DEFT Macro/6809 Assembly Language

1 Introduction

DEFT Macro/6809 is a program which reads assembly language source code and produces object code suitable for linking by **DEFT Linker**. This assembler features the following facilities:

- Motorola capatible source conventions and directives
- Builtin macro facility provides for substitution of up to 9 parameters
- Copy facility provides the ability to include several source files in a single assembly (very convenient for common equate and macro definition files)
- Object code format provides relocation, separate assembly and easy interfacing to Pascal via **DEFT Linker**

This section describes the **DEFT Macro/6809** assembly language. Readers are expected to already be familiar with the 6809 instruction set, registers and addressing modes.

2 Language Syntax

The syntax used in the **DEFT Macro/6809** is generally compatible with those found in other assemblers for the 6809.

2.1 Line Format

Assembly language source code is interpreted as a series of lines read from the source file (or *copy* files). Each line is made up of up to 4 fields. These fields are separated from each other with 1 or more blanks.

1. The *label* field is an optional field which, when present, contains an identifier that is to be defined (see below). This field is present when the first character in a line is non-blank.
2. The *opcode* field is a required field which begins with the first non-blank character following the first blank character in the line. It contains either a 6809 opcode, directive or macro and is used to control how the other fields in the line are used.
3. The *operand* field is the next field after the OPCODE field. It is a required field for some opcodes and is not present for others. Most of the discussion of language syntax describes the way that this field is used. Note that this field may contain blanks in some circumstances (see below).
4. The *comment* field is the last field in the line and consists of the remainder of the line following the operand field (or opcode field if the operand field is not present).

Note that in the listing produced by the assembler, these fields are automatically lined up on predetermined column boundaries. The use of the *label*, *opcode* and *operand* fields is more fully explained in the following sections.

2.2 Identifiers

An identifier is a name used to represent either an *absolute* or *relative* value. It is a set of up to 12 letters and numbers which must begin with a letter. Lower case letters are accepted and printed as such on the listing even though they are kept internally as upper case letters.

An identifier is defined when used in the *label* field in exactly 1 source line. The *opcode* field of that source line will determine what value is assigned to the identifier. This identifier can be used in the operand field of a source line where the identifier's value will be

used. In general, an identifier can be used as an operand even in source lines preceding the one in which it is defined. For all opcodes except *EXT*, *EXTA* and *EQU* the identifier acquires the value of the *location counter* (see below) at the point in the program where the identifier is defined.

An identifier which is defined in this manner has a *relative* value. This value is one which will be *relocated* by **DEFT Linker** into an *absolute* value when the final binary file is created. The eventual value of a relative identifier is determined by adding the location in memory where the object code is located to the relative value determined by the assembler.

An identifer may be defined with an *absolute* value by using an EQU opcode and an absolute expression in the *operand* field. See *Expression* and *EQU* for more information.

2.3 Location Counter

The *Location Counter* is a 16 bit value which is kept by the assembler that represents the number of bytes of object code produced so far. You can think of it as a relative memory address (relative to the beginning of the program). This value always starts at zero and increases in value as each source line is processed. The value of the location counter is printed at the left-hand side of the page for each line of source code printed.

The location counter is represented in the *operand* field via the symbol *.

2.4 Constants

A constant is always an *absolute* 16 bit value that is represented in some specific way. The following constants are supported by **DEFT Macro/6809**:

1. *Decimal* constants are numbers in the range from -32768 to +32767. Base 10 is the default base.
2. *Hexadecimal* constants numbers in the range from $0 to $FFFF. A hexadecimal constant is identified with a leading dollar sign ($).
3. A *Single ASCII character* constant is an ASCII character preceded with a single quote ('). The value of the resulting constant is the binary value of the ASCII character. Example: 'A

4. *Double ASCII character* constants are two ASCII characters preceded with a double quote ("). The value of the resulting constant is the binary value of the first character in the high 8 bits and the value of the second character in the low 8 bits. Example: "AB

2.5 Expressions

Identifiers and constants can be combined into *expressions* with the use of the arithmetic operators plus (+), minus (-), multiply (*) and divide (/). Expression evaluation is strictly left to right with no operator precedence. There are some restrictions on the creation of expressions:

- Relative values can not be multiplied or divided.
- You can add or subtract an absolute from a relative. This results in a relative value.
- You can subtract a relative from a relative. This results in an absolute value.
- You cannot subtract a relative from an absolute.
- You cannot add a relative to a relative.

