TSC 6800 Mnemonic Assembler for FLEX

The TSC Mnemonic Assembler for the FLEX Operating System is a very fast and powerful assembler for the 6800 computer system. The disk version will allow the assembly of very large source files and will produce machine readable binary directly to a disk file. Several assembly time options are also available.

GETTING THE SYSTEM STARTED

The general syntax of the ASMB command is:

```
ASMB,<file spec 1>[,<file spec 2>][,+<option list>]
```

The name of the file to be assembled is <file spec 1> and defaults to a TXT extension and to the working drive. If <file spec 2> is specified, that will be the name of the binary file written to the disk. If <file spec 2> is not included in the command line, the binary file will have the same 'name' as that of the file you are assembling, but the extension will be BIN. If a binary file of the same name already exists on the disk, the assembler will ask if it should be deleted. Respond with 'Y' if it should be.

There are 8 options which may appear in the <option list>. The option list is separated from the command line by a plus sign ('+'), and each option is represented by a single letter. Following is a list of the available options and what they mean:

B Do not create a binary file on the disk. If this option appears in the list, no binary file will be created, even if <file spec 2> is specified. This is useful while running the assembler and checking for errors, before the final program is completed.

G Turn 'NOGEN' on. Refer to the full assembler manual for a description of NOGEN.

L Turn the assembler LIST mode off. The assembler will normally produce an assembled listing. The L option will suppress this listing, and only error lines (if any exist) will be displayed.

S Suppress the output of the Symbol Table. Normally the assembler will output a sorted symbol table at the end of each assembly. Including the S in the option list will suppress the output of this table.

N Output line Numbers with the assembled listing. The assembler will normally not include line numbers in the listing. During program debugging it is often useful to have each statement numbered. These numbers will directly correspond to the numbers assigned to each line while in the Text Editing System. Error lines will always be output with line numbers, regardless of the status of the N option.

T Assemble the file for an 'S1' formatted object tape. This option allows the production of an 'S1' tape directly from the assembler. Consult the full manual for Tape details.

Y This option tells the assembler to automatically delete any old binary files of the same name as the source file if binary is being produced. If this option is not specified, the assembler will prompt you if a binary file already exists.

D This option instructs the assembler to omit the date from the title line of each page. Normally, if the PAG option is set, the system date will be printed in the title line. If not in the PAG mode, no date will be printed.

EXAMPLES

Some examples of assembler command lines follow:

```
ASMB,TEST
ASMB,TEST,+LS
ASMB,0.TEST,1.TEST.CMD,+S
ASMB,TEST,+BN
```

The first example would assemble the source file TEST.TXT from the working drive and create a binary file named TEST.BIN on the same drive. The second example would do the same operation as the first example, but this time the Listing and the Symbol table output would be suppressed. The next example would assemble the file named TEST.TXT on drive 0 and produce a binary file on drive 1 named TEST.CMD. Because of the S in the options field, no symbol table would be output. The last example will assemble the file named TEST.TXT, producing a listing with line numbers, and not produce a binary file (because of the B).

ASSEMBLED LISTINGS TO A PRINTER

To have the listing printed on a printer, as opposed to the terminal, simply precede the command with P (see the 'P' command in the FLEX UCS description). This is assuming that the appropriate PRINT.SYS file exists in the operating system). As an example:

```
+++P,ASMB,TEST
```

This would cause the assembled listing of the source file TEST.TXT to be printed on the printer.

SYMBOL TABLE SPACE

The number of symbols which the assembler will support is dependent upon the amount of memory installed in the computer. As a rough guide, 12K will allow approximately 200 symbols while 24K and above will allow 1000 symbols. It is roughly proportionate for values between these extremes. The assembler automatically determines the memory size.

AUTO FIELDING

This version of the assembler performs automatic output fielding. No matter what the source file looks like in the way of field spacing, the output will tab the fields into a columnar form.

ASSEMBLER DESCRIPTION

The TSC 6800 Mnemonic Assembler was written for maximum flexibility making it usable to owners of RAM-only systems as well as disk system owners. As always, flexibility adds complexity and therefore the user is advised to read the following application notes thoroughly before trying to use this program.

