DASM
ASSEMBLER

COMPUSENSE

Software for Dragon/Tandy

WHAT IS AN "ASSEMBLER"

You can't avoid references to "machine code" or "assembler language" when talking about computers or microprocessors. This is because a computer works with numbers and is controlled using numbers. These numbers are the "machine code".

The first computers were programmed using machine code and, as you might have guessed, it was quite difficult and slow.

Assembler programs were developed to make the job of programming the computers easier by giving an easily remembered name (mnemonic) to each different machine code instruction. This was called "assembly language".

The computer was then able to assemble (i.e. translate) the assembly language program into machine code which it could understand.

The DASM Assembler performs this same function on your own computer to allow you to make full use of the 6809 microprocessor inside your DRAGON (or TANDY COLOR COMPUTER).

The remainder of this booklet explains how to write assembly language and how to use the assembler. It is not intended to be a complete textbook on the programming of the 6809. However it does contain useful information and examples.

Please read this manual carefully before attempting to use DASM. You are recommended to try the Sample Program on page 20 as a first step in using DASM.

NOTE: in this manual the number zero is "0" be careful not to confuse this with the letter "O".

CAUTION: YOU MUST SWITCH OFF YOUR COMPUTER WHEN INSERTING OR REMOVING ANY CARTRIDGE.

HOW TO WRITE AN ASSEMBLY CODE PROGRAM

INSTRUCTIONS

The DASM Assembler Cartridge and Manual are supplied with a 6809 Reference Card. All the names of the instructions that you can use with the 6809 microprocessor are on this card. When you have mastered the programming of the 6809 you will be able to write your programs using this card occasionally. However until then you will need a good book on 6809 Assembler Programming.

The Assembly instructions are typed in and edited exactly as for BASIC programs. Several instructions may be written seperated by a ":" just as in BASIC. For Example:

```
0120 CLRA:CLRB
```

Note: some extra instructions to control DASM are described elsewere in a section called "Assembler Directives".

LABELS

Any instruction may be identified by a LABEL.

For example:

```
1000 @START LDX 0,Y:@LOOP CLR 0,X+:CMPX 2,Y:BLS @LOOP
```

The label is optional (except on the EQU Assembler Directive instruction - see below). A label is used to identify a data location or an instruction. Labels always start with an @ character and may have any number of alphabetic or numeric characters (i.e. A to Z and 0 to 9). If you use a label with more than six character after the @ the only the first 5 characters and the last characters are used. (i.e. the label @LABEL1 is treated exactly as @LABEL11 or @LABEL1111111111).

A label has a effective numeric value. In the case of @START in the above example this is the actual location, in the memory of the computer, of the LDX instruction.

OPERANDS

Most instructions must have an OPERAND. These are all instructions except those which are of the INHERENT type. For example: the CLR instruction must have an operand but CLRA is of the INHERENT type and does not have an operand (in fact the A register is the operand). In the example above 0,X+ is an operand of the Indexed type.

Types of Operand

There are four types of operand:

1) IMMEDIATE
2) DIRECT
3) INDEXED
4) EXTENDED

Each type must be coded in it's own particular way.

In the descriptions below, the symbol ■ will be used to represent a numeric value. This may be in any of the forms as shown:

a) decimal number eg. 123
b) hexadecimal number eg. $9AB
c) literal constant eg. !A (= $41 or 65)
d) the current address eg. *
(this is the value that a label on the same instruction would have)
e) label eg. @START
f) any simple sum formed from the terms above (i.e using + or -)
eg. *-@START+$AB-!A-123

Immediate Operands

An immediate operand starts with a "#" character in the general format: #■ . eg. LDD #1234.

Note: The EXG and TFR instructions have a different form specifying a pair of registers eg. EXG A,B. The PSH and PUL instructions can be coded with a list of registers. eg. PULS A,B,PC.

Register names used in EXG, TFR, PSH and PUL are: A B D X Y U S PC CC DP

Direct Operands

A direct operand starts with a ">" character in the general format >■ . eg. STB >$8A .