2.6 Strings

A string is a set of 1 or more ASCII characters delimited by slashes (/) or double quotes ("). The opcodes *fcc*, *title* and *stitle* are the only ones that use strings for operands. Strings cannot be combined into expressions.

2.7 Registers

The 6809 registers are named as follows:

- *A* - high order 8 bits of the general accumulator
- *B* - low order 8 bits of the general accumulator
- *CC* - 8 bit condition code register
- *D* - 16 bit general accumulator
- *DP* - 8 bit direct page register

- *PC* or *PCR* - 16 bit program counter. Both designations result in equivalent code.
- *S* - 16 bit system stack register
- *U* - 16 bit user stack register
- *X* - 16 bit index register
- *Y* - 16 bit index register

2.8 Addressing Modes

There are a number of addressing modes which may be used with the 6809 instruction opcodes. These are used in the operand field and are as follows:

- **Inherent** - this addressing mode has no operand field. The given opcode has all the addressing information necessary to complete the instruction.
- **Immediate** - this addressing mode is designated with a leading #. The expression following the # is the object of the instruction.
- **Direct** - this addressing mode is determined by **DEFT Macro/6809** when the operand expression is absolute and its high 8 bits are equal to the value in the most recent *SETDP*.
- **Extended** - this addressing mode is determined by the assembler when the operand expression is either relative, or its absolute and the high 8 bits are not equal to the value in the most recent *setdp* instruction.
- **Relative** - this addressing mode is determined by the opcode and requires a relative expression.
- **Indexed** - this addressing mode is determined when the operand is of one of the following forms:

Zero Offset	**,<reg>**
Constant Offset	**<absolute expression>,<reg>**
Accumulator Offset	**<accumulator>,<reg>**
Auto Increment	**,<reg>++**
Auto Decrement	**,—<reg>**
Program counter relative	**<relative expression>,PCR**
	<relative expression>,PC
Indirect	**[<indexed mode>]**

- **Register-Register** - this addressing mode is determined by the opcode and requires the following form: <reg>,<reg>
- **Multi-Register** - this addressing mode is determined by the opcode and requires the following form: <reg>,...,<reg>

3 6809 Instruction Summary

The following summary lists the 6809 instruction opcodes supported by the **DEFT Macro/6809** Assembler. The first column is the assembler opcode. The second column contains the addressing modes available for this opcode. The third column is the title of the instruction.

ABX	**Inherent**	**Add B to X**
ADCA	**Immediate** **Direct** **Indexed** **Extended**	**Add Memory with Carry to A**
ADCB	**Immediate** **Direct** **Indexed** **Extended**	**Add Memory with Carry to B**
ADDA	**Immediate** **Direct** **Indexed** **Extended**	**Add Memory to A**
ADDB	**Immediate** **Direct** **Indexed** **Extended**	**Add Memory to B**
ADDD	**Immediate** **Direct** **Indexed** **Extended**	**Add Memory to D**
ANDA	**Immediate** **Direct** **Indexed** **Extended**	**AND Memory to A**
ANDB	**Immediate** **Direct** **Indexed** **Extended**	**AND Memory to B**
ANDCC	**Immediate**	**AND Memory to CC**
ASL	**Direct** **Indexed**	**Arithmetic Shift Left Memory**

	Extended	
ASLA	Inherent	Arithmetic Shift Left A
ASLB	Inherent	Arithmetic Shift Left B
ASR	Direct Indexed Extended	Arithmetic Shift Right Memory
ASRA	Inherent	Arithmetic Shift Right A
ASRB	Inherent	Arithmetic Shift Right B
BCC	Relative	Branch on Carry Clear
BCS	Relative	Branch on Carry Set
BEQ	Relative	Branch on Equal
BGE	Relative	Branch on Greater Than or Equal
BGT	Relative	Branch on Greater Than
BHI	Relative	Branch on Higher
BHS	Relative	Branch on Higher or Same
BITA	Immediate Direct Indexed Extended	Bit Test Memory with A
BITB	Immediate Direct Indexed Extended	Bit Test Memory with B
BLE	Relative	Branch on Less Than or Equal
BLO	Relative	Branch on Lower
BLS	Relative	Branch on Lower or Same
BLT	Relative	Branch on Less Than
BMI	Relative	Branch on Minus
BNE	Relative	Branch on Not Equal
BPL	Relative	Branch on Plus
BRA	Relative	Branch Always