It is assumed that the user is familiar with assembly language and, in particular, the mnemonics of the M6800 assembly language. Those who are not are referred to the "M6800 Microprocessor Programming Manual" or the "M6800 Programming Reference Manual", both available from your Motorola distributor.

The source language (input) for the TSC 6800 Mnemonic Assembler consists of a subset of the 7-bit ASCII (American Standard Code for Information Interchange, 1968) character set. Special meaning is attached to many of these characters as will be described later. In all cases the parity bit (most significant bit) of each character must be 0. This restriction, of course, does not apply to line numbers, if present. All source files may contain upper or lower case mnemonics and hex characters.

Each line of source for the assembler consists of any number of bytes (possibly none) preceeding the first character of the source statement, followed by the source statement, followed by a carriage return (hex 0D). The source statement consists of up to four "fields" which are free format. From left to right, the four fields are label, operator (mnemonic), operand, and comment. There must be at least one space between each of these fields. Further restrictions and options for each of these fields are:

LABEL FIELD

1) The label must begin in the 1st column and must be unique.
2) Labels consist of letter (A-Z,a-z) and numerals (0-9).
3) Every label must begin with a letter (A-Z,a-z).
4) Only the first 6 characters of any label are significant, the rest are ignored.
5) The label field may be the only field present.

OPERATOR FIELD

1) The operator is 3 alphabetic characters (A-Z,a-z) which must be followed by a space.
2) Mnemonics such as LDA A and AND B may be written as LDAA and ANDB, respectively. In this case, the fourth character must be followed by a space.

OPERAND FIELD

1) The operand field may consist of an addressing mode indicator and expression or just an expression.
2) The addressing mode indicator is either a # (Pound sign) followed by an expression for immediate addression or an expression followed by ,X for indexed addression.
3) An operand may or may not be required depending on the addressing mode.

COMMENT FIELD

1) The comment field is optional.
2) Comments may contain any character from SPACE ($20) to DEL ($7F).

EXPRESSIONS

Expressions consist of combinations of numbers and symbols separated by one of the four arithmetic operators +, -, *, /. The arithmetic is done with 16 bit integer operands and truncated as necessary. 8 bit results are taken from the least significant 8 bits. Unary (+) and (-) are allowed. Expressions must not contain spaces.

NUMBERS

Numbers are groupings of the numerals 0-9 and possibly letters prefixed or postfixed by a base indicator. Possible base indicators are shown below. The ASCII base allows a single ASCII character ($20-$5F) to be used as an operand when preceeded by a single quote.

Base	Prefix	Postfix	Comment
Decimal	none	none	decimal assumed
Binary	%	B	0,1 allowed
Octal	@	O or Q	0-7 allowed
Hexadecimal	$	H	0-9, A-F allowed
ASCII	'	not allowed	ASCII equivalence

SYMBOLS

Symbols are groupings of letters and numerals the first 6 of which are significant and the first of which must be a letter. The single character * is a special symbol whose value is the current value of the program counter (PC). Upper case letters are not equivelent to lower case, so the label 'START' is different from 'start'.

EVALUATION OF SYMBOLS AND EXPRESSIONS

Since this is a two pass assembler all symbols must be resolved in the two passes. Therefore, only one level of forward referencing is allowed.

ASSEMBLER DIRECTIVES

In addition to the 72 M6800 mnemonics this assembler supports 14 assembler directive or pseudo-ops. These pseudo-ops are listed below along with a brief description. More detailed descriptions follow.

FCC	form constant character
FCB	form constant byte
FDB	form double byte
SPC	insert spaces in output listing
OPT	activates or deactivate assembler options
PAG	skip to next page of output
ORG	define new origin (PC)
EQU	assign value to symbol
END,MON	signal end of source program
NAM,TTL	specify name or title
RMB	reserve memory bytes
LIB	process library file

FCC

The function of FCC is to create character strings for messages or tables. The character string 'text' is broken down to ASCII, one character per byte. The two allowable formats are shown below.

```
   label FCC  count, text
or
   label FCC  delimiter text same delimiter
```

where count is any decimal number. In the case where a number is used as a delimiter the first character of text must not be a comma. The character limit of any single FCC statement is 255. The use of label is optional.