Direct Addressing mode is used in conjunction with the Direct Page register which forms the most significant byte of the address. The Direct Operand forms the least significant byte of the address.
i.e. if the DP register contains 0 (zero) then the above example stores the contents of register B at location $008A.

Note: the DP register may be set using EXG, TFR of PUL instructions. This should normally be avoided or done with care as unpredictable results may occur if the DP register is not restored when returning to BASIC.

Indexed Operands

The indexed addressing modes allow a variety of ways of using data in the 6809 registers to reference information in memory.In fact the power and flexibility of this indexed addressing mode is the main reason for the 6809 being called "the Programmer's Micro".

The indexed modes are grouped together as : Constant Offset, Accumulator Offset, Auto-Increment/Decrement and Indirect Modes.

In the following desciptions the symbol R stand for the X,Y,U or S registers,

Constant Offset

General Format: ,R or N,R . (,R is the same as 0,R)

Example:

```
1001  LDY #1234 LOAD Y REG WITH IMMEDIATE VALUE 1234
1003  LDA ,Y    LOAD A REG FROM MEMORY LOCATION 1234
1005  LDB 5,Y   LOAD B REG FROM MEMORY LOCATION 1239
```

Acummulator Offset

General Format: A,R B,R or D,R. This mode is similar to the Constant Offset above except that the offset is actually in the A, B or D Register. This is a very powerful feature and gives, for instance, an easy way of using tables.

Example:

```
1010 LDY #TABLE LOAD ADDRESS OF TABLE
1012 LDA @INCHAR GET CHARACTER TO TRANSLATE
1014 LDA A,Y    GET TRANSLATION OF CHARACTER
```

Auto-Increment/Decrement

```
Auto-Increment Formats   ,R+  and  ,R++
Auto-Decrement Formats   ,-R  and  ,--R
```

This indexing mode is especially useful in program loops to process a table or list. Each time that the instruction is executed the index register (X, Y, U or S) is incremented (or decremented) by 1 or 2. The difference in the notation between the auto-increment (the + is after the register) and the auto-decrement (the - is before the register) is to remind you, the programmer, that the index register is INCREMENTED AFTER the instruction is carried out but DECREMENTED BEFORE by the number of "+" or "-" signs.


```
0110  LDX #1000
0120  LDX #1020
0130  LDB #10
0140  BSR @REVERSE
....
....
0250 @REVERSE EQU *
0260 LDA ,X+  GET A BYTE FROM THE INPUT FIELD
0270 STA ,-Y  MOVE TO REVERSED FIELD
0280 DECB COUNT
0290 BNE @REVERSE LOOP UNTIL B IS ZERO
0300 RTS
```

This example shows how auto-increment and auto-decrement can be used to move a list of 10 characters in reverse order.

The character at location 1000 is moved to 1019
" " " " 1001 " " " 1018
. . . .
" " " " 1009 " " " 1010.

Program Counter Relative

Formats: N,PCR (8/16 bit offset) N,PCRB (8 bit offset)

This mode is particularly important when writing programs which are to be RELOCATABLE. This means that the program must still work when it is copied to a different place in the memory of the computer. This can be difficult (if not impossible) with some microprocessors but is especially easy with the 6809 because of the PC Relative Mode.

When you use the PC Relative Mode the DASM Assembler calculates the offset from the value of the Program Counter (PC Register). If your program is relocated then the value in the PC register will be different but the relative offset will be the same.

Example:

This shows how the previous example (for Auto-increment/decrement) can be made RELOCATABLE using PC Relative indexing.

```
0010 @DATA  RMB 20  DEFINES THE DATA AREA
....
0110   LEAX @DATA,PCR
0120   LEAY @DATA+20,PCR
0130   LDB  #10
0140   BSR  @REVERSE
```


Note: PCR sometimes gives a 16 bit offset where a 8 bit offset is sufficient (i.e. were a reference is made to a label later in the program). If you want this to be an 8 bit offset then use PCRB.

