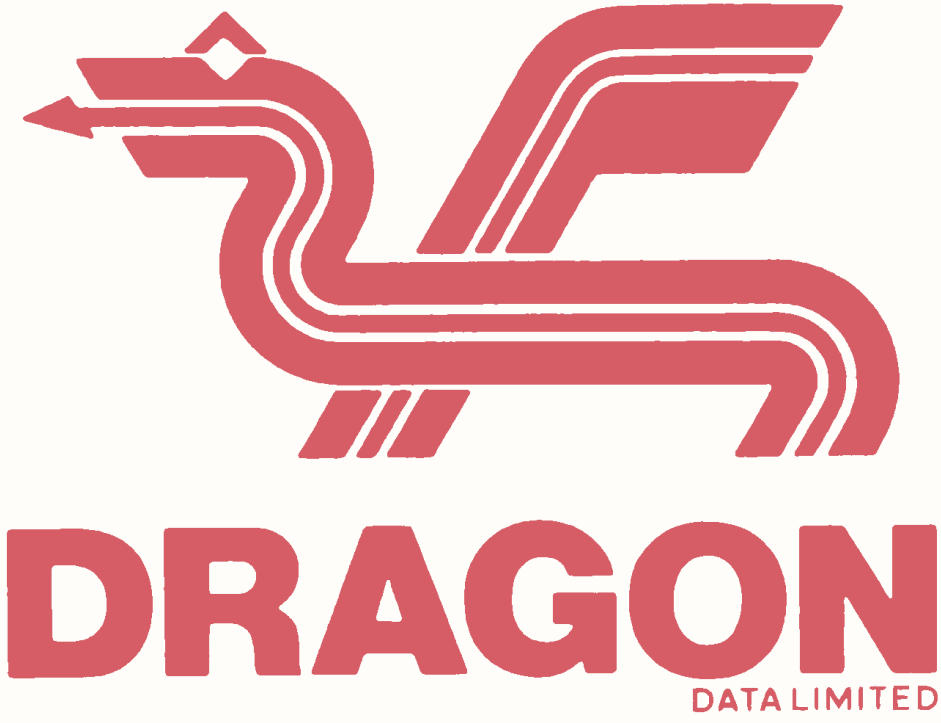
DREAM

AO516



WELCOME TO DRAGON

Dragon Data Limited welcome you to your new software for your Dragon computer. We hope that you enjoy using it as much as we enjoyed producing it for you.

Look out for new titles in the Dragon software range.

DREAM

AO516

DRAGON 6809 EDITOR-ASSEMBLER CONTENTS

LOADING THE PACKAGE 2
USING THE EDITOR 4
CASSETTE INPUT-OUTPUT 13
6809 PROGRAMMING 19
6809 ADDRESSING MODES 22
ASSEMBLER DIRECTIVES 32
ASSEMBLER OPERATION 37
SUPPLEMENTARY INFORMATION 42
APPENDIX A SAMPLE MEMORY MAP 44
APPENDIX B EDITOR OPERATION 47
APPENDIX C1 INSTRUCTION OP-CODES 50
APPENDIX C2 BRANCH INSTRUCTIONS 53
APPENDIX C3 ASSEMBLER DIRECTIVES 54
APPENDIX D ASSEMBLER ERROR CODES & KEYBOARD COMMANDS 54
APPENDIX E SYSTEM ERROR MESSAGES 56
APPENDIX F SAMPLE PROGRAMS 58

LICENCE

Dream, in all machine readable formats and the written documentation accompanying them, are copyrighted. The purchase of Dream conveys to the purchaser a licence to use Dream for his/her own use and not for sale or free distribution to others. No other licence, expressed or implied, is granted.

DREAM

DREAM is a software package for the Dragon 32 Micro Computer, designed to enable the creation of files of text, and especially for the preparation, maintenance and conversion to object code, of Assembler source programs for the 6809 micro-processor.

DREAM will produce machine code routines which can be called from BASIC programs, using the USR function.

Loading the Package

Before loading DREAM from cassette you must first tell the Dragon to reserve some memory for the package and its work areas. DREAM normally resides just below the BASIC ROM, and will use RAM just below itself to hold the text table and other information. Space is reserved by using the CLEAR statement. A good starting point

might be:

CLEAR 200,20000

This will reserve 4992 bytes for DREAM (27776 to 32767) and 7775 bytes for it to use (20001 to 27775) — sufficient for Assembler programs up to about 450 typical length lines, or about 230 full lines of text. About 18k still remains for a BASIC program, 200 bytes of string space, and four high resolution graphics pages. Appendix A shows a sample memory map with DREAM installed.

Now you can load DREAM. As it is a machine code program, you must use the CLOADM command. Type:

CLOADM "DREAM" and press ENTER

After about 30 seconds, the OK message should appear. (If you haven't reserved memory, DREAM will not load correctly.)

Now you can execute DREAM by typing:

EXEC

A title screen will appear, and the question

OLD TEXT?

DREAM is asking whether you already have some text in memory which you want to display. As you haven't answer no by typing.

N

A blank screen will appear with the cursor flashing in the top left hand corner. You are now ready to type in some text, or maybe your first program.

It is suggested that you first get familiar with the Editor by using it to manipulate some general text.

Using the Editor

All the keys on the Dragon keyboard will operate typomatically in DREAM. That is, if you keep a key depressed for more than about a second, then it will repeat at the rate of ten per second. You will find this particularly handy for cursor positioning and scrolling.

DREAM can handle normal and inverse video characters. While the cursor is moving,

its position is indicated by a reversal of background colour of the character under the cursor. Inverse video characters will be sent as lower case letters to an attached printer. Lower case can be obtained by using SHIFT while upper case lock is set, and vice versa.

Type some words onto the top line. You can use → and ← to position the cursor anywhere on the line. If you move the cursor over some typed characters you will notice that they are not erased — you can correct any characters without re-typing the rest of the line. Nor do you need to pass the cursor over the rest of a line after making any changes.

The CLEAR key can be used to blank out the rest of a line. All characters under and to the right of the cursor are erased — only the current line is affected.

Pressing SHIFT and SPACE causes a skip to the next TAB column. These are predefined to columns 8, 14, 23 and 31.

Inserting and Deleting Characters

To delete characters within a line, position the cursor on the leftmost unwanted character and press shift and ← together. The line will 'close up' and spaces are

shifted into the right hand end To insert characters, use shift and → and a gap will open up at the cursor position ready for you to type the missing characters A word of warning any characters shifted off the end of the line are lost

When the cursor reaches column 32 it will not move as further characters are typed You need to press ENTER to get onto the next line to continue typing Pressing ENTER also causes DREAM to accept the last line and store it in its text table in memory — this is true of any command that moves the cursor onto a different line

Scrolling

Type something onto each line of the screen — using ENTER to access subsequent lines When you reach the foot of the screen keep entering lines and the whole display will scroll up each time you press ENTER

Press the up arrow and keep it pressed to go into typomatic mode When the cursor reaches the top of the screen the display will scroll back again until the first line you entered is again displayed Using the four arrow keys you can position the cursor anywhere within the text ready to overtype, insert or delete characters at that position

A faster way of scrolling is to press SHIFT and ↓ or SHIFT and ↑ These cause the display to scroll by 8 lines (half a screen) at a time Whenever possible the cursor is kept on the same logical line

The Editor Commands

So far we have discussed cursor positioning, scrolling, and general text editing DREAM also has a wide range of 'commands' which are accessed by typing the BREAK key followed by a letter indicating the required action As soon as you press BREAK the current line is erased and an inverse oblique appears in column 1 to show that you are in command mode The cursor is positioned in column 2 ready for you to type in the command The data from the current line is not lost — it will be re-instated when the command has been completed Commands are not executed until either ENTER or an up or down arrow key is pressed Appendix B lists all the Editor commands.