BRN	Relative	Branch Never
BSR	Relative	Branch to Subroutine
BVC	Relative	Branch on Overflow Clear
BVS	Relative	Branch on Overflow Set
CLR	Direct Indexed Extended	Clear Memory
CLRA	Inherent	Clear A
CLRB	Inherent	Clear B
CMPA	Immediate Direct Indexed Extended	Compare Memory from A
CMPB	Immediate Direct Indexed Extended	Compare Memory from B
CMPD	Immediate Direct Indexed Extended	Compare Memory from D
CMPS	Immediate Direct Indexed Extended	Compare Memory from S
CMPU	Immediate Direct Indexed Extended	Compare Memory from U
CMPX	Immediate Direct Indexed Extended	Compare Memory from X
CMPY	Immediate Direct	Compare Memory from Y

	Indexed Extended	
COM	Direct Indexed Extended	Complement Memory
COMA	Inherent	Complement A
COMB	Inherent	Complement B
CWAI	Immediate	Mask CC and Wait for Interrupt
DAA	Inherent	Decimal Adjust A
DEC	Direct Indexed Extended	Decrement Memory
DECA	Inherent	Decrement A
DECB	Inherent	Decrement B
EORA	Immediate Direct Indexed Extended	Exclusive Or Memory with A
EORB	Immediate Direct Indexed Extended	Exclusive Or Memory with B
EXG	Reg-Reg	Exhange Registers
INC	Direct Indexed Extended	Increment Memory
INCA	Inherent	Increment A
INCB	Inherent	Increment B
JMP	Direct Indexed Extended	Jump
JSR	Direct Indexed Extended	Jump to Subroutine

LBCC	Relative	Long Branch on Carry Clear
LBCS	Relative	Long Branch on Carry Set
LBEQ	Relative	Long Branch on Equal
LBGE	Relative	Long Branch on Greater Than or Equal
LBGT	Relative	Long Branch on Greater Than
LBHI	Relative	Long Branch on Higher
LBHS	Relative	Long Branch on Higher or Same
LBLE	Relative	Long Branch on Less Than or Equal
LBLO	Relative	Long Branch on Lower
LBLS	Relative	Long Branch on Lower or Same
LBLT	Relative	Long Branch on Less Than
LBMI	Relative	Long Branch on Minus
LBNE	Relative	Long Branch on Not Equal
LBPL	Relative	Long Branch on Plus
LBRA	Relative	Long Branch Always
LBRN	Relative	Long Branch Never
LBSR	Relative	Long Branch to Subroutine
LBVC	Relative	Long Branch on Overflow Clear
LBVS	Relative	Long Branch on Overflow Set
LDA	Immediate Direct Indexed Extended	Load Memory into A
LDB	Immediate Direct Indexed Extended	Load Memory into B
LDD	Immediate Direct Indexed Extended	Load Memory into D

LDS	Immediate Direct Indexed Extended	Load Memory into S
LDU	Immediate Direct Indexed Extended	Load Memory into U
LDX	Immediate Direct Indexed Extended	Load Memory into X
LDY	Immediate Direct Indexed Extended	Load Memory into Y
LEAS	Indexed	Load S with Effective Address
LEAU	Indexed	Load U with Effective Address
LEAX	Indexed	Load X with Effective Address
LEAY	Indexed	Load Y with Effective Address
LSL	Direct Indexed Extended	Logical Shift Left Memory
LSLA	Inherent	Logical Shift Left A
LSLB	Inherent	Logical Shift Left B
LSR	Direct Indexed Extended	Logical Shift Right Memory
LSRA	Inherent	Logical Shift Right A
LSRB	Inherent	Logical Shift Right B
MUL	Inherent	Multiply
NEG	Direct Indexed Extended	Negate Memory

NEGA	Inherent	Negate A
NEGB	Inherent	Negate B
NOP	Inherent	No Operation
ORA	Immediate Direct Indexed Extended	Inclusive Or Memory with A
ORB	Immediate Direct Indexed Extended	Inclusive Or Memory with B
ORCC	Immediate	Inclusive Or Memory with CC
PSHS	Multi-Reg	Push Registers on System Stack
PSHU	Multi-Reg	Push Registers on User Stack
PULS	Multi-Reg	Pull Registers from System Stack
PULU	Multi-Reg	Pull Registers from User Stack
ROL	Direct Indexed Extended	Rotate Left Memory
ROLA	Inherent	Rotate Left A
ROLB	Inherent	Rotate Left B
ROR	Direct Indexed Extended	Rotate Right Memory
RORA	Inherent	Rotate Right A
RORB	Inherent	Rotate Right B
RTI	Inherent	Return from Interrupt
RTS	Inherent	Return from Subroutine
SBCA	Immediate Direct Indexed Extended	Subtract Memory with Borrow from A

