ALLDREAM

DRAGON 6809 EDITOR-ASSEMBLER AND MONITOR

120001

CONTENTS

● ALLDREAM

Alldream is a software package for the Dragon 32 Microcomputer, comprising an Editor Assembler (DREAM) and Monitor (DREAMBUG).

DREAM is designed to maintain files of general purpose text, and especially for the preparation, maintenance and conversion to object code, of Assembler source programs for the 6809 micro-processor. DREAM can produce complete machine code programs, and routines which can be called from BASIC programs via the USR function.

DREAMBUG is a full feature machine code testing tool and disassembler.

● LOADING THE PACKAGE

As ALLDREAM is designed for easy use together with BASIC, the cartridge is not self-starting. Switch off the Dragon, insert the cartridge, and switch on again. Before executing ALLDREAM, you must tell the machine to reserve some memory for the package's work areas. ALLDREAM uses RAM just below the BASIC ROM, working downwards from 32639. You must control the lowest address available to ALLDREAM, by using the CLEAR statement. A good starting point might be:

```
CLEAR 200,20000
```

This will reserve memory from 20001 to 32767 for ALLDREAM
sufficient for Assembler programs up to about 600 typical length lines, or about 350 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 ALLDREAM installed.

Now you can execute DREAM by typing:

```
EXEC 49152
```

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 Assembler 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 unwaI}ted 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 1 or SHIFT and r. 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 \cD 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. Hen_e 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 for
C/string1/string2/ A	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 and lost.

Repeating a FIND or CHANGE

Pressing SHIFT and@togetherwill repeat the last FH'J.D 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 oflines 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 to 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 ENDMARK 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 the press ENTER.

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

S25 filename

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.,

LP

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 fbund, a flashing yellow blob at the right end of the command line indicates that loading is proceeding.

Should any loading error occur, an inverse 'E' will appear to the left of the flashing blob. In the eveat of this happening, press the reset button and reload the file.

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 PM to print the next 'n' lines or
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 '0' 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 3_O is initialised to 1, and locations 331, 332 to the values for CR (hex 0DJ and LF (hex 0AJ. 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 POKEs.

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 predefined 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 Accumulator	B Accumulator
D	
X : Index Register	
Y : Index Register	
U : Stack Pointer / Index Register	
S : Stack Pointer / Index Register	
PC : Program Counter	
E F H I N Z V C	CC : Condition Code Reg
DP	Direct Page Register

A and B are 8-bit general purpose accumulators used for arithmetic and logical operations. They can also be considered together as the 16bit 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
Bl V Overflow	Set on 2's complement overflow
BO 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 use.d 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 over-ride 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 8bits of the memory address. The high order 8-bits are supplied by the Direct Page Register. DREAM will use Direct mode when the operand 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 I,X -
ASL STX SMALL,Y
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.

D 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 I-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 opcode mnemonic.

e.g., BRA LBEQHERE (short)
THERE (long)

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

The Program Counter can be used as a pointer register with an 8 or 16bit 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 applied 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 BEQ   Instruction with a label
          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 1024        Absolute decimal address
LDA DATA        Symbolic address Symbolic
LDA LEN,X       offset Absolute offset
LDA 5,X
```

An asterisk in the operand field implies the value *of* the Assembler's program counter at the star't *of* the current statement.

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

Examples

```
LE  AY          TAG+5,PCR END-
LDA             ST ART + $2C *+8
BRA
```

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

```
LDA     #'+     immediate character
LDB     #$2F    immediate hex
CMP A   # 123   limmediate decimal
LDX     #TAG    [mmediate symbolic
                I
```

Controlling the Size of Offsets

In the absence *of* any '<' *or*\ '>' symbol, DREAM always selects the optimum offset size *for* absolmte 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 8-
LDA       <OFFSET,X       bit offset requested 16-bit
LDA       >OFFSET,X       offset requested note the
JMP LDA   «TABLE,PCR)     sequence (then < force a long
          >l,X            offset
```

Registers

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

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

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     $lF 123         1 hex byte
FCB     -5,'A'          1 decimal byte
FCB                     1 signed decimal byte followed by one
                        character byte
FCC     /TEXT/,0        A character string terminated by a null The
FCB     VALUE           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 LABEL        gives hex. 0001
FDB     1,2,3           gives 16-bit address
FDB                     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 2 passes to generate the object program.