The HOME command displays the first 16 lines, putting the cursor at the start of line 1 The END command shows the last 8 lines with the cursor on the last line positioned half way down the screen With the cursor sitting anywhere within the text table, type BREAK followed by the letter H (for HOME) or E (for END) Press ENTER and the command is obeyed instantly.

The QUIT command exits from DREAM back to BASIC Type BREAK then Q and ENTER You can then re-enter DREAM by typing EXEC and reply Y to the OLD TEXT? prompt to re-display the text in memory, or reply N to erase the table ready for something new

Inserting Lines

The method of inserting new lines is to first add a block of blank lines at the required point, and then to type the new data onto those blank lines. Position the cursor anywhere on the line prior to which the new lines are to appear, and type BREAK I followed by a number indicating the number of lines to insert. If you don't specify a number then 1 line will be inserted Any number of lines can be inserted up to a practical limit of about 6000 depending on the amount of memory you have reserved for DREAM Press ENTER and the requested number of blank lines will be inserted, leaving the cursor at the start of the first new line, ready for you to type in the new data The line over which you typed the insert command is re-instated to its original data content

If you attempt to insert more lines than will fit in the workspace, DREAM will insert as many lines as it can, and will then clear the screen and show the message FULL. Press any key to see the text table as it stands. You will now need to delete one or

more lines before making any further changes

Blank lines only occupy 4 bytes each in the text table As you fill them with data so they grow in length up to 35 bytes for a full line Hence the FULL message may appear any time an extended line is accepted You can always exit from DREAM (QUIT command), reserve more memory (via CLEAR), and re-enter DREAM with OLD TEXT ?' equals Y.

Line Deletion

To delete lines, position the cursor on the first line to delete, and type BREAK Dn where 'n' is the number of lines to be deleted Omit 'n' to delete just one line 'n' can be any number up to about 65000 but DREAM will never delete the last line of the text table. You can use this fact to keep your own 'end of file' marker on the last line if you like.

Lines can also be deleted by first using the MARK and END-MARK commands (cf) to mark a block of lines, then type BREAK DM to delete the block

Whenever a line is deleted, or when a line is replaced by a shorter line, DREAM always compresses the text table to drop the redundant material Hence the text table always

occupies as little valuable memory space as possible. Deleting a lot of lines can be relatively slow as DREAM has a lot of housekeeping work to do. A flickering blob on the command line shows that Dream is still at work.

String Searching

DREAM can be made to search the text table for any character string. Searching always starts from the current line. To search the whole text table, first do a HOME command.

The FIND command (BREAK F) will position the cursor on the first occurrence of a string, with that line placed in the middle of the screen (if possible). If the string is not found, the table is positioned at END. The command is entered on the first line to be searched and has the form:

F/string/

Any non-blank character can be used to delimit the string e.g.,

F"12 1/2%"

Whatever character marks the start of the string must also mark the end.

The CHANGE command will search for one or all occurrences of a string, replacing it by a second string. e.g.,

C/string1/string2/	for one occurrence
C/string1/string2/A	for all occurrences

The replacement string can be longer or shorter than the searched string. To remove a string completely, string2 can be a 'null' string. e.g.,

C/deletable//

If a new string is longer, take care, as it may cause characters to be shifted off the right end of a line are lost.

Repeating a FIND or CHANGE

Pressing SHIFT and @ together will repeat the last FIND or CHANGE command, starting at the next line. For example, to find all occurrences of a string, use F/string/ first, then type SHIFT @ repeatedly to find all subsequent occurrences.

Marking a Block of Lines

Any number of continuous lines in the text table can be 'marked' for later use by the REPLICATE, DELETE, PRINT or SAVE commands. First locate the first line of the block to be marked, and type BREAK M (for MARK) on it and press ENTER. Next locate the last line of the block and type BREAK N (for END-MARK) and press ENTER.

The first and last of the marked lines are flagged by a block graphic replacing the right-most character.

Once 'marked' a block stays marked until cleared by the UN-MARK command (BREAK U), or until another block is marked, or until any DELETE or INSERT command is executed.

Duplicating Lines

A marked block of lines can be copied into another part of the text table by using the REPLICATE command (BREAK R). First mark the lines using MARK and END-MARK, then place the cursor on the line in front of which the duplicated lines are be inserted and type BREAK R (for REPLICATE).

On pressing ENTER the lines are copied They have not been deleted from their original position—they now exist twice in the text table You can delete them from the old place by the DM command if required

Don't try to copy lines into a position between the MARK and END-MARK lines You will not get a meaningful result

Cassette Input—Output

Having created a valuable table of text using DREAM you will want to be able to save it on tape and recall it at a later date This can be done by using the SAVE and LOAD commands. To save the complete text table, position the cursor at HOME and enter the command:

S filename

Only one space must exist between the S and the filename, which is not enclosed in quotes. Only the first 8 bytes of 'filename are significant

Set the cassette machine ready to record and press ENTER As each line is recorded the display scrolls up so you can monitor the progress

You can save a selected number of lines only, by adding a parameter to the SAVE command e.g.,

S25 filename

This will create a named text file on cassette consisting of 25 lines starting with the line on which you typed the command. Just one space must follow the number, which must immediately follow the S

A 'marked' block of lines can be saved by typing:

SM filename

This will create a tape file consisting of all lines between those marked by the MARK and END-MARK commands inclusive.

Loading from Cassette

To load a DREAM text file from tape, enter the command:

L filename

observing the syntax rules specified for the SAVE command Only the number of characters you give for 'filename' will be used to match files on tape e g .

L P

will match any file whose name starts with the letter P

As the tape is read, the names of all file headers encountered will be shown on the command line. When a file name that matches the LOAD command is found, a flashing white blob at the right end of the command line indicates that loading is proceeding

Each line is individually parity checked as it is read. If any errors occur an inverse 'E' will appear to the left of the flashing blob Loading proceeds, and the error line is replaced by '???'. This avoids the irretrievable loss of a file should one line get corrupted. Often you can re-construct a line by examining the previous and following lines.

Merging Text Files

The LOAD command can be used to insert a DREAM file from tape into an existing text table in memory. Thus you can save a block of lines from one text table, load

another text file. And insert the saved lines at any point. Do this by entering the LOAD command on the line prior to which insertion is required.