Example:

```
1500  LDA @VALUE,PCRB
....
....
3300 @VALUE FCB 55
```

Indirect Modes

To the newcomer to assembler programming the indirect addressing modes can be very difficult to understand and use. Good prLrams can be written without using indirect addressing at all. However when designing your program it is worthwhile considering whether indirect addressing techniques can be used advantageously. This often means choosing a design which uses pointers and tables.

The formats for indirect operands are as follows:

Constant Offset	(,R)	(N,R)
Accumulator Offset	(A,R)	(B,R) (D,R)
Auto-Increment/Decrement	(,R++)	(,--R)
PC Relative	(N,PCR)	(N,PCRB)
Extended	(N)	

Note 1 : Because of the limited keyboard layout of the DRAGON and TANDY computers the DASM assembler allows both round ( ) or square brackets [ ] to be use in indirect notation. The standard Motorola notation uses square brackets only for indirect operands.

Note 2: the single increment/decrement forms (i.e. ,R+ and ,-R) are not allowed for the indirect mode.

Note 3: the Extended Indirect format is a special case and treated as an indexed format operand.

The Indirect Mode works by first calculating the operand inside the square brackets. This gives the location not of the data, but of a pointer to the data.

For example:

```
2000  LDA (@POINTER)  LOADS "X" INTO REG A
....
3000 @POINTER FDB @DATA
....
4000 @DATA FCC "X"
```


Extended

Format: **N** . This is the simplest way of referencing a specific location in memory.

For example:

```
1000 @DATA EQU 1000
....
3300 LDA @DATA  LOAD THE CHARACTER AT LOCATION 1000
3310 STA $400  STORE AT LOCATION $400 = 1024
```

Note that the extended mode gives RELOCATABLE code only when it is used to reference data or subroutines which are always in the same place. (eg. in the DEMON monitor or the BASIC ROM or Reserved Areas).

COMMENTS

DASM allows you to write two types of comment:

a) after the operand (if there is one)

for example:
```
1010  CLR 0,X+ CLEAR NEXT BYTE
1020  CLRA CLEAR REGISTER A
```

b) as a separate line starting with "*"

for example:
```
2010 * THIS IS A COMMENT
2020  CLRA:* THIS IS ALSO A COMMENT
```

Note: If you use the second type of comment then the rest of the line is treated as a comment even if you code a ":" and another instruction.

ASSEMBLER DIRECTIVES

You will have noticed in the examples some extra instructions which are not described on your 6809 Reference Card (i.e. EQU FCC RMB). These are instructions to the DASM Assembler to do essential tasks such as reserving space for data.

END

This is the most important Assembler Directive as it must be coded as the last line of your Assembler Program. END may have an operand which is the start address of your program. For example :-

```
10 @START EQU *

program lines .....

999 END @START
```


The END statement has put the value of @START into the register used by BASICs EXEC command. When DASM has finished simply enter EXEC and the machine code program will begin working from @START. Any label can be used for this purpose and it need not be at the beginning of the program. The EXEC statement can be part of the BASIC program following the end of the DASM assembly. Note if another EXEC is used at any time then the value set by DASM will be overwritten.

EQU

This is very useful as it allows a label to be defined without generating any machine code. This label may be a data location or a data value which is used in many places in your program but may change.

For example:

```
4000 @TABLE EQU *
4005   RMB 100  RESERVE 100 BYTES FOR TABLE
4010 @LENGTH EQU *-@TABLE LENGTH OF TABLE
```

RMB

This is used (as in the above example) to reserve space in your program for data.

FCB

This generates one or more constant bytes (8 bits) of data in your program.

For example:

```
2000  FCB  0,$FF,-3,5+$12
```

FDB

This is similar to FCB but generates 2 character (16 bit) constants.

For example:

```
2010 FDB $1234,@LABEL
```

FCC

This is again similar to FCB but allows a strings of characters to be defined as a constant. As in FCB a single byte (8 bit) constant may also be defined. This is useful as a delimiter for the string. The string must start and end with a " character. If you want a " in the string then type two " characters.

