP PASS 2
65 GET% 1,T:WIND T         'WIND TO START
70 MEMSET LIMIT            'ALLOCATE OBJECT CODE SPACE
75 RUN "FILE1.SRC",R
80 END
90 '------------------------------------------------
100       ASM I
           :               ;N+1th SOURCE CODE LINE
           :
           :               ;LAST SOURCE CODE LINE
970 LIMIT EQU *            ;FIRST UNUSED MEMORY LOCATION
980       ASM OFF
990 RETURN
```

A1.1 Loading Assembler from ROM cartridge or microcassette

To load Assembler from a ROM cartridge or microcassette perform the following steps:

1 Switch off the HX20 and connect either the ROM cartridge or microcassette drive to the HX-20.

2 Switch the HX20 on, and save any important machine code programs or data held below MEMSET as these are destroyed when you link Assembler

3 Enter BASIC

4 If you are loading from microcassette, place the program cassette in the microcassette drive and wind the cassette to the start of the tape

5 Type

MEMSET &H2B00:LOADM "",,R

and press the RETURN key

This loads Assembler and runs the linker routine. Assembler then returns to the main menu and re-enters BASIC. The linker moves Assembler to a protected area of memory above BASIC, and resets MEMSET to the base address of &H0A40. The linker is destroyed and is not copied to the protected area. Note that you can now change MEMSET so that you can use the area of memory below MEMSET for your own programs.

> **WARNING:** The linker unlinks all ROM application programs except BASIC and the monitor. The reason for this is that some ROM programs are not implemented in a manner consistent with the use of application files as described in *HX-20 Technical Reference Manual* (Section 2, sub-section 18.4)

6 If you are using the microcassette drive, rewind the program tape and remove it from the microcassette drive. If you are using the ROM cartridge, you may now switch off the HX-20, and replace the ROM cartridge with the microcassette drive

A1.1.1 **Making a back-up copy on microcassette**

To make a back-up copy of Assembler you must use the copy utility. This utility is provided as part of the linker routine and is designed to be run before you run the linker.

To load the copy utility perform the following steps:

1. Switch off the HX20 and connect the microcassette drive to the HX20

2. Switch the HX20 on and save any important machine code programs or data held below MEMSET, as these are destroyed when you load the copy utility

3. Enter BASIC

4. Place the program cassette in the microcassette drive and wind the cassette to the start of the tape

5. Type

   MEMSET &H2B00:LOADM:EXEC &HA40

   and press the RETURN key

This loads and runs the copy utility. The copy utility displays the following:

Copy utility V1.0
Device (M/C) ?
File = ASSEMBLR.REL
Size = 04780 Bytes

Press either **M** to record on the internal microcassette, or **C** to record on an external cassette recorder. The program is then saved on the specified cassette and the copy utility returns to BASIC. You can now link the program into the system for use by typing the following:

EXEC &HB40

and pressing the RETURN key.

A1.2 Loading Assembler from disk

Assembler is provided on a master disk. However, before you can run Assembler, you must create an Assembler system disk containing both Assembler and Disk BASIC. To create an Assembler system disk perform the following steps:

1. Enter Disk BASIC (see section 4.2 of *HX-20 Disk BASIC Reference Manual*)

2 Place the Assembler master disk in drive A

3 Type

RUN "SYSGEN.BAS"

and press the RETURN key. Follow the instructions provided by the program to create an Assembler system disk. Note that you will require a blank disk that is not write protected, a Disk BASIC system disk and the supplied Assembler system disk label.

4 The SYSGEN.BAS program converts the blank disk into an Assembler system disk and automatically re-boots BASIC. On entry to BASIC the Assembler is automatically loaded along with Disk BASIC, and displays a copyright message to indicate successful loading. The BASIC program "SYSGEN.BAS" is automatically cleared from memory. Note that if there is insufficient memory available you might find that either Disk BASIC, or Disk BASIC and Assembler, will not load (see section 4.2 in *HX-20 Disk BASIC Reference Manual*).

You can now use the Assembler system disk as a replacement for the standard Epson system disk.

A1.2.1 Making a back-up copy on disk

Back-up copies of the new Assembler system disk can be made using either the *SYSGEN* command or the volume copy facility (see section 3.3 (2) in *HX-20 Disk BASIC Reference Manual* and section 4.6 in the same manual).