SBCB	Immediate Direct Indexed Extended	Subtract Memory with Borrow from B
SEX	Inherent	Sign Extend B into A
STA	Immediate Direct Indexed Extended	Store Memory from A
STB	Immediate Direct Indexed Extended	Store Memory from B
STD	Immediate Direct Indexed Extended	Store Memory from D
STS	Immediate Direct Indexed Extended	Store Memory from S
STU	Immediate Direct Indexed Extended	Store Memory from U
STX	Immediate Direct Indexed Extended	Store Memory from X
STY	Immediate Direct Indexed Extended	Store Memory from Y
SUBA	Immediate Direct Indexed Extended	Subtract Memory from A

SUBB	Immediate Direct Indexed Extended	Subtract Memory from B
SUBD	Immediate Direct Indexed Extended	Subtract Memory from D
SWI	Inherent	Software Interrupt 1
SWI2	Inherent	Software Interrupt 2
SWI3	Inherent	Software Interrupt 3
SYNC	Inherent	Synchronize to Interrupt
TFR	Reg-Reg	Transfer Register to Register
TST	Direct Indexed Extended	Test Memory
TSTA	Inherent	Test A
TSTB	Inherent	Test B

4 General Directives

In addition to the opcodes listed in the preceding section, which translate directly into 6809 opcodes, **DEFT Macro/6809** contains a number of directives which provide memory initialization, reservation and assembly control.

4.1 COPY

This directive allows you to copy source lines from another file into the current assembly. The standard file name found in the *operand* field is openned (with a default suffix of ASM) and read to end of file. In the current version of the assembler, files that have been copied cannot themselves contain COPY directives. Example:

```
COPY RECRDEQU:1
COPY MYMACROS
```

4.2 END

This directive is provided in order to allow the programmer to terminate his program with an END. The END directive has no OPERAND and does not result in code generation. The assembler will continue processing any source lines following an END. The assembler does not require a program to have an END since source lines are fetched until end of file is reached. Example:

```
END
```

4.3 EQU

This directive provides the capability of defining an identifier to have a specific value. The identifier found in the *label* field is assigned the value of the expression found in the *operand* field. Example:

```
LABEL1   EQU     $50
LABEL2   EQU     LABEL1*3
LABEL3   EQU     *-EARLIERLABEL
```

4.4 FCC

This directive creates an ASCII string of characters. The *operand* field contains the ASCII string to be created enclosed in either slashes (/) or double quotes ("). Example:

```
NAME    FCC   /John Q. Smith/
NAME2   FCC   "Mary Jones/MD"
```

4.5 FCB

This directive creates individual bytes with the values of the expression(s) found in the operand field. More than one byte may be defined by separating the expressions with commas (,). Example:

```
BYTES   FCB  6,$F,LABEL1,'A
        FCB  *-BYTES
```

4.6 FDB

This directive creates individual words with the values of the expression(s) found in the operand field (high bits in low order byte). More than one word may be defined by separating the expressions with commas (,). Example:

```
WORDS   FDB  5,6
        FDB  WORDS+3
```

4.7 MAIN

This directive tells the **DEFT Macro/6809** (and subsequently **DEFT Linker**) where execution should begin. Only one *main* directive should be included in a set of modules to be linked together. *Main* has no operand, so execution will be at the value of the location counter. Example:

```
        MAIN
START   ...
```

4.8 RMB

This directive reserves memory which is preinitialized to zero. The absolute expression found in the *operand* field specifies the number of bytes of memory to reserve. Example:

```
WRKAREA RMB  $200
```

4.9 SETDP

This directive specifies to the assembler what page number should constitute the direct page. This directive should generally follow a TFR instruction that loads the DP register with a new value. The expression (evaluated at assembly time) in the *operand* field is used as the new direct page number. If no *setdp* directive is given, page number 0 is assumed to be the direct page. Example:

```
LDA     #DATATABLE/256    A=Page Number
TFR     A,DP              Put in DP
SETDP   #DATATABLE/256    Tell assembler
```

Note that the above example works only if *DATATABLE* is an absolute.