Printing

The whole text table, or a selected number of lines, can be sent to a printer. Position the cursor on the first line to be printed and enter the command:

P	to print all following lines	or
Pn	to print the next 'n' lines	or
PM	to print a 'marked' block of lines	

If you try this command without a printer connected, the Dragon will 'hang'and you will have to press the 'RESET' button to recover. This will take you back into BASIC.

You can pause the printer at the end of a line by pressing BREAK. Restart by typing 'P'. Typing 'O' will terminate printing.

The Dragon allows you some control over the control bytes which are sent to the printer at the end of each line. Location 330 (decimal) is a count of the number of bytes to be sent. Locations 331 onwards contain the required control bytes. When you switch on the Dragon, location 330 is initialised to 1, and locations 331, 332 to the values for CR.

(hex 0D) and LF (hex 0A) If your printer needs both a CR and LF then you should POKE 2 into location 330 before entering DREAM Other line termination sequences can be obtained by doing the appropriate POKE's

Cancelling Command Mode

DREAM commands are not executed until you press ENTER or an up or down arrow If you press a key after BREAK which does not correspond to a recognised command, DREAM will reply with '?' when you press ENTER If you decide you don't want to execute any command, type BREAK again and the data line will re-appear and command mode is cancelled

The sequence BREAK, BREAK is in fact a valid quick way of returning the cursor to column one.

The Recover Command

The RECOVER command (BREAK V) will enable you to recover from a typing slip For example, if you inadvertently hit the CLEAR key while editing a line, and can't remember what you have lost, enter BREAK V and the line will re-appear as it was before you started editing it This will only work if you have the presence of mind not to move the cursor off the current line before using RECOVER

Executing the Assembler

The ASSEMBLE command (BREAK A) will transfer control to the Assembler part of DREAM. Before using this command you will need to read the instructions on how to prepare Assembler source statements, and, if you are new to the 6809, the section on 6809 programming. The Assembler uses the source code you have prepared in the text table, and produces the corresponding machine 'object' code, storing it directly in memory, from where it can be directly executed.

Tabbing

When typing Assembler source programs, it is sometimes desirable, though not necessary, to divide the statements into fixed fields on the screen for neatness and readability. Tab positions have been pre-defined in DREAM at columns 1, 8, 14, 23 and 31, giving convenient positions for Label, Op-code, Operand, and comments. Pressing SHIFT and the Space-bar together causes the cursor to skip to the next Tab position.

Restoring Text Mode

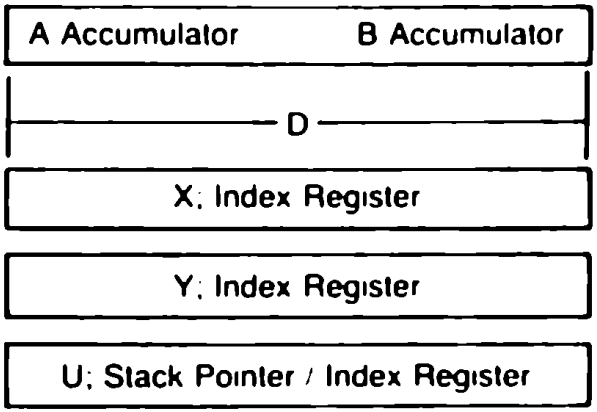
This final Editor command is described here for completeness, though its significance

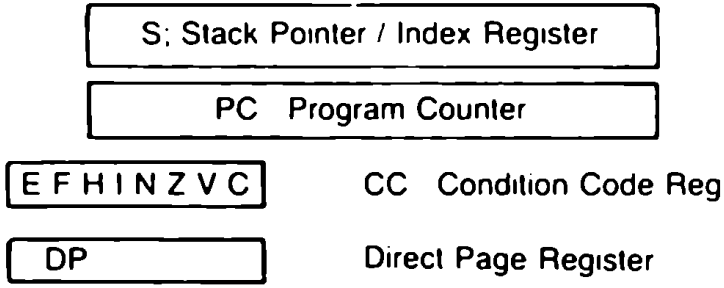
will be more apparent after you have used the Assembler and executed machine code

BREAK T re-initialises the Dragon to text mode, and is necessary when returning to the Editor from a machine code program that has left the Dragon in graphics mode

6809 PROGRAMMING

With its several 2-byte registers, the 6809 can operate internally as a 16-bit processor for many operations The registers are -





A and B are 8-bit general purpose accumulators used for arithmetic and logical operations. They can also be considered together as the 16-bit D accumulator for certain instructions.

X and Y are 16-bit index registers, each used in a large number of indexed mode and other instructions.

U and S are stack pointers, S being inherently used by the 6809 for subroutines and interrupts. Both U and S share the same indexed mode addressing capabilities as X and Y

The program counter is a 16-bit pointer to the next instruction to be executed It can be used in Program Counter Relative addressing not just by branch instructions, but

wherever indexed mode addressing is allowed. This enables the painless implementation of position independent object code.

The condition code Register holds 8 flags which represent the state of the processor and the results of previous instructions. The flags are:

B7 E Entire Flag	Used by the RTI instruction
B6 F FIRQ Mask	Prevents FIRQ Interrupts
B5 H Half-Carry	Used by the DAA instruction
B4 I IRQ Mask	Prevents IRQ Interrupts
B3 N Sign Flag	Set on negative result
B2 Z Zero Flag	Set on Zero result
B1 V Overflow	Set on 2's complement overflow
B0 C Carry	Set on an un-signed overflow and by shift type operations

The 6800 and 6502 microprocessors have a fast, economical Direct Page addressing mode in which a constant zero is used as the top 8-bits of the address. The 6809 extends this concept by using the current contents of the DP register to form the top 8-bits. Hence Direct Page Addressing can be used to access any chosen 256 byte page of memory.

6809 ADDRESSING MODES

Inherent e.g., MUL

No operand field is coded as the operand is implied by the instruction.

Accumulator e.g., CLRA
RORB

Accumulator addressing is used by instructions that operate on a chosen accumulator only, with no reference to memory. No operand field is allowed.

Immediate e.g., LDA #1
CMPX #$7FFF

Put a hash symbol in front of the operand to specify immediate addressing, implying that it is to be used as a value rather than an address. The first example will load the value 1 into A, not the contents of memory location 1.

Extended e.g., LDX >LABEL

For extended addressing, DREAM generates pairs of bytes (High,Low) to fully specify a 16-bit memory address. The '>' symbol is only necessary when it is required to override conditions that would cause Direct addressing to be used.

Direct	e.g.,	STA	FIELD
		INC	<COUNT

Direct addressing produces 1 address byte which forms the low order 8-bits of the memory address. The high order 8-bits are supplied by the Direct Page Register. DREAM will use Direct mode when the operated address is computed to be within the page equal to the current SETDP setting (initially zero). The optional '<' symbol will force Direct mode for this one instruction.

Extended Indirect	e.g.,	JSR	(RADD)

In this mode, the operand is the address of a 16-bit value in memory, which will be used to point to the operand.