Back-up copies of the supplied Assembler master disk can only be made using the volume copy facility.

A1.3 Installing Assembler on ROM

To install Assembler on ROM you should perform the following steps:

1 Switch the HX20 off and install the supplied ROM or ROMs according the instructions given in the document *Installing EPROMS* supplied with the product

2 Perform a Cold start (see section 1.1.2 of *HX-20 BASIC Reference Manual*)

3 Select **BASIC/ASSEMBLER** from the system menu

You can now use the Assembler whenever you enter BASIC

This appendix contains details of all HD6301 instructions for use with the assembler. The instructions are given in tabular form and are divided into the following six categories:

- Data movement
- Arithmetic
- Logical
- Comparison and test
- Branch
- Program transfer and miscellaneous

The tables are divided into four main columns. The first column provides a brief description of the operation. The second column lists the mnemonic, including the register if required.

The third column is divided into five sub-columns, one for each possible addressing mode. The addressing modes are: immediate (Imm), direct (Dir), indexed (Ind), extended (Ext) and implied (Imp). The possible entries in a sub-column are as follows:

o The addressing mode is valid for the instruction

• The addressing mode is valid for the instruction. The instruction may be relocated if a non-zero offset is specified in an *ORG* command

Blank The addressing mode is not valid for the instruction

The fourth column is used to specify the effect of the instruction on the six flags: half carry (H), interrupt (I), negative (N), zero (Z), overflow (V) and carry (C). The possible entries under each flag are as follows:

R The flag is reset to zero

S The flag is set to one

X The flag may be set or reset depending on the result of the operation performed by the instruction

• The flag is unchanged by the instruction

Table A2-1
Data movement instructions

Operation	Mnemonic	Mode					Flags					
		Imm	Dir	Ind	Ext	Imp	H	I	N	Z	V	C
Clear carry	CLC					o	♦	♦	♦	♦	♦	R
Clear interrupt	CLI					o	♦	R	♦	♦	♦	♦
Clear accumulator or memory location	CLR A					o	♦	♦	R	S	R	R
	CLR B					o	♦	♦	R	S	R	R
	CLR			o	●		♦	♦	R	S	R	R
Clear overflow	CLV					o	♦	♦	♦	♦	R	♦
Load accumulator	LDA A	o	o	o	●		♦	♦	X	X	R	♦
	LDA B	o	o	o	●		♦	♦	X	X	R	♦
Load double accumulator	LDA D	o	o	o	●		♦	♦	X	X	R	♦
	LDD	●										
Load SP	LDS	●	o	o	●		♦	♦	X	X	R	♦
Load X	LDX	●	o	o	●		♦	♦	X	X	R	♦
Push register on to stack	PSH A					o	♦	♦	♦	♦	♦	♦
	PSH B					o	♦	♦	♦	♦	♦	♦
	PSH X					o	♦	♦	♦	♦	♦	♦
Pull register from stack	PUL A					o	♦	♦	♦	♦	♦	♦
	PUL B					o	♦	♦	♦	♦	♦	♦
	PUL X					o	♦	♦	♦	♦	♦	♦
Set carry	SEC					o	♦	♦	♦	♦	♦	S
Set interrupt	SEI					o	♦	S	♦	♦	♦	♦