FCB

The FCB pseudo-op causes an expression to be evaluated and the resultant 8 bits placed in memory. Usage is shown below:

```
        label   FCB   expression 1,expression 2,...,expression N
```

Each expression is separated by a comma with a maximum of 255 expressions per FCB statement. The label is optional.

FDB

The function of the FDB directive is identical to FCB except 16 bit quantities are assembled, i.e., two bytes generated for each expression. The required format is shown below;

```
        label FDB   expression 1,expression 2,...,expression N
```

where the label is optional. The maximum number of expressions is 127.

SPC

The SPC operator causes the specified number of spaces to be inserted in the output listing. The format is shown below.

```
        SPC   expression
```

Notice that no label is allowed. If 'expression' evaluates to zero, one space is inserted. The operator SPC itself does not appear in the output listing. If PAGE mode is selected SPC will not cause spacing past the top of the next page.

OPT

The directive OPT is used to activate or deactivate the assembler options. The format is shown below. Notice that no label is allowed and no code is generated.

```
        OPT   option 1,option 2,...,option N
```

The allowable options are:

SYM	print sorted symbol table after the listing (default)
NOS	do not print the symbol table
GEN	print code generated by FCB, FDB, and FCC (default)
NOG	print only one line for each FCB, FDB, or FCC
LIS	print the assembled source listing (default)
NOL	suppress the printing of the source listing
PAG	enable page formatting and numbering
NOP	disable page mode (default)

TAP	Enable the production of MIKBUG object tape
NOT	disable the production of MIKBUG object tape (default)

If contradicting options appear, the last one appearing takes precedence. All options take effect simultaneously at the beginning of pass 2. The default options specified take effect unless the user specifies a particular option. Only the first 3 characters of an option name are significant and multiple options are separated by a comma. Some of the consequences and uses of the options will be explained later. The disk version operates a little different than the standard assembler. The options NOS, NOG, NOL, and TAP have no affect when contained in the source file. The calling option parameters S, G, L, and T should be used respectively (see Disk version description).

PAG

The PAG operator, if the PAG option is on, causes a page eject and subsequently causes the title (if any), date (if D not set), and page number to be printed at the top of the next page. No label is allowed and no code is produced. Notice that the first page of any listing is page 0 and no title is printed on that page. The PAG operator itself wil not appear in the listing. The usual procedure is to have all the options and the title declaration folowed by a PAG by the first statements in a program.

ORG

The ORG operator, whose format is shown below, causes a new origin address (PC) for the code following.

```
        ORG   expression
```

No label is allowed and no code is produced. If no ORG appears an origin of 0000 is assumed.

EQU

EQU is used to equate a symbol to an expression as shown below. A label is required and no code is generated. Only one level of forward referencing is allowed and the equate must not be recursive.

```
        label EQU   expression
```

No code is produced by EQU.

END or MON

These operators signal the assembler that the end of the source input has occurred. No label is allowed and no code is generated.

A second version of the END statement allows for the assignment of a transfer address to the binary file created. This can be accomplished by putting a label or value in the 'operand field' of the END statement. As an example, suppose the program you are assembling is to start executing at location $100, and in the source file you have the label START on the statement which is ORGed at $100. The end statement should now include this label in its operand field in order to assign $100 as the transfer address.

```
        ORG     $100
START   LDX     $FFFF

        program here

        END     START
```

The above shows what this might look like.

NAM or TTL

These operators are used to assign a title to be printed at the top of all pages (other than page 0) if the PAG option is on. If the PAG option is off this operator has no effect. The format, as shown below allows up to 32 characters in the title. No label is allowed.

```
        TTL    text for the title
```

and no code is generated. If more than one TTL or NAM operator appears the last one "executed" will be printed on the next page.