Register	e.g.,	TFR	A,DP
		PULS	D,PC

Register addressing refers to the selection of the various 6809 Registers

Indexed

Indexed addressing mode has several options. DREAM will create the correctly coded 'post-byte' following the op-code, for each option.

a) Zero Offset e.g., LDA ,X

This, the fastest indexing mode, uses the specified X, Y, S or U register to point directly to the operand in memory.

b) Post Increment e.g., CLR ,Y+
LDD ,U++

The specified indexing register is incremented by 1 or 2 after addressing the operand in memory. No offset is permitted.

c) Pre Decrement e.g., CMPA ,-X
STB ,--Y

The indexing register is decremented by 1 or 2 before being used to address memory No offset is permitted.

d) Constant Offset e.g. NEG 1,X
ASL -SMALL,Y
STX BIG,U

In this mode a signed offset and the contents of the indexing register are added to form the effective operand address. The register contents are not changed The offset is a signed 5, 8 or 16-bit value. The assembler will usually create the optimum size of offset.

e) Accumulator Offset e.g. CMPX A,U
LEAY D,X

A signed value in the A, B or D accumulator is used as the offset Thus the offset can be calculated at run time.

f) Indexed Indirect e.g., STD (,Y++)
ADDB (4,S)

All the indexing modes excepting auto increment / decrement by 1, may have an additional level of indirection specified The indexing mode is calculated first, and the 2-bytes at the memory thus addressed are used as the effective address of the operand

Relative Addressing (Branching)

The byte(s) following any branch instruction op-code are treated as a signed two's complement offset which is added to the Program Counter to obtain the address of the next instruction Offsets can be 1-byte (short) or 2-bytes (long). The assembler computes the offset — the programmer only has to code the memory address (explicitly or symbolically), and request a short or long offset by selection of the op-code mnemonic

e.g.	BRA	HERE	(short)
	LBEQ	THERE	(long)

Program Counter Relative e.g., LEAY LABEL,PCR

The Program Counter can be used as a pointer register with an 8 or 16-bit signed offset. This addressing mode allows machine code programs to be produced which can execute without alteration, anywhere in memory. DREAM will select the optimum offset size, or it can be specified by the programmer

A level of indirection can be appplied additionally

e.g.,ADDD (POINT,PCR)

Writing Assembler Source Code

Using DREAM, you can code one Assembler source statement per line These statements consist of four fields — label, op-code, operand and comments The op-code field is mandatory, label and comments are optional, an operand is required by some op-codes. The fields are separated by one or more spaces

```
e.g., LOOP    LDA   DATA  . comment Instruction with a label
              BEQ   LOOP  . comment Instruction with no label
              RTS         . comment Instruction with op-code only
```

When present, the label field must start in column 1 and have a maximum length of 6 characters. All characters in the label are significant and are used when matching labels coded in operand fields. Labels must only consist of alphabetic and numeric characters, and must start with an alphabetic There is one exception: one label can start with the @ sign and is used to tell DREAM the first statement to execute when you

come to run the resultant machine code program. It is advisable not to use the single letters A, B, D, X, Y, U or S nor CC, DP or PC as labels as these are normally used to refer to the internal registers of the 6809.

Examples of valid labels

TAG
@START
PART5

Examples of invalid labels

3RD	must not start with a numeric
SECTION	too long
IN-OUT	must not contain any special characters

The Op-code Field

At least one space must separate the label from the op-code field. If no label is present, then there must be a space in column 1. The op-code field can contain any of the standard Motorola mnemonics for 6809 instructions or Assembler directives as listed in Appendix C.

The Operand Field

One space or more must separate the op-code field from the operand. Not all Assembler statements require an operand, but when present it is used to represent an address or a constant or one or more registers on which the instruction will operate.

Within the operand field, hexadecimal numbers are preceded by a dollar sign ($). Numbers not preceded by $ are assumed to be decimal.

An address operand, or an offset for indexed addressing, can be specified as a decimal or hexadecimal value, or as a symbolic label which must be defined in the label field of one source statement.

Examples

LDA $1234	Absolute hex. address
LDA 1Ø24	Absolute decimal address
LDA DATA	Symbolic address
LDA LEN,X	Symbolic offset
LDA 5,X	Absolute offset

An asterisk in the operand field implies the value of the Assembler's program counter at the start of the current statement.

Labels and absolute values can be combined using '+' and '-'. DREAM will do the necessary arithmetic at assembly time.

Examples

```
LEAY    TAG+5,PCR
LDA     END-START+$2C
BRA     *+8
```

Immediate addressing mode is signified by the hash sign (#).

```
LDA     #'+         immediate character
LDB     #$2F        immediate hex
CMPA    #123        immediate decimal
LDX     #TAG        immediate symbolic
```

Controlling the size of Offsets

In the absence of any '<' or '>' symbol, DREAM always selects the optimum offset size for absolute offsets for indexed instructions. With symbolic offsets, DREAM can select the optimum size, but may have to execute several extra assembly passes to fully resolve the program. You can avoid this by telling DREAM the offset size to use. The

symbol '<' selects an 8-bit offset, and '>' selects a 16-bit offset. You cannot request a 5-bit symbolic offset.

DREAM will flag an error if an 8-bit offset is requested where a 16-bit is necessary.

Examples

```
LDA     OFFSET,X        size selected by Assembler
LDA     <OFFSET,X       8-bit offset requested
LDA     >OFFSET,X       16-bit offset requested
JMP     (<TABLE,PCR)    note the sequence (then <
LDA     >1,X            force a long offset
```

Registers

Standard Motorola mnemonics are accepted for specifying registers, e.g.:

```
PSHS    D,X
PULU    CC,Y,PC
TFR     A,DP
LEAX    B,S
LDB     ,U
```

The Comments Field

Two or more spaces and a semi-colon should precede the optional comments field, which is completely free-form

ASSEMBLER DIRECTIVES

FCB/FCC

FCB and FCC are used to format data bytes. The data can be specified as decimal, hexadecimal, character or symbolic.

DREAM recognises both the FCC and FCB recommended Motorola directives, however they are handled identically. This gives the advantage of allowing decimal or hex, bytes to be coded together with character strings in the operand field.

Either an oblique or a quote can be used to bound character strings — the same symbol must occur both ends.

A comma is used to separate sub-operands.

Examples of FCB (FCC)

```
FCB     $1F         1 hex byte
FCB     123         1 decimal byte
FCB     -5,'A'      1 signed decimal byte followed by one character byte
FCC     /TEXT/,Ø    A character string terminated by a null
FCB     VALUE       The symbol must be defined elsewhere
```

When the value assigned to a byte exceeds 256, the low order byte value only is used e.g.,

```
FCB     TAG
```

will generate the displacement of TAG within its page

The FDB Directive

FDB will format 16-bit values occupying 2 bytes each. Character values are preceded by a single quote. Commas separate sub-operands. Each sub-operand generates 2 bytes.