Examples

```
RMB     5 LENGTH        reserve 5 bytes
RMB                     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 symbol 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     500         assign the value 500 to TAG
XYZ     EQU     TAG+$66     XYZ will have the value 602 decimal V AL
VAL     EQU     FLD         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     $5000       set p.c. to hex. 5000
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 workspace. 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     $68E0
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 $00 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 programm_ng.

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 nondisplay 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". Dream tests for the existence of the DEBUG package in memory and returns the message 'NF' if it can not be found. mREAMBUG 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. Ifthere 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 = USR0(0)
```

● SUPPLEMENTARY INFORMATION

Saving Object Code on Cassette

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

Alternate 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 32001 dec.) so it can be used with a very large BASIC program. Assemble the code with an ORG of 32001, but with a PUT to the bottom of the workspace, and save the object code from the PUT area, using CSA VEM. Switch the Dragon off and on again to release DREAM's memory to BASIC. Load the saved object code using CLOADM with an appropriate offset so it will be positioned at 32001.

Smaller routines can be assembled directly into the 128 byte area at the top of RAM (hex. 7F80 to 7FFF) which ALLDREAM does not use. Start the program with

```
ORG   $7F80
PUT   *
```

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., 49154 decimal (hex. CO02).

To use the routine, load register X with the 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 contain hex 7E - the direct page used by ALLDREAM. If the D register 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'.

APPENDIX A

Sample Memory Map with ALLDREAM Installed:

Decimal Address			Hex Address
65535 – 65280	SYSTEM & I/O AREAS		FFFF-FFOO
65279 – 57344	UNUSED		FEFF-EOOO
57343 - 49152	ALLDREAM CARTRAGE		DFFF-COOO
49151 - 32768	BASIC INTERPRITER ROM		BFFF-8000
32767 - 32640	UNUSED		7FFF-7F80
32639 - 32512	Debug breakpoint and trace history tables.		7F7F-7E00
32511 - 20001	DREAM WORK SPACE	control fields	7EFF-7E00
		text table (grows down)	7DFF-4E21
		symbol table (grows down)	
		unused	
		object code (grows up)	
20000 -19800	BASIC STRING STORAGE		4E20-4D58
19799 - 7680	SYSTEM STACK (grows down)		4D57-1E00
	FREE SPACE		
	BASIC PROGRAM STORAGE (grows up)		
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 usage of memory with the ALLDREAM cartridge installed, assuming an initial CLEAR 200,20000, and assuming the default 4 high resolution graphics pages. The arrows show the direction in which expandable area will grow.

APPENDIX B - EDITOR OPERATION

Cursor Positioning, Scrolling, and Editing

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

COMMANDS

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 Cl - 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 I m.b. plus carry flag to A or B
ADDA,ADDB	Add I 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
ASLA,ASLB	Arithmetic left shift of 1 m.b. or A or B
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
OR,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 of 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
SWI,SWI2,SWI3	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