For example:

```
2020  FCC "AB""CD",4   AB"CD FOLLOWED BY $04
```


ORG

This is used to tell DASM where to put the machine code that it produces. Normally this is not necessary as DASM automatically picks the free space after BASIC (see the section on running the DASM Assembler).

ORG must be used with care as it is possible to make DASM overwrite it's own or BASIC's work areas. (see the description of PPO below if you think this might be happening).

For example:

```
2030 ORG $400       DEFAULT VIDEO PAGE
2040 GVPAGE RMB $200 DEFINE VIDEO PAGE AREA
```

DSP

This is used to make DASM display the results of the assembly run on the monitor or TV. This is the initial setting and you do not normally need to use this unless you have used PRT or OFF.

You can slow down the rate at which the lines are displayed by DASM with an operand parameter on the DSP command. For example: 0010 **DSP $FFFF** will give the maximum delay of approx. 1 second per line.

PRT

This is used to make DASM print the results of the assembly run on the printer.

You may specify two control character to be sent to the printer at the end of each page (see the PAG command below) to make the printer skip to the top of the next physical sheet of paper. For example if your printer requires a $0C to do this then code: **PRT $C.** This will result in null ($00) and top of page ($0C) control characters being sent to the printer after the last line on each page. Check the manual for your printer to find the control characters that your printer requires. If you do not want to use this "page throw" feature or if your printer does not support it then use **PRT 0.**

The number of lines per page is set with the PAG command.

OFF

This is used to stop DASM printing or displaying anything. this should only be used when you have eliminated all the bugs from your program. You will not be told about errors when you use OFF (except via the Error Indicator).

ALL

This is used to make DASM print all the instructions that it processes. You can use ALL, OFF and ERR to select only the parts of your program that you want to display/print.

ERR

This is used to make DASM print only the instructions which are in error. This is the normal setting.

FML

This is used to make DASM display/print in Long format. This includes the address in memory and the first 8 bytes in hexadecimal. Use FML with PRT and ALL to get a full listing of you program on your printer. FML is not recommended for normal use when displaying on your TV/Monitor because of the limited 32 character line length.

FMS

This is used to make DASM display/print in Short format. Short format contains just the BASIC line number and the full instruction. This is easier to read on the TV/Monitor than the Long format. FMS is the default setting.

PAG

This is used to make DASM stop after displaying a number of lines on your TV/Monitor. You can continue by pressing any key except BREAK. Note: Pressing BREAK at any time stops DASM and returns you to BASIC mode.

You can also use PAG to control the paging when you have specified output to printer using PRT. See the description of the PRT command (above) for full details.

If you use PAG without an operand then DASM will stop at that point (or start a new page on the printer). Using PAG with an operand changes the page size only.

The initial page size is 15. i.e equivalent to PAG 15.

Examples of use:

```
0010 PAG 10:DSP $4000:ALL:* STOP EVERY 10 LINES

0010 PRT $C:PAG 55:PAG:ALL:* PRINT WITH 55 LINES PER PAG
```

PPO

DASM normally prints during the second pass through your program. The PPO command make DASM print during the first pass as well. This may be used to locate certain types of programming error which can cause DASM to fail during Pass One without displaying any messages.

Note that forward references to labels are always indicated as errors in Pass One and that these error messages disappear in Pass Two.

HOW TO ASSEMBLE AND RUN YOUR PROGRAM

The first step is to plug in the DASM cartridge to the expansion port on the right hand side of your computer. **CAUTION: you must switch off the computer whenever a cartridge is to be inserted or removed.**

To assemble a program using DASM you must precede your program with the BASIC statements CLEAR and EXEC.

CLEAR is used to define the areas which DASM can use for it's label table (the string space area is used for the label table) and the free space in memory (after the area allowed for use by BASIC) where DASM can put the machine code that is generated. Each label uses 10 bytes of memory. When you code CLEAR you must specify an address in free memory but **NOT** in the BASIC ROM (i.e. below &H8000).