Set overflow	SEV					o	♦	♦	♦	♦	S	♦

Operation	Mnemonic	Mode Imm	Dir	Ind	Ext	Imp	Flags H	I	N	Z	V	C
Store accumulator	STA A		o	o	●		♦	♦	X	X	R	♦
	STA B		o	o	●		♦	♦	X	X	R	♦
Store double accumulator	STA D STD		o	o	●		♦	♦	X	X	R	♦
Store SP	STS		o	o	●		♦	♦	X	X	R	♦
Store X	STX		o	o	●		♦	♦	X	X	R	♦
Transfer A to B	TAB					o	♦	♦	X	X	R	♦
Transfer A to CCR	TAP					o	See (1)					
Transfer B to A	TBA					o	♦	♦	X	X	R	♦
Transfer CCR to A	TPA					o	♦	♦	♦	♦	♦	♦
Load X with SP+1	TSX					o	♦	♦	♦	♦	♦	♦
Load SP with X-1	TXS					o	♦	♦	♦	♦	♦	♦
Exchange D and X	XDX XGD X					o	♦	♦	♦	♦	♦	♦

(1) CCR is loaded with the contents of A as follows:

Bit in A	*Flag in CCR*
5	H
4	I
3	N
2	Z
1	V
0	C

Table A2-2
Arithmetic instructions

Operation	Mnemonic	Mode					Flags					
		Imm	Dir	Ind	Ext	Imp	H	I	N	Z	V	C
Add B to A	ABA					o	X	◆	X	X	X	X
Add B to X	ABX					o	◆	◆	◆	◆	◆	◆
Add with carry to accumulator	ADC A	o	o	o	●		X	◆	X	X	X	X
	ADC B	o	o	o	●		X	◆	X	X	X	X
Add to accumulator	ADD A	o	o	o	●		X	◆	X	X	X	X
	ADD B	o	o	o	●		X	◆	X	X	X	X
Add double	ADD D	o	o	o	●		◆	◆	X	X	X	X
Arithmetic shift left accumulator or memory location	ASL A					o	◆	◆	X	X	X	X
	ASL B					o	◆	◆	X	X	X	X
	ASL			o	●		◆	◆	X	X	X	X
Double ASL	ASL D					o	◆	◆	X	X	X	X
Arithmetic shift right accumulator or memory location	ASR A					o	◆	◆	X	X	X	X
	ASR B					o	◆	◆	X	X	X	X
	ASR			o	●		◆	◆	X	X	X	X
Decimal adjust A	DAA					o	◆	◆	X	X	X	X
Decrement accumulator or memory location	DEC A					o	◆	◆	X	X	X	◆
	DEC B					o	◆	◆	X	X	X	◆
	DEC			o	●		◆	◆	X	X	X	◆
Decrement SP	DES					o	◆	◆	◆	◆	◆	◆
Decrement X	DEX					o	◆	◆	◆	X	◆	◆
Increment accumulator or memory location	INC A					o	◆	◆	X	X	X	◆
	INC B					o	◆	◆	X	X	X	◆
	INC			o	●		◆	◆	X	X	X	◆

Operation	Mnemonic	Mode					Flags					
		Imm	Dir	Ind	Ext	Imp	H	I	N	Z	V	C
Increment SP	INS					o	♦	♦	♦	♦	♦	♦
Increment X	INX					o	♦	♦	♦	X	♦	♦
Multiply A by B	MUL					o	♦	♦	♦	♦	♦	X
Negate accumulator or memory location	NEG A					o	♦	♦	X	X	X	X
	NEG B					o	♦	♦	X	X	X	X
	NEG			o	•		♦	♦	X	X	X	X
Subtract B from A	SBA					o	♦	♦	X	X	X	X
Subtract with carry from accumulator	SBC A	o	o	o	•		♦	♦	X	X	X	X
	SBC B	o	o	o	•		♦	♦	X	X	X	X
Subtract from accumulator	SUB A	o	o	o	•		♦	♦	X	X	X	X
	SUB B	o	o	o	•		♦	♦	X	X	X	X
Subtract double	SUB D	o	o	o	•		♦	♦	X	X	X	X

Table A2-3
Logical instructions

Operation	Mnemonic	Mode: Imm	Dir	Ind	Ext	Imp	Flags: H	I	N	Z	V	C
AND immediate	AIM #n,		o	o			♦	♦	X	X	R	♦
AND accumulator	AND A	o	o	o	●		♦	♦	X	X	R	♦
	AND B	o	o	o	●		♦	♦	X	X	R	♦
Ones complement accumulator or memory location	COM A					o	♦	♦	X	X	R	S
	COM B					o	♦	♦	X	X	R	S
	COM			o	●		♦	♦	X	X	R	S
EOR immediate	EIM #n,		o	o			♦	♦	X	X	R	♦
Exclusive OR accumulator or memory location	EOR A	o	o	o	●		♦	♦	X	X	R	♦
	EOR B	o	o	o	●		♦	♦	X	X	R	♦
Logical shift right accumulator or memory location	LSR A					o	♦	♦	R	X	X	X