TST,TSTA,TSTB

APPENDIX C2 - BRANCH INSTRUCTIONS

Gp-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.

```
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     #$0400
4003                    *
4003                    * Put the letter A into the A register
4003                    *
4003 8641                       LDA     #'A
4005                    *
4005                    *Store the contents of A into the *memory byte
4005                    indexed by X,
4005                    *then increment X by 1 to point to the *next
4005                    VDU byte
4005                    *
4005                    LOOP
4005 A 780                      STA     ,X+
4007                    *
4007                    * Ha ve we reached the end of the screen?
4007                    *-If not go back to LOOP
4007                    *
4007    8C0600                  CMPX    #$0600
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 BEQ $8006           Any key typed? -no
400F 27FB                               WAIT            (value is 0)
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     #$0400
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 -l,Y
400B 313F       DELAY   LEAY    DELAY   decrement Y repeat
400D 26FC               BNE     #$0600  until zero
400F 8C0600             CMPX    LOOP
4012 25FI               BLO     $8006
4014 BD8006 WAIT        JSR     WAIT
4017 27FB               BEQ             Any key typed? -no
4019                                    (value is 0)
4019 39                 RTS
401A            *
4000            *
4000            *Now let's output the alphabet
4000            *repeatedly
4000            *
4000 8E0400             LDX     #$0400
4003 8641       RESET   LDA     #'A ,X+
4005 A 780      LOOP    STA
4007
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 RESET
400A 22F7               BHI             go and reset to A
400C  400C 108E03E8
4010 313F       DELAY   LDY     # 1 000 -      (a bit faster)
401226FC                LE  AY  l,Y
4014                    BNE     DELAY
40148C0600 401,725EC
4019 BD8006 WAIT        CMPX    #$0600
401C 27FB               BLO     LOOP
401E                    JSR     $8006
401E 39                 BEQ     WAIT
401F
                        RTS
```

```
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 1AlO                ORCC    #$10        set IRQ mask bit
4002            *
4002 8E0400              LDX     #$0400
4005 8641 4007   RESET   LDA     #'A
A 780 4009       LOOP    STA     ,X+
4009 4C 400A     *
815A 400C                INCA
22F7
400E
400E                *Now wait for the interrupt-as it
400E                *has been masked, execution will
400E                *continue with the next instruction *after
400E                SYNC
400E 13             *
400F                     SYNC                ;wait for IRQ
400F                *
400F                * Now clear the source of the interrupt *-
400F                this is done by 'reading' one of *the
400F                special I/O locations
400F F6FF02         *
4012                     LDB     $FF02       clear irpt source
4012 8C0600      *                           CMPA    #'Z
4015 25FO                                    BRI     RESET
4017 BD8006              CMPX    #$0600
401A 27FB                BLO     LOOP
401C 401C
1CEF 401E 39     WAIT    JSR     $8006
401F                     BEQ     WAIT
                 *                           Enable IRQ again
                         ANDCC   #$EF
                         RTS
0245 0245        * Using Direct Page mode LDA
8603 0247                #$03 TFR A,DP
1F8B 0249                SETDP $03 STB
0249 D704                STORE               tell assembler
```

```
024B 9EOO                   LDX CLR VALUE      .....etc.
024D OF34                   ADCB    <OTHER     force direct mode force
024F F9025B       * Branch etc      >TAG       extended mode
0252                        BRA
0252 2004                   LBNE    NEAR
0254 012603E8               JSR     FAR $8006
0258 BD8006       NEAR                         absolute
025B *
025B              * Examples of constants & field defines.
025B *
025B 0258         TAG       FDB    NEAR
0300 ORG _OO PUT                   $0300
0300 C350         VALUE     FDB    $3900
0302 FDA8         NEG       FDB    50000      2 byte decimal
0304 96           STORE     FCB    -600       negative
0305              WORK      RMB    150 12     1 byte decimal
0311 5065746572 NAME FCC           /Peter/    reserve 12 bytes
0316 ODOA         CRLF      FCB    $D,$A      character string
03181C2B09 MIX FCB                 $IC,'+',9  hexadecimal bytes
031B 0311         ANAME     FDB    NAME       hex, char, dec.
031D *                                        address constant
031D              *examples of equated values
031D *
031D 800F         CONOUT EQU
031D 0310         END       EQU    $800F      hex. address
031D OOOC         LWORK EQU        WORK + 11
031D 04BO         BIG       EQU    NAME- WORK 1200
031D 0014         SMALL     EQU    decimal 20
031D 0640         FAR       EQU    NEAR+1O00
031D 1234         OTHER     EQU    $1234
031D *
```

● DREAMBUG

DREAMBUG is a comprehensive machine code test tool for the Dragon 32 microcomputer, and is supplied as part of the ALLDR1<;AM cartridge.

With DREAMBUG you can:

Disassemble any 6809 machine code.

Obtain a formatted dump of any part of memory, in hexadecimal and Ascii.

Dynamically switch between dump mode and disassembly.

Single step through machine code programs.

Dynamically trace the execution of machine code showing all registers after each instruction, and the instructions in hexadecimal and disassembled form.

Set up a "stop condition", e.g. to test when an area of memory is being corrupted, and execute machine code until the condition is met.

Display the last 8 instructions that were executed up to the stop condition, or from any trace.

Maintain a table of up to ten "break-points" for insertion in code to be tested, and execute the code showing the registers and next instruction when a break-point is reached.

Examine and change memory and registers.

Optionally output the results to a printer.

Do hexadecimal arithmetic and decimal to hexadecimal converSIOn.

● ENTERING DREAMBUG

You can enter DREAMBUG either directly from BASIC or from DREAM after assembling a program.

The entry point from BASIC is at 54148 decimal (hex. D384). Enter:

EXEC 54148

From DREAM type:

BREAK G at the end of the assembly process

DREAMBUG will display a title and. the prompt symbol ">".