Examples

FDB	*	star and space
FDB	$1	gives hex 0001
FDB	LABEL	gives 16-bit address
FDB	1,2,3	gives 3 consecutive 16-bit fields

The RMB Directive

RMB is used to 'reserve memory bytes' containing no specific value. The Assembler's program counter is moved on by the number of bytes specified. If the operand of the RMB includes any symbolic reference, DREAM may have to use more than two passes to generate the object program.

Examples

RMB	5	reserve 5 bytes
RMB	LENGTH	skip the number of bytes which LENGTH is resolved as.
RMB	LABEL-*	the next statement will start at the memory location specified by LABEL

The EQU Directive

Equate can be used to assign a value to a symbolic label. The Assembler's program

counter is not affected. If the operand includes a symbol not yet defined, DREAM will execute more than 2 assembly passes.

Examples

```
TAG  EQU  5ØØ       assign the value 5ØØ to TAG
XYZ  EQU  TAG+$66   XYZ will have the value 6Ø2 decimal
VAL  EQU  FLD       VAL has the same value or address as FLD
```

The ORG Directive

ORG is used to set the Assembler's program counter, and hence the logical origin for the following generated code. DREAM initially assumes an ORG value equal to the bottom of the work-space. The label field is optional.

Examples

```
     ORG  $5ØØØ    set p.c. to hex. 5ØØØ
NEW  ORG  *+8       equivalent to RMB 8
```

The PUT Directive

PUT directs the object code from the Assembler to a specific area of RAM. DREAM

assumes an initial PUT to the bottom of the work-space. Normally PUT is preceded by an ORG, but they need not specify the same address if you are going to move the object code before executing it, or if your program uses position independent code throughout. Do not code a label field.

Examples

```
PUT         $68EØ
PUT         *
```

The SETDP Directive

SETDP is used to tell the Assembler the current value of the Direct Page. The Assembler will then generate Direct Page addressing mode where appropriate for all following instructions where the operand is computed to be within the specified page. DREAM assumes an initial SETDP of zero. The operand should be a hex. value in the range $ØØ to $FF.

Example

```
SETDP       $6A
```

ASSEMBLER OPERATION

The Assembler operates in a minimum of two passes of the source code. If you have written a good number of statements, you will notice a pause during pass 1 with just the pass number displayed. A symbol table is built during this pass, occupying memory within the DREAM workspace, just below the text table. This pass takes about 2 seconds per hundred source statements.

The second pass follows automatically, and the results are displayed on the screen. You can freeze the display by pressing BREAK. DREAM will pause itself whenever an error is found. Continue by typing A. Typing B will give a slower scroll (BROWSE mode).

The display shows the following columns:-

1. The ORG'd address of the object code in hexadecimal
2. Up to 5 bytes of object code in hexadecimal
   OR an error message
3. The original source code. This column is continued on the next line if necessary.

Error Messages

When DREAM detects a source code error, it displays ERROR in inverse video in the object code column, followed by a letter indicating the type of error. The codes are listed in Table 1 in Appendix D.

If you specify a PUT directive that attempts to load the object code into ROM memory, the message ROM? is displayed in the object code field.

If the object code attempts to overlay DREAM, or the text table or symbol table, the screen is cleared and the message FULL displayed. Pressing any key will return control to the Editor. Modify the PUT statement (if any) or QUIT from DREAM, allocate more workspace and re-enter DREAM to try again.

The Assembler process normally completes at the end of pass 2, but if you have coded a program involving complex resolving of symbolic labels, then DREAM will automatically go into a third or even four or more passes. If the program cannot be resolved in 8 passes, Assembly terminates with the message 'UR' (un-resolvable). Go back to the Editor and supply more control over offset lengths, or remove forward referencing labels from the operand field of such statements as RMB or EQU. In general a 'UR' error is indicative of confused programming.

Assembler Keyboard Commands

At the end of the last pass, the total number of errors is shown, and DREAM waits for a command. Table 2 in Appendix D lists all the Assembler keyboard commands. To give a command, type BREAK followed by the letter and then press ENTER. If you type a command wrongly, type BREAK again and re-enter it.

You can display any number of further passes by BREAK A or B. Passes occur in cycles on eight, with the first of each cycle being a non-display pass.

BREAK P will output the results to a printer as well. At the end of a line, typing BREAK will pause the printer and display. Type P to continue printing. A or B will continue without print.

Unless you are experienced in Assembler programming, DREAM will probably have flagged some errors in your coding. When assembly is finished, the command BREAK Q will QUIT from the Assembler back to the Editor so you can correct your source code. The cursor will be positioned on the last statement that you edited.

Alternatively, while the Assembler is pausing during a display pass, e.g., when an error is shown, you may decide to return directly to the Editor to correct that statement rather

than wait for the assembly to complete. Pressing Q at this stage will return control to the Editor with the same block of text displayed that was on the Assembler screen

Once you have a clean Assembly, the resultant program can be tested The command BREAK X will transfer control to your object program

BREAK G will enter the DEBUG package "DREAMBUG" if it is installed DREAM tests for the existence of the DEBUG package in memory and returns the message 'NF' if it can not be found. (DREAMBUG assists the testing of machine code programs by providing breakpoint and memory examine and change facilities etc)

Testing the Object Program

If you have not coded any PUT statement, then DREAM will store the object code from the Assembler starting at the bottom of the workspace. e.g., if you reserved space with a statement CLEAR ssss,20000 then the first assembled instruction will start at address 20001 decimal (hex. 4E21). Without an ORG statement, this will also be the logical address of that instruction as displayed on the output from the Assembler

When you pass control to the object code by typing BREAK X at the end of assembly, execution will start at the instruction that had a label starting with the @ symbol If there

is no such label, then execution will start at the address of the bottom of the workspace (20001 etc.). On entry to your program the direct page register is set to zero. DREAM transfers control by issuing a JSR instruction. If your program ends with RTS then control will be returned to the Editor (assuming successful execution)

DREAM automatically generates an RTS instruction at the end of your program, provided your last source statement was an instruction and not an Assembler Directive, but it is not a good idea to rely on this safety feature in your programs.

If your object program sets the Dragon into a graphics mode, then you may need to type BREAK T ENTER when execution has completed, to re-establish text mode for the Editor.

Object code generated by DREAM can also be executed via the Dragon EXEC command or USR function, if you leave DREAM and use the relevant BASIC statements. e.g.,

```
EXEC 20001    or
DEFUSR = 20001 : A = USRØ(Ø)
```

SUPPLEMENTARY INFORMATION

Saving Object Code on Cassette

Object code can be saved on tape and re-loaded by using the CSAVEM and CLOADM commands from BASIC as described in the DRAGON handbook. The code can then be used at a later date without DREAM being installed in memory.

Alternative Positioning of Object Code

You might wish to assemble a routine which you would prefer to have positioned at the top of RAM (say starting at 30001 dec.) so it can be used with a very large BASIC program. Assemble the code with an ORG of 30001 and save the output with CSAVEM. Switch the Dragon off and on again to release DREAM's memory to BASIC. Load the BASIC program and type CLEAR ssss,30000. Load the saved object code using CLOADM with an appropriate offset so it will be positioned at 30001.