	LSR B					o	♦	♦	R	X	X	X
	LSR			o	●		♦	♦	R	X	X	X
Double LSR	LSR D					o	♦	♦	R	X	X	X
OR immediate	OIM #n,		o	o			♦	♦	X	X	R	♦
Inclusive OR accumulator or memory location	ORA A	o	o	o	●		♦	♦	X	X	R	♦
	ORA B	o	o	o	●		♦	♦	X	X	R	♦
Rotate left accumulator or memory location	ROL A					o	♦	♦	X	X	X	X
	ROL B					o	♦	♦	X	X	X	X
	ROL			o	●		♦	♦	X	X	X	X
Rotate right accumulator or memory location	ROR A					o	♦	♦	X	X	X	X
	ROR B					o	♦	♦	X	X	X	X
	ROR			o	●		♦	♦	X	X	X	X

Table A2-4
Comparison and test instructions

Operation	Mnemonic	Mode					Flags					
		Imm	Dir	Ind	Ext	Imp	H	I	N	Z	V	C
Bit test accumulator	BIT A	o	o	o	•		♦	♦	X	X	R	♦
	BIT B	o	o	o	•		♦	♦	X	X	R	♦
Compare B with A	CBA					o	♦	♦	X	X	X	X
Compare accumulator	CMP A	o	o	o	•		♦	♦	X	X	X	X
	CMP B	o	o	o	•		♦	♦	X	X	X	X
Compare X	CPX	o	o	o	•		♦	♦	X	X	X	X
BIT immediate	TIM #n,		o	o			♦	♦	X	X	R	♦
Test accumulator or memory location for positive, zero or negative	TST A					o	♦	♦	X	X	R	R
	TST B					o	♦	♦	X	X	R	R
	TST			o	•		♦	♦	X	X	R	R

Table A2-5
Branch instructions

Operation	Mnemonic	Branch test		Flags					
		Unsigned	Signed	H	I	N	Z	V	C
Branch if C clear	BCC	>=		♦	♦	♦	♦	♦	♦
Branch if C set	BCS	<		♦	♦	♦	♦	♦	♦
Branch if Z set	BEQ	=	=	♦	♦	♦	♦	♦	♦
Branch if >= zero	BGE		>=	♦	♦	♦	♦	♦	♦
Branch if > zero	BGT		>	♦	♦	♦	♦	♦	♦
Branch if > zero	BHI	>		♦	♦	♦	♦	♦	♦
Branch if <= zero	BLE		<=	♦	♦	♦	♦	♦	♦
Branch if <= zero	BLS	<=		♦	♦	♦	♦	♦	♦
Branch if < zero	BLT		<	♦	♦	♦	♦	♦	♦
Branch if minus	BMI		< 0	♦	♦	♦	♦	♦	♦
Branch if Z clear	BNE	<>	<>	♦	♦	♦	♦	♦	♦
Branch if plus	BPL		>= 0	♦	♦	♦	♦	♦	♦
Branch always	BRA			♦	♦	♦	♦	♦	♦
Branch never	BRN			♦	♦	♦	♦	♦	♦
Branch subroutine	BSR			♦	♦	♦	♦	♦	♦
Branch if V clear	BVC		No error	♦	♦	♦	♦	♦	♦
Branch if V set	BVS		Error	♦	♦	♦	♦	♦	♦

Note: The assembler expects an extended mode address to be supplied as the destination of a branch instruction, and automatically converts the address into the relative offset required by the HD6301.

Table A2-6
Program transfer and miscellaneous instructions

Operation	Mnemonic	Mode					Flags					
		Imm	Dir	Ind	Ext	Imp	H	I	N	Z	V	C
Jump	JMP			o	•		♦	♦	♦	♦	♦	♦
Jump subroutine	JSR		o	o	•		♦	♦	♦	♦	♦	♦
No operation	NOP					o	♦	♦	♦	♦	♦	♦
Interrupt return	RTI					o	See (1)					
Subroutine return	RTS					o	♦	♦	♦	♦	♦	♦
Sleep	SLP					o	♦	♦	♦	♦	♦	♦
Interrupt	SWI					o	♦	S	♦	♦	♦	♦
Await interrupt	WAI					o	♦	S	♦	♦	♦	♦

(1) The CCR is loaded from the stack

/0 11 Division by zero
The divisor is zero
- The divisor is an undefined label

BF 51 Bad file mode
The file number used in a *LST* command refers to a file not opened for output.

BN 50 Bad file number
The file number used in a *LST* command either refers to a file not opened for output, or is not an integer in the range 1 to 16.