● DREAMBUG COMMANDS

All commands to DREAMBUG consist of a single letter followed by up to three parameters. Example:

A 20001,3

The single letter identifies the command. One or more spaces can separate the letter from the parameters (if any) which are separated by commas, and can be decimal or hexadecimal values. For some commands, parametl;rs can also be specified as symbolic labels referring to instructions in a program you have just assembled, provided you entered DREAMBUG from DREAM.

Parameters written as numbers are assumed to be decimal. Hexadecimal values must be preceded by a dollar sign ($). Symbolic labe,ls start with a letter or the character "@". Example:

123	decimal
$123	hexadecimal
TAG5	symbolic label

Commands are typed immediately after the ">" prompt symbol. A command line cannot exceed 20 bytes and is completed by typing ENTER. While typing a command, corrections can be made by typing <- to backspace and erase for re-typing.

Commands which are not recognised, or which are syntactically or logically incorrect, are flagged by"??".

Appendix G lists all the commands.

● BREAK POINTS

You can maintain and use a table of up to 10 break points via the B, C, I and L commands. "B" adds a break point entry to the table, "c" clears

an entry, "L" lists the table and "I" inserts an SWI (software interrupt) at each break-point position, into the program to be tested, and then executes the target program.

When an SWI instruction is executed, control returns to DREAMBUG which will display the contents of the CPU registers, and the next instruction to be executed. You can then use other DREAMBUG commands such as examine and alter memory and registers as desired, before continuing execution.

Whenever control returns to DREAMBUG, either by reaching an SWI or by. exiting from the target program via an RTS instruction etc., DREAMBUG replaces all the inserted SWI break-points with the original instructions, so any non break-point execution can occur without interference.

Break-points cannot be inserted into ROM.

● ADDING AND DELETING BREAK POINTS

To add a break-point, type:

Ba

Where "a" signifies the address for the break-point. Example:

B $4E26 B
5892
B LABEL

.Note that this only adds an entry to the table. Break-points are not inserted into the target program until the "I" command is given.

To clear a break-point type:

Ca

e.g. C $4E26 C
5892 C
LABEL

Again, this only removes an entry from the table and does not directly affect the machine code program.

To list the current break-point table, use the "L" command, which has no parameters:

L

The list shows the hex address and the instruction in disassembled form, at each break-point position. The symbolic label is also shown for break-point addresses that were specified in that way.

● TESTING WITH BREAK POINTS

To execute the program to be tested, using break-pointing, use the "I" command, which has one optional parameter to specify the starting address for execution. If no parameter is given, execution continues from the last break-point reached, or from the last traced instruction (see commands A, Sand T). Example:

```
I@
I LABEL I
$4E55 I
```

(processed from break-point or trace)

All breakpoints in the current table are inserted into the target code, and control passes to the target program at the specified address. On the nrst entry, the Direct Page register will be set to zero. Subsequently, all registers will be reset to their values when execution was interrupted.

Execution then proceeds at full speed until a breakpoint is reached, when DREAMBUG immediately regains control and lists the CPU registers across the screen, in the sequence:

```
CC A B DP X Y U PC
```

A heading line is shown before the register display line, only when such a heading line does not already exist on the screen.

All the register values are shown in hexadecimal. Example:

```
84 48 2A 00 14EF 0004 2112 4E27
```

The instruction that was about to be displayed is then shown in hex and disassembled form. Example:

```
4E27 2715          BEQ $4E3E
```

ThiE\ is the instruction that will be executed next if you give an "I" or a trace command with no start address.

If no break point was reached but the target program exists back to DREAMBUG, then EXIT is shown in inverse video followed by the register display. Again, all break points are removed from the target program.

When DREAMBUG has regained control you are free to execute any other command to help testing the program. If you choose to enter the target program again from a specific address, you should ensure that it is an address at the same level on the system stack as when execution was interrupted.

● INSTRUCTION TRACING

The ability to trace the execution of machine code in RAM or ROM is supported by four commands, A, H, S, and T.

● AUTO TRACING

The "A" command (Auto trace) reports the execution of each instruction and proceeds immediately with the next. It takes two optional parameters. The first specifies the address of the first instruction to be traced. If omitted, auto-trace occurs from the last trace or break-point. The second parameter controls the information displayed after each instruction: a value of 1 gives a register dump, 2 shows the instruction about to be executed, and 3 shows both registers and instructions. The default value is 1. Example:

```
A TAG,2
A 20009
A ,3                    (param 1 omitted)
A
```

Always use option 3 when printing an Auto Trace. Auto trace mode is terminated by typing BREAK.