EXEC is used to execute the DASM assembler at address **&HCFFA.**

The last instruction in your program must be END. If you put an operand on the END instruction then the value of the operand is put into the BASIC EXEC vector. This means in simple terms that if you put the start address of your program as the operand on END then you can execute your program by using EXEC in BASIC or entering EXEC on the keyboard.

For example:

```
0010 CLEAR 1000,&H6000:EXEC &HCFFA
0015 * 100 LABELS - PLACE MACHINE CODE AT $6000
1000 @START EQU * START ADDRESS OF PROGRAM
.....
.... your 6809 assembler program
.....
4990 END @START
4999 REM DASM RETURNS TO BASIC HERE - NOW RUN PROGRAM
5000 EXEC
9999 END
```

By clever use of labels and the FDB/FCB commands you can automatically get DASM to put the length and start address and other information where it can be picked up from BASIC eg. at the start of the program.

Having written, assembled and run your program you will probably have met your first big problem - the program doesn't do what you wanted it to do.

Unfortunately DASM cannot help you any further with debugging your program but it's sister cartridge the DEMON monitor has many facilities for just this purpose. In particular the setting of breakpoints. DEMON also has some useful routines to make writing programs for the DRAGON easier.

Whether or not you are using DEMON to debug your program you should think about how to test your program when you are designing it - not when you find out that it doesn't work. A good design will allow the testing of parts of the program seperately so that errors can be isolated more easily.

The final stage of putting the tested parts together will then be a much more successful and rewarding experience.

WARNING: SAVE YOUR WORK ON TAPE FREQUENTLY AS AN UNTESTED ASSEMBLER PROGRAM CAN DESTROY THE COPY IN MEMORY.

This can happen when an error in your program makes the computer 'lock up' until it has been switched off and on again.

ERROR MESSAGES

DASM checks your program and produces an error message if it finds an error. The following are all the messages with an explanation of why they are produced.

<E> **LABEL?**

The label field is incorrect. i.e. not a valid label.

<E> **INSTRUCTION?**

The instruction is not a valid 6809 instruction or a DASM Assembler Directive.

<E> **OPERAND?**

The operand field is in error. The operand may be omitted, not a valid type for this instruction (eg. STA #0) or incorrect (eg. LDA ,X- or EXG A,Y).

<E> **DUPLICATE LABEL**

The label has been defined previously in the program. Labels may only be assigned a value once.

<E> **NEED LONG BRANCH**

A Short form BRANCH instruction has been coded but the relative offset is too big for 8 bits. Change the instruction to the Long form (eg. BRA to LBRA) or restructure the program.

<E> **LABEL UNDEFINED**

A label value has been used in the operand field but this lall has not been defined yet. This error occurs when you use a label defined later in the program on an ORG or RMB statement as DASM (and most assemblers) does not allow this. If you cannot easily solve the problem by restructuring the program then define the label values at a fixed location by using ORG.

<E> **OPERAND TRUNCATED**

In a situation were an 8 bit value is required (for example: LDA #VALUE) a value was found with was too large for 8 bits. The reason for the error should be investigated. The value has been truncated to 8 bits and then used as normal.

<E> **LABEL TABLE TOO SMALL**

The space for the label table (defined with CLEAR) is too small. Remember that you must have 10 bytes for each label that you will use in your program.

<E> **NOT RAM**

This error occurs when you attempt to assemble into part of the computer

which does not contain Random Access Memory. For example if you write a long program which extends past the first 32K of memory (i.e. $7FFF is the highest location at which you can assemble your program.

You will get this error if you forget the CLEAR before running the assembler. This error may also occur if the memory in you computer is faulty.

Other errors may also be reported when this error occurs but can be ignored until all the NOT RAM errors have been resolved.

ERROR INDICATOR

When DASM has finished assembling your program the first byte of DASM's work area at &H600 will be set to zero if no errors have occured. This indicator can be tested with PEEK in BASIC (see the SAMPLE PROGRAM below).