BS 9 Bad subscript
An error has occurred when using global labels with the RAM file
- The RAM file is too small for the number of global labels used
- The RAM file record size is less than three bytes

CN 17 Cannot continue
An attempt is made to restart assembly after a break-in or error

DD 10 Duplicate definition
A label is assigned a value twice in RDF N mode
- The same label is defined more than once
- A label is assigned a different value in pass 1 and 2

FC 5 Function call error
An incorrect value has been used as a parameter
- The destination of a branch instruction is out of range
- A value less than zero, or greater than 255, is used as an eight bit data item
- The value of MEMSET is too low for the assembled object code
- The assembled object code is located below &HA40 or above the current value of MEMSET

IO 53 Device I/O error
The device used for the listing file is faulty or has responded to the break key

IU 59 Device in use
The device used for the listing file is being used by another process, or has been incorrectly aborted

MO 22 Missing operand
A parameter is missing from a command or an instruction

NO 57 File not open
The BASIC file used for the listing file has been closed or was never opened

OM 7 Out of memory
Insufficient memory space for the symbol table

OV 6 Overflow
The result of an expression is outside the range -32768 to 32767

SN 2 Syntax error
The format required for the command or instruction has not been followed

- Missing or incorrect register name or parameter
- A label starts with a keyword or is missing a "." or "!"
- Illegal mnemonic or command name
- Missing space after command or mnemonic

This appendix gives the complete listing for a simple clock program. The listing is entered using the standard BASIC screen editor. The clock program is executed automatically once the source code is assembled by RUNning the program. You can stop the clock program by pressing the BREAK key. The clock can be re-started by pressing CTRL PF3.

A4.1 Listing of the clock program

```
1000 DEFINT A-Z:I=1:GOSUB 1050:MEMSET CLOCKEND
1010 WIDTH "LPT0:",24:OPEN "O",1,"LPT0:"
1020 DEFINT A-Z:FOR I=1 TO 2:GOSUB 1050:NEXT I
1030 EXEC CLOCK
1040 END
1050 '
1060 '-----------------------------------------------
1070 '
1080             ASM I
1090 ;
1100             TTL "PROGRAM TO DISPLAY CLOCK ON LCD"
1110 ;
1120             FMT 255,24,,"","" : LST 1
1130 ;
1140             OBJ
1150 ;
1160 SNSCOM      EQU $FF19   ;SEND BYTE TO SLAVE
1170 DSPLCN      EQU $FF49   ;CLEAR SCREEN
1180 DSPLCH      EQU $FF4C   ;DISPLAY CHARACTER
1190 CNTIO       EQU $FFAF   ;CONTINUE I/O AFTER BREAK
1200 LCRECV      EQU $DFEE   ;RECOVER VIRTUAL SCREEN
1210 RDCLK       EQU $E1FA   ;READ TIME
1220 SLEEP       EQU $FFA9   ;SLEEP MODE ROUTINE
1230 ;
1240 TICK        EQU 800     ;TICK FREQUENCY
1250 DURATION    EQU 5       ;TICK DURATION
1260 SLVSPCOM    EQU $31     ;SLAVE SPEAKER COMMAND
1270 XPOS        EQU 6       ;X CO-ORD OF CLOCK ON LCD
1280 YPOS        EQU 1       ;Y CO-ORD OF CLOCK ON LCD
1290 COLON       EQU "\:"    ;SEPARATOR
1300 BREAKFLAG   EQU $80     ;<BREAK> KEY FLAG
1310 PHYSFLAG    EQU $40     ;PHYSICAL SCREEN FLAG
1320 UIEFLAG     EQU $10     ;CLOCK INTERRUPT ENABLE FLAG
1330 CLKINTFLG   EQU 8       ;CLOCK INTERRUPT FLAG
1340 CLKREGB     EQU $4B     ;CLOCK REGISTER
1350 RNMOD       EQU $7B     ;RUN MODE VARIABLE
1360 MIOSTS      EQU $7D     ;MASTER I/O STATUS
1370 CT3ADR      EQU $126    ;CONTROL PF3 VECTOR
1380 BUFFER      EQU $190    ;6 BYTE BUFFER CLOCK BUFFER
1390 ;
```

```
1400 ;-----------------------------------------------------
1410 ;
1420            ORG $A40
1430 ;
1440 CLOCK
1450 ;
1460 ;SET PHYSICAL SCREEN FLAG
1470            OIM #PHYSFLAG,RNMOD
1480 ;
1490 ;CLEAR PHYSICAL SCREEN
1500            CLR B
1510            JSR DSPLCN