Accessing the Text Table

It can be useful to use the DREAM Editor to maintain files of data which can be read by your own machine code programs. DREAM contains a routine which you can access that will extract any line from the text table. An entry point to this routine exists at 2 bytes into DREAM i.e. for DREAM at its default position of 27776, the entry point to the

extract routine is at 27778 decimal (hex. $6C82).

To use the routine, load register X with the start address of a 32 byte area into which the line is to be returned. Load register D with the line number required. (the first line is numbered zero). You **must** set up the direct page register to the top page of the workspace (normally hex. 6B). If D refers to a line outside the current text table range, then a 'plus' condition is returned in the condition code register For a valid line, the condition code will be set to 'minus'

Relocating DREAM

DREAM has been written using position independent code throughout and hence can reside anywhere within the memory map. It dynamically searches for the first complete page of RAM below itself and sets the direct page register to point to it The workspace starts there, working downwards The bottom of the workspace is one higher than the second number you gave on the CLEAR statement

DREAM has to be relocated lower in memory when being used with the Debug package, which is available separately.

DREAM Entry Point

To enter DREAM after loading any other machine code program, type

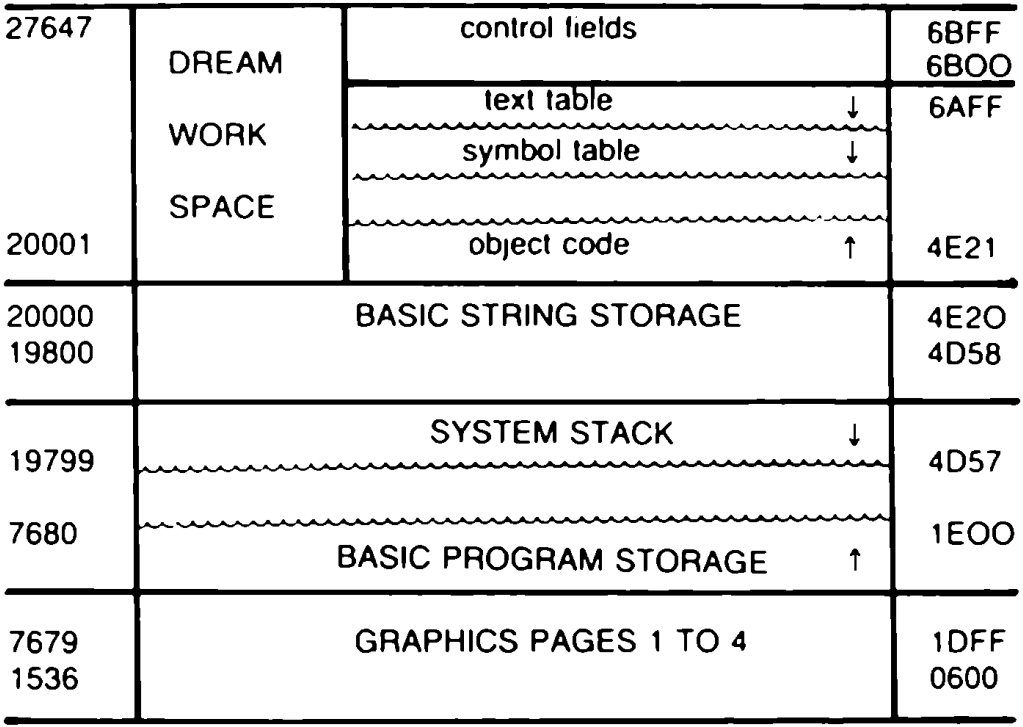
```
EXEC 27776
```

or whatever address DREAM is loaded at.

APPENDIX A

Sample Memory Map with DREAM Installed.

Decimal Address		Hex Address
65535 65280	SYSTEM & I/O AREAS	FFFF FFOO
65279 49152	CARTRIDGE ROM MEMORY	FEFF COOO
49151 32768	BASIC INTERPRETER ROM	BFFF 8000
32767 27776	DREAM	7FFF 6C80
27775 27648	Debug breakpoint table	6C7F 6C00


27647
DREAM
WORK
SPACE
20001
control fields
6BFF
6BOO
text table ↓
6AFF
symbol table ↓
object code ↑
4E21
20000
19800
BASIC STRING STORAGE
4E2O
4D58
SYSTEM STACK ↓
19799
4D57
7680
1EOO
BASIC PROGRAM STORAGE ↑
7679
1536
GRAPHICS PAGES 1 TO 4
1DFF
0600

1535 1024	TEXT SCREEN MEMORY	05FF 0400
1023 0	SYSTEM USE	03FF 0000

The map shows the typical usage of memory within the Dragon with DREAM installed, assuming an initial CLEAR 200,20000 and assuming the default 4 high resolution graphics pages. The arrows show the direction in which expandable areas will grow.

APPENDIX B — EDITOR OPERATION

Cursor Positioning, Scrolling, and Editing

→	cursor right
←	cursor left
SHIFT →	insert character
SHIFT ←	delete character
BREAK, BREAK	cursor to column 1
CLEAR	erase rest of current line
ENTER	cursor to next line
↑	cursor up
↓	cursor down
SHIFT ↑	scroll back
SHIFT ↓	scroll forwards
SHIFT @	repeat last FIND or CHANGE command
SHIFT SPACE	tab to pre-defined columns

Type BREAK then the command

A	execute assembler
C/s1/s2/	change string1 to string2
C/s1/s2/A	ditto — all occurrences
D	delete 1 line
Dn	delete n lines
DM	delete marked block
E	end — display last 8 lines
F/s1/	find string1
H	home — display first 16 lines
I	insert 1 line
In	insert n lines
L name	load DREAM file from tape
M	mark first line of a block
N	mark end line of a block
P	print rest of text file
Pn	print next n lines

PM	print marked block
Q	quit — return to BASIC
R	replicate marked block
S name	save complete text table on tape
Sn name	save next n lines
SM name	save marked block
T	re-establish text mode
U	un-mark marked block
V	recover line as before editing

APPENDIX C1 — INSTRUCTION OP-CODES
(Excluding Branch Instructions)

m.b. = memory byte
16-bit m.v. refers to the contents of 2 consecutive memory bytes

ABX	Unsigned add of B to X
ADCA,ADCB	Add 1 m.b. plus carry flag to A or B
ADDA,ADDB	Add 1 m.b. to A or B accumulator
ADDD	Add 16-bit m.v. to D accumulator
ANDA,ANDB	Logical AND of 1 m.b. with A or B
ANDCC	Logical AND of immediate byte with CC
ASL,ASLA,ASLB	Arithmetic left shift of 1 m.b. or A or B
ASR,ASRA,ASRB	Arithmetic right shift of 1 m.b. or A or B
BITA,BITB	Set CC as for ANDA,ANDB but leave A or B unchanged
CLR,CLRA,CLRB	Clear 1 m.b. or A or B to zero
CMPA,CMPB	Compare A or B with 1 m.b.
CMPD	Compare D with 16-bit m.v.
CMPS,CMPU	Compare stack pointer with memory
CMPX,CMPY	Compare index register with memory