● TRACE 1 OR MORE INSTRCCTIONS

The "T" command (Trace) executes 1 or more instructions and then returns to give a register dump and prompt. The first parameter specifies the starting address. When omitted, trace continues from the last trace or break point. Parameter 2 is also optional. It specifies the number of instructions to be executed before giving a register dump and prompt. The default value is 1, the maximum is 32767: Example:

```
T LABEL
T $4E21,5
T
T ,20          Execute the next 20 instructions.
```

● STOP MODE TRACE

The "S" command allows you to specify a condition to be tested for after each machine instruction. When the condition is found, a register dump is given, and the next instruction displayed. The command takes three parameters. The optional first gives the starting address as in A and T commands. The second gives the memory address involved in the stop condition test.

The optional third parameter specifies the condition to be tested for. When omitted, trace stops when the byte at the test address changes from its value at trace start. When parameter 3 is given as a decimal or hex. value, then trace stops when one byte at the test address contains that value.

Parameter 3 can alternatively be coded as an asterisk (*). This will cause the trace to terminate when the instruction at the test address is about to be executed. Example:

```
S @,$4E55 stop when the contents of $4E55 changes
S ,TAG,$FF stop when the byte at TAG contains hex FF S ,$4E21, *
stop when the program counter reaches $4E21
```

The trace commands execute the target program at a maximum rate of 25 instructions per second. The 50 Hz timer clock is used extensively by DREAMBUG and so programs which use the IRQ interrupt cannot be traced.

All the trace commands can be prematurely terminated by typing BREAK.

If any traced program goes through the exit back into DREAMBUG, then the EXIT message is shown and tra_e mode terminates.

If trace command is given with no starting address (param. 1 omitted) but DREAMBUG does not know where to trace from, the command is flagged as logically invalid. This can occur Jor instance after the EXIT has been taken.

● TRACE HISTORY

The "H" command (History) displays the last 8 (or less) instructions that were executed by any of the "A", "S", or "T" trace commands. The instructions are shown in hex. and disassembled form. The display is followed by a register dump and the next instruction to be executed is shown.

This command is particularly useful after a stop condition has been encountered, to show for instance the path via which an error occurred.

The history table is reset by any command that requests target program execution from a specific address.

● NORMAL EXECUTION

The "X" command executes the target program without break points or tracing. The one optional parameter specifies the starting address. DREAMBUG regains control only when the target program successfully negotiates the exit. Example:

X@	execute from the start label
X	execute from the last trace or break-point

● MEMORY DUMP

The "D" command displays the contents of memory starting from the address in parameter 1, up to the parameter 2 address. **If** address 2 is omitted, a continuous dump is given.

. The dump shows the memory address in hexadecimal and the memory contents in hex. and Ascii characters. Non-displayable Ascii bytes are shown as full stops.

The dump can be terminated by typing BREAK.

While dumping, the display speed can be controlled by typing a number from 1 to 9, 1 gives the slowest scroll, 9 the fastest.

Typing a "0" (zero) will freeze the display until another character is typed.
Typing "U" will switch into the un-assemble mode (see Unassemble command).
Type "D" to get back into dump mode. Example:

```
D $4E21,$4E28

                  AB.3
4E21 41 42 07 33  @.*.
4E25 40 83 2A IF
```

● DIS-ASSEMBLY (UN-ASSEMBLE COMMAND)

The "U" command gives a display of memory interpreted as machine instructions, shown both in hexadecimal and in pseudo assembly language form. The parameters are the start and finish addresses as in the "D" command.

Bytes which cannot be interpreted as valid 6809 instructions are shown in hex. and Ascii where displayable. The character value of immediate operands is also shown where displayable.

The keyboard can be used to control the display as in the Dump command.

Type "D" to switch into dump mode. Type BREAK to terminate disassembly. Example:

```
U $79BB,$79C1

79BB   BD8006        JSR    $8006
79BE   8157          CMPA   #$57    W
79CO   27F5          BEQ    $79B7
```

● USING A PRINTER

If you have a printer available, the results from the trace commands, memory dumps, and disassembly, can be output to the printer as well as the screen.

The DREAMBUG command "P" enables printing for all subsequent output. The "0" command turns offprinting. These commands have no parameters.

● MEMORY EXAMINE AND CHANGE

The Memory Modify command (M) allows you to change the contents of any bytes in RAM. The one optional parameter specifies the first byte to be displayed for possible modification. If the parameter is omitted, the command continues from the last byte accessed.