1520 ;
1530 ;ENABLE CLOCK INTERRUPTS ONCE PER SECOND
1540            OIM #UIEFLAG,CLKREGB
1550 ;
1560 ;SET UP CONTROL PF3 VECTOR
1570            LDX #CLOCK
1580            STX CT3ADR
1590 ;
1600 ;
1610 MAINLOOP
1620 ;
1630 ;TEST FOR <BREAK> KEY
1640            TIM #BREAKFLAG,MIOSTS
1650            BNE EXIT
1660 ;
1670 ;SLEEP UNTIL INTERRUPT
1680            JSR SLEEP
1690 ;
1700 ;CHECK CLOCK INTERRUPT
1710            TIM #CLKINTFLG,MIOSTS
1720            BEQ MAINLOOP
1730 ;
1740 ;RESET CLOCK INTERRUPT FLAG
1750            EIM #CLKINTFLG,MIOSTS
1760 ;
1770 ;DISPLAY COLONS ON LCD
1780            LDA A #COLON
1790            LDX #(XPOS+2)*256+YPOS
1800            JSR DSPLCH
1810            LDA A #COLON
1820            LDX #(XPOS+5)*256+YPOS
1830            JSR DSPLCH
1840 ;
1850 ;READ AND DISPLAY TIME
1860            BSR DISPLAYTIME
1870 ;
1880 ;PRODUCE TICK
1890            BSR PLAYTICK
1900 ;
1910 ;BRANCH BACK FOR NEXT SECOND
1920            BRA MAINLOOP
1930 ;
1940 ;
```

```
EXIT
;
;RESTORE STATUS BEFORE RETURNING TO BASIC
;
                ;RESET BREAK FLAG
                JSR CNTIO
                ;
                ;RESET PHYSICAL SCREEN FLAG
                EIM #PHYSFLAG,RNMOD
;
                ;DISABLE CLOCK INTERRUPT
                EIM #UIEFLAG,CLKREGB
;
                ;RESTORE VIRTUAL SCREEN
                JSR LCRECV
;
                ;RETURN TO BASIC
                RTS
;
;------------------------------------------------------------
;
DISPLAYTIME
;
;DISPLAY TIME ON LCD
;
                ;DISABLE INTERRUPTS
                SEI
;
                ;READ TIME
                LDX #BUFFER
                JSR RDCLK
;
                ;DISPLAY HOURS
                LDA A X+3
                LDA B #XPOS
                BSR DISPLAY
;
                ;DISPLAY MINUTES
                LDA A X+4
                LDA B #XPOS+3
                BSR DISPLAY
;
                ;DISPLAY SECONDS
                LDA A X+5
                LDA B #XPOS+6
                BSR DISPLAY
;
                ;ENABLE INTERRUPTS
                CLI
;
                RTS
;
;------------------------------------------------------------
;
DISPLAY
```

```
2500 ;
2510 ;DISPLAY 2 DIGITS ON LCD
2520 ;A = 2 BCD DIGITS
2530 ;B = POSITION ON LCD
2540 ;
2550            PSH A
2560            LSR A
2570            LSR A
2580            LSR A
2590            LSR A
2600            BSR DIGIT
2610            PUL A
2620 DIGIT
2630            PSH X
2640            AND A #$F
2650            ADD A #"0"
2660            PSH A
2670            TBA
2680            LDA B #YPOS
2690            XGD X
2700            PUL A
2710            JSR DSPLCH
2720            XGD X
2730            TAB
2740            PUL X
2750 ;
2760            RTS
2770 ;
2780 ;-----------------------------------------------
2790 ;
2800 PLAYTICK
2810 ;
2820 ;PRODUCE TICK
2830 ;
2840            LDA A #SLVSPCOM
2850            JSR SNSCOM
2860            LDA A #TICK \ 256
2870            JSR SNSCOM
2880            LDA A #TICK MOD 256
2890            JSR SNSCOM
2900            LDA A #DURATION \ 256
2910            JSR SNSCOM
2920            LDA A #DURATION MOD 256
2930            JSR SNSCOM
2940 ;
2950            RTS
2960 ;
2970 ;-----------------------------------------------
2980 ;
2990 CLOCKEND   EQU *
3000            ASM OFF
3010 '
3020 '-----------------------------------------------
3030 '
3040 RETURN
```

This appendix gives the complete listing for multiple file assembly of the simple clock program described in Appendix 4. The listings of the three files are entered using the standard BASIC screen editor. The clock program is executed automatically once the source code is assembled by RUNning the header program. You can stop the clock program by pressing the BREAK key. The clock can be re-started by pressing CTRL PF3.