COM,COMA,COMB	Invert all bits in a m.b. or A or B
CWAI	AND immed. byte with CC and wait for interrupt
DAA	Decimal adjust A accum.
DEC,DECA,DECB	Decrement 1 m.b. or A or B by 1
EORA,EORB	Exclusive-OR A or B with 1 m.b.
EXG	Exchange contents of any 2 like-size regs.
INC,INCA,INCB	Increment 1 m.b. or A or B by 1
JMP	Jump to effective address
JSR	Jump to subroutine
LDA,LDB	Load A or B from 1 m.b.
LDD	Load 16-bit m.v. into D
LDS,LDU	Load 2 m.b.'s into stack pointer
LDX,LDY	Load 2 m.b.'s into index register
LEAS,LEAU	Load effective address into stack pointer
LEAX,LEAY	Load effective address into index register
LSL,LSLA,LSLB	Logical left shift of 1 m.b. or A or B
LSR,LSRA,LSRB	Logical right shift of 1 m.b. or A or B
MUL	Unsigned multiply D = A times B
NEG,NEGA,NEGB	Negate 1 m.b. or A or B
NOP	Single byte no-operation
ORA,ORB	Logical OR of 1 m.b. with A or B

ORCC	Logical OR of immediate byte with CC
PSHS,PSHU	Push any subset or regs onto S or U stack
PULS,PULU	Pull any subset of regs from S or U stack
ROL,ROLA,ROLB	Rotate left 1 m.b. or A or B with carry
ROR,RORA,RORB	Rotate right 1 m.b. or A or B with carry
RTI	Return from Interrupt
RTS	Return from Subroutine
SBCA,SBCB	Subtract 1 m.b. and carry flag from A or B
SEX	Extend the sign bit of B throughout A
STA,STB	Store A or B into 1 m.b.
STD	Store D into 2 m.b.'s
STS,STU	Store stack pointer in memory
STX,STY	Store index register in memory
SUBA,SUBB	Subtract 1 m.b. from A or B
SUBD	Subtract 16-bit m.v. from D
SW1,SW12,SW13	Software interrupts
SYNC	Synchronise with interrupt
TFR	Transfer contents of any register to any other of like size
TST,TSTA,TSTB	Test the value of 1 m.b. or A or B

APPENDIX C2 — BRANCH INSTRUCTIONS

Op-codes starting with L are 'long' branches producing a 16-bit signed offset

BCC,LBCC	Branch if carry clear
BCS,LBCS	Branch if carry set
BEQ,LBEQ	Branch if equal
BGE,LBGE	Branch if greater or equal (signed)
BGT,LBGT	Branch if greater (signed)
BHI,LBHI	Branch if high (un-signed)
BHS,LBHS	Branch if high or same (un-signed)
BLE,LBLE	Branch if less or equal (signed)
BLO,LBLO	Branch if lower (un-signed)
BLS,LBLS	Branch if lower or same (un-signed)
BLT,LBLT	Branch if less than (signed)
BMI,LBMI	Branch if minus
BNE,LBNE	Branch if not equal
BPL,LBPL	Branch if plus
BRA,LBRA	Branch always
BRN,LBRN	Branch never (no-operation)
BSR,LBSR	Branch to subroutine (un-conditional)

BVC,LBVC	Branch if overflow clear
BVS,LBVS	Branch if overflow set

APPENDIX C3 — ASSEMBLER DIRECTIVES

EQU	Equate
FCB/FCC	Format bytes and concatenated characters
FDB	Format double bytes
ORG	Set logical origin
PUT	Set physical target address
RMB	Reserve memory bytes
SETDP	Declare Direct Page

APPENDIX D

TABLE 1 — Assembler Error Codes

B	8 bit offset requested where a 16 bit is required
C	Invalid register combination on EXG or TFR
D	Duplicate defined label
F	Short branch used where a long branch is required

I	Invalid indexing
L	Invalid label field
O	Invalid Op-code
R	Invalid register
S	Syntax error in operand field
U	Undefined label found in operand
X	Invalid constant

TABLE 2 — Assembler Keyboard Commands (Precede by BREAK)

A	Assemble again (a further pass)
B	Assemble at browse speed
G	Enter Debug package
P	Print assembled program
Q	Quit — return to Editor
X	Execute object program

APPENDIX E

System Error Messages

Message	Meaning
NO TEXT	A reply of 'Y' was given to OLD TEXT? on the title screen, but a valid text table does not exist in RAM. Reply N to start a new text table.
FULL	The DREAM workspace is full, or the output from the Assembler is attempting to overlay DREAM or its internal tables. Press enter to return to the Editor, then delete some lines or correct any PUT/ORG statements.
NF	Attempting to enter the Debug package but the Debug identification pattern was not found in the expected position.
UR	The Assembler has done 8 passes of the source code but has not been able to resolve the program.

Return to the Editor and simplify the usage of forward symbolic references. The problem can also be caused by coding a program that attempts to generate different object code for the same area of memory, by bad use of the ORG or PUT statements

ERROR X — The Assembler has found an error in the source program.

Look in Appendix D for an explanation of X

ROM? — The object code from the Assembler is attempting to overlay an area of ROM.

Return to the Editor and correct the PUT or ORG statements.