Type M and the RAM address. On typing ENTER the memory address is displayed followed by its contents in hexadecimal. A dollar sign prompt is then shown to prompt you that any replacement value must be given in hexadecimal.

To move on to the next byte, just press ENTER. Hold the ENTER key down to go into auto-repeat.

To change memory contents, type pairs of hexadecimal characters against the required start address. On pressing ENTER the values are loaded into memory and the display moves onto the byte after the last one changed.

The memory bytes changed are re-read and checked equal to the requested new values. If a mismatch exists, the message ROM? is shown and the "M" command terminates.

As DREAMBUG's keyboard buffer is 20 bytes long, you can enter new data for up to 10 consecutive bytes before typing ENTER. Each new byte value must be typed as a hexadecimal pair.

For ease of readability, you can type commas between pairs - they are optional.

Typing a minus sign (-) causes the previous byte to be displayed. A string of minuses can be entered to go back to a maximum of 20 bytes.

Type BREAK to exit from the "M" command.

Example: Assume bytes 4E60 to 4E63 contain the word "DARK" which is to be changed to "DUCK".

```
M $4E60                 (press ENTER)
4E60 44 $               (press ENTER, no change)
4E61 41 $55,43          (press ENTER, changes 2 bytes)
4E63 4B $--             (ENTER, go back 3 bytes)
4E60 44 $               (hold down ENTER)
4E61 55 $
4E62 43 $
4E63 4B $
>                       (press BREAK)
```

● REGISTER MODIFICATION

The "R" command is a special case of memory modification as it displays the memory bytes where the register contents were saved when DREAMBUG regained control from your program. No parameter is required. The register bytes are accessed in the sequence CC A B DP X-high X-low Y-high Y-Iow V-high V-low PC-high PC-low. You may find it helpful to use the "H" command first to get a labelled register dump across the screen.

After modifying registers as desired, these new values will be in force when you continue execution of the target program.

● EVALUATE COMMAND

The "V" command takes two address type parameters and displays in hexadecimal the sum of the two, and the value of the first minus the second. The second parameter is optional - the default value is zero. Example:

```
V $400,$280
0680 0180          (hex sum & difference)

V 25,100
OOFD FFB5

V LABEL
4E25 4E25          (address of LABEL)

V 10966            (decimal to hex
2AD6 2AD6          conversion)
```

. LEAVING DREAMBUG

The "Q" command (Quit) returns control to DREAM or BASIC.

The break point table will have to be re-created on re-entry to DREAMBUG.

● MAXIMUM WORKSPACE

To allocate as much memory to ALLDREAM as possible, proceed as follows before entering DREAM. Firstly use the PCLEAR command in BASIC to reserve the minimum 1 high resolution graphics page. Then use the CLEAR statement with zero string space, and lowering the second address. In order to leave plenty of space for the CPU stack, it is advisable not to go below 3400 decimal. Example:

```
PCLEAR 1
CLEAR 0,3400
```

Then execute DREAM. The workspace now has over 29,000 bytes.

APPENDIX G - DREAMBUG COMMANDS

Parameter Types:

a	address (label or decimal or hex, max, value 65535)
b	1 byte value (decimal or hex, max, value 255)
c	counter (decimal or hex, max, value 32767)
i	indicator (decimal, value 1, 2 or 3)
*	special (indicates PC value)

Commands: (brackets mark optional parameters - do not code brackets as part of a command)

A (a) (,i)	Auto trace from address "a" i = 1 displays registers (default) i = 2 shows instructions executed i = 3 shows registers and instructions
B a	Add break point for address "a" to table
C a	Clear break point for address "a" from table
D (a1) (a2)	Dump memory from a1 to a2
H	Show trace history
I (a)	Insert break points into target code & execute from address "a"
L	List break point table
M (a)	Memory examine and modify Printer off
P	Printer on
Q	Quit - return to DREAM or BASIC
R	Register examine and modify
S (a1), a2	Trace execute from aI, stop when byte at a2 changes
S (a1),a2,b	Stop when byte at a2 equals value "b"
S (a1),a2,*	Stop when PC reaches a2
T (a1) (,c)	Execute "c" instructions starting from a1
U (a1) (,a2)	Un-assemble memory from a1 to a2
V a1 (,a2)	Evaluate a1 +a2 and a1-a2
X (a1)	Execute from a1 without trace or break points
Y	Re-establish text mode