A5.1 Header file

The header file consists of the following program:

```
10 CLEAR 200,8*12+4                 '12 GLOBAL LABELS
20 DEFFIL 2,0:PUT%0,1               'SET UP PASS NUMBER
30 OPEN "O",1,"LPT0:"               'OPEN LISTING FILE
40 WIDTH "LPT0:",24                 'WIDTH OF LISTING FILE
50 T=TAPCNT:PUT%1,T                 'SAVE TAPE POSITION
60 RUN "CLOCK1.SRC",R
```

A5.2 First source file

```
1000 DEFINT A-Z
1010 DEFFIL 2,0:GET%0,P                 'PASS NUMBER
1015 DEFFIL 8,4                         'SET UP RAM FILE
1020 IF P=1 THEN I=1 GOSUB 1050
1030 IF P=2 THEN FOR I=3 TO 4:GOSUB 1050:NEXT I
1040 RUN "CLOCK2.SRC",R
1050 '
1060 '-------------------------------------------------
1070 '
1080            ASM I
1100            TTL "PROGRAM TO DISPLAY CLOCK ON LCD"
1120            FMT 255,24,,"","" : LST 1
1140            OBJ
1150 ;
1160 !SNSCOM    EQU $FF19   ;SEND BYTE TO SLAVE
1210 !RDCLK     EQU $E1FA   ;READ TIME
1180 !DSPLCH    EQU $FF4C   ;DISPLAY CHARACTER
1190 CNTIO      EQU $FFAF   ;CONTINUE I/O AFTER BREAK
1170 DSPLCN     EQU $FF49   ;CLEAR SCREEN
1200 LCRECV     EQU $DFEE   ;RECOVER VIRTUAL SCREEN
1220 SLEEP      EQU $FFA9   ;SLEEP MODE ROUTINE
1230 ;
1300 !BUFFER    EQU $190    ;6 BYTE CLOCK BUFFER
1240 !TICK      EQU 800     ;TICK FREQUENCY
1250 !DURATION  EQU 5       ;TICK DURATION
1260 !SLVSPCOM  EQU $31     ;SLAVE SPEAKER COMMAND
1270 !XPOS      EQU 6       ;X CO-ORD OF CLOCK ON LCD
```

```
1280 !YPOS      EQU 1       ;Y CO-ORD OF CLOCK ON LCD
1290 COLON      EQU "\:"    ;SEPARATOR
1300 BREAKFLAG  EQU $80     ;<BREAK> KEY FLAG
1310 PHYSFLAG   EQU $40     ;PHYSICAL SCREEN FLAG
1320 UIEFLAG    EQU $10     ;CLOCK INTERRUPT ENABLE FLAG
1330 CLKINTFLG  EQU 8       ;CLOCK INTERRUPT FLAG
1340 CLKREGB    EQU $4B     ;CLOCK REGISTER
1350 RNMOD      EQU $7B     ;RUN MODE VARIABLE
1360 MIOSTS     EQU $7D     ;MASTER I/O STATUS
1370 CT3ADR     EQU $126    ;CONTROL PF3 VECTOR
1410 ;-----------------------------------------------------------------
1420            ORG $A40
1440 !CLOCK
1460 ;SET PHYSICAL SCREEN FLAG
1470            OIM #PHYSFLAG,RNMOD
1490 ;CLEAR PHYSICAL SCREEN
1500            CLR B
1510            JSR DSPLCN
1530 ;ENABLE CLOCK INTERRUPTS ONCE PER SECOND
1540            OIM #UIEFLAG,CLKREGB
1560 ;SET UP CONTROL PF3 VECTOR
1570            LDX #!CLOCK
1580            STX CT3ADR
1610 MAINLOOP
1630 ;TEST FOR <BREAK> KEY
1640            TIM #BREAKFLAG,MIOSTS
1650            BNE EXIT
1670 ;SLEEP UNTIL INTERRUPT
1680            JSR SLEEP
1700 ;CHECK CLOCK INTERRUPT
1710            TIM #CLKINTFLG,MIOSTS
1720            BEQ MAINLOOP
1740 ;RESET CLOCK INTERRUPT FLAG
1750            EIM #CLKINTFLG,MIOSTS
1770 ;DISPLAY COLONS ON LCD
1780            LDA A #COLON
1790            LDX #(!XPOS+2)*256+!YPOS
1800            JSR !DSPLCH
1810            LDA A #COLON
1820            LDX #(!XPOS+5)*256+!YPOS
1830            JSR !DSPLCH
1850 ;READ AND DISPLAY TIME
1860            BSR !DISPLAYTIME
1880 ;PRODUCE TICK
1890            BSR !PLAYTICK
1910 ;BRANCH BACK FOR NEXT SECOND
1920            BRA MAINLOOP
1950 EXIT
1970 ;RESTORE STATUS BEFORE RETURNING TO BASIC
1990            ;RESET BREAK FLAG
2000            JSR CNTIO