SAVING YOUR PROGRAM ON TAPE

Use CSAVE to save the source program.

The assembled machine code can be saved on tape by using the standard CSAVEM command. To use this you must determine the first and last addresses of the program and the execution start address. Addresses are displayed by DASM when you use the Long Format (see FML assembler directive). The sample program shows one way that the CSAVEM addresses can be found and used automatically.

Your saved machine code can then be reloaded using CLOADM. Remember to use CLEAR beforehand to reserve space for the program.

SAMPLE PROGRAM

The following program demonstrates how the low resolution video page can be used by an assembler program. All the possible characters that can be displayed on the DRAGON/TANDY COLOR are displayed on the screen with a small delay (needed as the assembler program is so fast). Also illustrated are: the use of the ERROR INDICATOR, saving the assembled machine code to tape and displaying the Symbol Table.

```
10  CLEAR 400,&H4000
15  EXEC &HCFFA
16  ALL
18  @FROM FDB @FROM,@TO,@START FOR AUTO CSAVEM
20  @START EQU *
24  CLRA              START WITH CHARACTER $00
30  LDX #$400         START OF VIDEO PAGE
40  @LOOP STA 0,X+    CHANGE NEXT BYTE IN VIDEO PAGE
50  CMPX #$0600       REACHED END OF VIDEO PAGE ?
60  BNE @LOOP         LOOP UNTIL END OF VIDEO PAGE
70  LDX #$400         RESET POINTER TO START OF VIDEO PAGE
80  LDY #$8000        DELAY COUNT
90  @DELAY LEAY -1,Y:BNE @DELAY     DELAY LOOP
100 INCA              DO NEXT CHARACTER
110 BNE @LOOP         LOOP UNTIL A IS ZERO AGAIN
120 RTS               END OF PROGRAM - RETURN TO BASIC
130 @TO END @START
400 PRINT"SYMBOL TABLE":FOR I=&H4000-400-1 TO &H4000-1 STEP 10
410 IF PEEK(I+4)<>0 THEN PRINT"@";:ELSE 450
420 FOR J=I+4 TO I+9:PRINT CHR$(PEEK(J));:NEXT J
430 PRINT,HEX$(PEEK(I)*256+PEEK(I+1))
440 NEXT I
450 IF PEEK(&H600)<>0 THEN PRINT"ERRORS":END
460 INPUT"PRESS ENTER TO EXECUTE PROGRAM";X$:EXEC
500 INPUT "SAVE PROGRAM TO TAPE Y/N & ENTER";X$
510 IF X$="N" THEN END ELSE IF X$<>"Y" THEN 500
520 B=&H4000
530 CSAVEM"SAMPLE",PEEK(B+1)*256+PEEK(B+2),PEEK(B+3)*256+PEEK(B+4),
    PEEK(B+5)*256+PEEK(B+6)
999 END
```

Turn your computer off and plug in the DASM cartridge. Turn your computer on, type this program in and then RUN it. It is not necessary to type in the comments. If any errors are detected by DASM, correct them and RUN again.

If you delete Line 16 then only errors will be displayed.

DASM - Two Pass Symbolic Assembler for DRAGON/TANDY

The DASM Assembler is an original software product written by:

COMPUSENSE Ltd
286D GREEN LANES
PALMERS GREEN
LONDON N13 5XA

Telephone 01-882-0681/6936
Telex 8813271 GECOMS G

COMPUSENSE was established in 1979 and has specialised in software and hardware for the Motorola 6800, 6809 and 68000 microprocessors.

COMPUSENSE supplies hardware and software for systems based on the SS-50 BUS in addition to an expanding range of products for the DRAGON and TANDY COLOUR computers.

All enquiries are directed to the above address.

A series of programs written using the DASM assembler with the full source are available. Titles include: a Disassembler and The Game of Life.

DRAGON is a trade mark of DRAGON DATA Ltd.

TANDY is a trade mark of the TANDY CORPORATION.