APPENDIX F—SAMPLE PROGRAMS

```
4000              *
4000              *This program outputs the letter A
4000              *to all positions of the screen
4000              *
4000              *Put the address of the start of VDU
4000              *RAM into the X index register
4000              *
4000 8E0400               LDX         #S04ØØ
4003              *
4003              *Put the letter A into the A register
4003              *
4003 8641                 LDA         #'a
4005              *
4005              *Store the contents of A into the
4005              *memory byte indexed by X,
4005              *then increment X by 1 to point to the
4005              *next VDU byte
4005              *
4005 A780         LOOP    STA         ,X+
```

```
4007                *
4007                *Have we reached the end of the screen?
4007                *—If not go back to LOOP
4007                *
4007 8C0600                 CMPX        #$Ø6ØØ
400A 25F9                   BLO         LOOP
400C                *
400C                *All done—now wait for the key to be
400C                *typed (JSR $8006 is like INKEY$)
400C                *
400C BD8006         WAIT    JSR         $8006       Any key typed?
400F 27FB                   BEQ         WAIT        —no (value is Ø)
4011                *
4011                *A key is pressed, return to DREAM
4011                *
4011 39                     RTS
4012                *
```

```
4000                *
4000                *That program executed too quickly to
4000                *see what was happening—let's slow
```

```
4000                    *it down a bit
4000                    *
4000 8E0400                     LDX     #$Ø4ØØ
4003 8641                       LDA     #'A
4005 A780               LOOP    STA     ,X+
4007                    *
4007                    *Now load the value 5000 into Y and
4007                    *count down to zero between each 'POKE'
4007                    *to the screen
4007                    *
4007 108E1388                   LDY     #5000
400B 313F               DELAY   LEAY    -1,Y            decrement Y
400D 26FC                       BNE     DELAY           repeat until zero
400F 8C0600                     CMPX    #$Ø6ØØ
4012 25FI                       BLO     LOOP
4014 BD8006             WAIT    JSR     $8006           Any key typed?
4017 27FB                       BEQ     WAIT            —no (value is Ø)
4019                    *
4019 39                         RTS
401A                    *
4000                    *
```

```
4000              *Now let s output the alphabet
4000              *repeatedly
4000              *
4000 8E0400               LDX       #$0400
4003 8641         RESET   LDA       # A
4005 A780         LOOP    STA       ,X+
4007              *
4007              *Now increment the A register and test
4007              *when it exceeds the letter z
4007              *
4007 4C                   INCA                        ;increment A reg
4008 815A                 CMPA      # Z
400A 22F7                 BHI       RESET             go and reset to A
400C              *
400C 108E03E8             LDY       #1000             (a bit faster)
4010 313F         DELAY   LEAY      -1,Y
4012 26FC                 BNE       DELAY
4014              *
4014 8C0600               CMPX      #$0600
4017 25EC                 BLO       LOOP
4019 BD8006       WAI     JSR       $8006
```

```
401C 27FB              BEQ       WAIT
401E                   *
401E 39                RTS
401F                   *
```

```
4000                   *
4000                   *This version uses the Dragon's 50 hz
4000                   *clock to control the timing
4000                   *
4000                   *First, disable the interrupt so we can
4000                   *use it
4000                   *
4000 1A10              ORCC      #$10        set the IRQ mask bit
4002                   *
4002 8E0400            LDX       #$0400
4005 8641      RESET   LDA       #'A
4007 A780      LOOP    STA       ,X+
4009                   *
4009 4C                INCA
400A 815A              CMPA      #'Z
400C 22F7              BHI       RESET
400E                   *
```

```
400E                *Now wait for the interrupt—as it
400E                *has been masked, execution will
400E                *continue with the next instruction
400E                *after SYNC
400E                *
400E 13                     SYNC                        ;wait for IRQ
400F                *
400F                *Now clear the source of the interrupt
400F                *—this is done by 'reading' one of
400F                *the special I/O locations
400F                *
400F F6FF02                 LDB       $FF02             clear irpt source
4012                *
4012 8C0600                 CMPX      #$0600
4015 25FO                   BLO       LOOP
4017 BD8006         WAIT    JSR       $8006
401A 27FB                   BEQ       WAIT
401C                *
401C 1CEF                   ANDCC     #$EF              Enable IRQ again
401E 39                     RTS
401F                *
```

APPENDIX F — Cont'd

Sample Output from the Assembler

N B This is not a meaningful program — just a demonstration of using various instructions and directives

```
4000                *
0200                        ORG     $200
0200                        PUT     $3800
0200                *The PUT statement directs the object
0200                *code to a specific area of RAM
0200                *PUT must always be preceded by an ORG
0200                *and can specify a different address
0200                *if position independent code is used
0200                *
0200                *
0200                *The label @ indicates the start
0200                *point for program execution
0200                *
0200 4F             @       CLRA
0201                *
```

```
0201                    *Immediate operands
0201 8B0A               ADDA    #10         decimal
0203 108E1F2C           LDY     #$1F2C      hex
0207 C167               CMPB    #'g         character
0209 8E04B0             LDX     #BIG        equated value
020C CEFB2E             LDU     #-1234      16 bit negative
020F                    *Indexing
020F A784               STA     ,X
0211 E6A0               LDB     ,Y+
0213 01AF81             STY     ,X++
0216 6DC2               TST     ,-U
0218 EDE3               STD     ,--S
021A 684C               ASL     12,U        5 bit offset
021C 6388EC             COM     <-SMALL,X   8 bit offset
021F 6FA904B0           CLR     >BIG,Y      16 bit
0223 6C86               INC     A,X         register offset
0225 6FAB               CLR     D,Y
0227                    *PC relative
0227 308C2E             LEAX    <NEAR,PCR   short
022A E38D0412           ADDD    >FAR,PCR    long
```

```
022E                    *Indirect modes
022E A691                       LDA     (,X++)      indexed
0230 6D9FC31B                   TST     (ANAME)     extended
0234 AD9D00E3                   JSR     (>ANAME,PCR) PC relative
0238 309C20                     LEAX    (<TAG,PCR)
023B                    *Push/pull, exchange & transfer
023B 3440                       PSHS    U
023D 3652                       PSHU    A,X,S
023F 35FF                       PULS    D,DP,CC,X,Y,U,PC
0241 1F89                       TFR     A,B
0243 1E31                       EXG     U,X
0245                    *
0245                    *Using Direct Page mode
0245 8603                       LDA     #$03
0247 1F8B                       TFR     A,DP
0249                            SETDP   $03         tell the assembler
```

```
0249 D704                STB     STORE
024B 9E00                LDX     VALUE       etc
024D OF34                CLR     <OTHER      force direct mode
024F F9025B              ADCB    >TAG        force extended mode
0252            *Branch etc
0252 2004                BRA     NEAR
0254 012603E8            LBNE    FAR
0258 BD8006      NEAR    JSR     $8006       absolute
025B             *
025B             *Examples of constants & field defines
025B             *
025B 0258        TAG     FDB     NEAR
0300                     ORG     $0300
0300                     PUT     $3900
0300 C350        VALUE   FDB     50000       2 byte decimal
0302 FDA8        NEG     FDB     -600        negative
0304 96          STORE   FCB     150         1 byte decimal
0305             WORK    RMB     12          reserve 12 bytes
0311 5065746572  NAME    FCC     /PETER/     character string
0316 0D0A        CRLF    FCB     $D,$A       hexadecimal bytes
0318 1C2B09      MIX     FCB     $1C,'+',9   hex, char, dec
```

```
031B 0311      ANAME   FDB     NAME      address constant
031D           *
031D           *examples of equated values
031D           *
031D 800F      CONOUT  EQU     $800F     hex. address
031D 0310      END     EQU     WORK+11
031D 000C      LWORK   EQU     NAME- WORK
031D 04BO      BIG     EQU     1200   decimal
031D 0014      SMALL   EQU     20
031D 0640      FAR     EQU     NEAR+1000
031D 1234      OTHER   EQU     $1234
031D           *
```


WARRANTY STATEMENT

Dragon Data products sold by authorised dealers are offered under the provisions of the Supply of Goods (Implied Terms) Act 1973. In order to provide a satisfactory service to our customers, Dragon Data Ltd. warrants the following:

I. All faulty components due to defective manufacture will be replaced free of charge for a period of 12 months from the original date of purchase.

II. All labour and/or services will be provided free of charge to repair your Dragon Data product which fails in its specified performance due to manufacturing defects for a period of 12 months from original date of purchase.

NB (a) The guarantee is restricted to the original purchaser.

(b) Claims will not be accepted if any unauthorised modification is made to the product or if the serial number or guarantee labels have been removed or defaced.

(c) Dragon Data's liability is limited to the cost of repair or replacement (at Dragon Data's discretion) of the defective product.

This warranty is offered as an extra benefit and does not affect customers' statutory rights.