2020            ;RESET PHYSICAL SCREEN FLAG
2030            EIM #PHYSFLAG,RNMOD
2050            ;DISABLE CLOCK INTERRUPT
2060            EIM #UIEFLAG,CLKREGB
```

```
2080            ;RESTORE VIRTUAL SCREEN
2090            JSR LCRECV
2110            ;RETURN TO BASIC
2120            RTS
2130 ;-------------------------------------------
2155            ASM OFF
2165 '-------------------------------------------
2170 RETURN
```

A5.3 Second source file

The second source file ("CLOCK2.SRC") consists of the following program:

```
1000 DEFFINT A-Z
1010 DEFFIL 2,0:GET%0,P              'GET PASS NUMBER
1020 DEFFIL 8,4                      'SET UP RAM FILE
1030 IF P=1 THEN 1080
1040 FOR I=5 TO 6:GOSUB 2100:NEXT I:CLOSE
1050 EXEC (!CLOCK)                   'EXECUTE OBJECT CODE
1060 GOTO 1140
1080 I=5:GOSUB 2100
1090 DEFFIL 2,0
1100 PUT%0,2                         'SET UP PASS 2
1110 GET%1,T:WIND T                  'WIND TO START
1120 MEMSET CLOCKEND                 'ALLOCATE OBJECT CODE SPACE
1130 RUN "CLOCK1.SRC",R
1140 END
2100 '
2105 '-------------------------------------------
2110 '
2120            ASM I
2130 ;
2160 !DISPLAYTIME
2180 ;DISPLAY TIME ON LCD
2200            ;DISABLE INTERRUPTS
2210            SEI
2230            ;READ TIME
2240            LDX #!BUFFER
2250            JSR !RDCLK
2270            ;DISPLAY HOURS
2280            LDA A X+3
2290            LDA B #!XPOS
2300            BSR DISPLAY
2320            ;DISPLAY MINUTES
2330            LDA A X+4
2340            LDA B #!XPOS+3
2350            BSR DISPLAY
2370            ;DISPLAY SECONDS
2380            LDA A X+5
2390            LDA B #!XPOS+6
2400            BSR DISPLAY
2420            ;ENABLE INTERRUPTS
2430            CLI
```

```
2450            RTS
2470 ;-----------------------------------------------
2490 DISPLAY
2510 ;DISPLAY 2 DIGITS ON LCD
2520 ;A = 2 BCD DIGITS
2530 ;B = POSITION ON LCD
2550            PSH A
2560            LSR A
2570            LSR A
2580            LSR A
2590            LSR A
2600            BSR DIGIT
2610            PUL A
2620 DIGIT
2630            PSH X
2640            AND A #$F
2650            ADD A #"0"
2660            PSH A
2670            TBA
2680            LDA B #!YPOS
2690            XGD X
2700            PUL A
2710            JSR !DSPLCH
2720            XGD X
2730            TAB
2740            PUL X
2760            RTS
2780 ;-----------------------------------------------
2800 !PLAYTICK
2820 ;PRODUCE TICK
2840            LDA A #!SLVSPCOM
2850            JSR !SNSCOM
2860            LDA A #!TICK \ 256
2870            JSR !SNSCOM
2880            LDA A #!TICK MOD 256
2890            JSR !SNSCOM
2900            LDA A #!DURATION \ 256
2910            JSR !SNSCOM
2920            LDA A #!DURATION MOD 256
2930            JSR !SNSCOM
2950            RTS
2970 ;-----------------------------------------------
2990 CLOCKEND   EQU *
3000            ASM OFF
3010 '-----------------------------------------------
3030 '
3040 RETURN
```

Index entries refer to chapters or to sections within chapters. The main reference is listed first. Note that C*n* refers to Chapter *n*, A*n* to Appendix *n* and TA2-*n* to table A2-*n* in Appendix 2.

71 May Tree Close, Winchester, SO22 4JF