RMB

This operator causes the assembler to reserve memory for data storage. No code is produced and therefore the contents of those memory locations are undefined at run time. The label is optional as shown below

```
        label RMB  expression
```

where 'expression' is a 16 bit quantity.

LIB

This operator causes FLEX to temporarily leave the current source file and open the file specified (on the working drive and with a default extension of TXT) for source processing. It has the following form:

```
        LIB  file spec
```

Note that no label field is allowed. The LIB will not show up in the output listing, but instead, the assembled source from the file specified will be listed. The LIB operator is similar to a macro call, except the source is found on an external file and no parameter passing is supported. If an END statement is found in the LIB file it will be ignored. When the file has been depleted, the assembler will switch back to the original file and continue where it left off. The LIB operators can not be nested, or in other words, one LIB file may not call another LIB file.

DESCRIPTION OF ASSEMBLER OPERATION

Pass 1 - PASONE ($0293)

Pass 1 is used to build the symbol table which is used to resolve forward references. Pass 1 must be run before PASS 2 and again before PASS 3.

Pass 2 - PASTWO ($02C3)

During pass 2 several things may happen. If the LIST option is on, the assembled source listing is printed with error messages, if any. If the LIST option is off, only offending source lines and their corresponding error messages are printed. If the TAPE option is on, a MIKBUG formatted object record is output (through a different control point than the source listing). If the SYMBOL option is on, a sorted symbol table will be printed after the assembled listing (if any). Pass 1 must be run before PASS 2.

Pass 3 PASTHR ($0495)

Pass 3 is used when the user does not have a "punch" device, on which to save the MIKBUG formatted records, which operates independently fromthe list device. Pass 3 is identical in operation to pass 2 except that NOSYM, NOLIST, and TAPE options are forced and error messages are suppressed. Pass 1 must be run before PASS 3, PASS 2 and PASS 3 are independent.

Initialization

There exists in the assembler an initialization routine for each of the passes which must be run once before that pass is run. These are called P1INIT, P2INIT, and P3INIT, for passes 1, 2, and 3, respectively.

ADAPTING TO YOUR SYSTEM

Due to the inherent flexibility of this assembler it is necessary that each user cutomize it to fit the particular system. This involves very few changes and can be made by any individual familiar with 6800 assembly language. Each point to be adapted is explained below.

Tape Output Character Routine

The address at $020B must be changed to that of your tape punch (or tape record) routine. It is through this control point that the MIKBUG formatted object code is outputted. If you do not have a separate punch or record device, this address may be the same as the Output Character routine address, i.e., tape device same as list device.

Tape Control Characters

There are provisions at $037F and $0383 for four control characters to activate and deactivate, respectively, your punch or record device. Simply place the appropriate control characters for your device in each of the strings. If you desire to send less than four characters, change the byte at $0372 to the appropriate value (even 0). This will, of course, affect both turn on and turn off simultaneously.

Tape Control Delay

The byte at $0388 controls the number of half-seconds (1MHz clock0) of delay between tape turn on and data and also between data and tape turn off. The delay is set now to 2 seconds. If you don't need delay at all set the byte to 00.

Page Control

Page Eject

The four bytes at $1166 are provided for the user to insert the necessary control characters to cause the printer to form feed, i.e., eject to the top of the next page. If you need only 1 character, simply place the 04 after that character in the string. The control character is currently set to $0C (form feed).

Top Margin Control

The byte at $10C4 controls the number of lines from the form feed position to the title and page number line (can be 0).

Page Length Control

The byte at $06A2 controls the number of lines to be printed on each page before the form feed is issued. This count includes the top margin and the title line and should be larger than (top margin + 1). The user may want to alter other features such as the number of columns printed in the symbol table, etc. Most modifications of this type will be needed by only a few users and therefore will not be elaborated upon here. These users are encouraged to study the code to facilitate making the desired modifications.

ASSEMBLER DATA POINTERS

Before calling any assembler routines the user must set several pointers to data areas. This feature allows much flexibility but restrictions which apply to each pointer are outlined below. No assembler routines modify these pointers. Disk users need not pay any attention to these descriptions.

```
        LBLBEG - $0040
        LBLEND - $0042
```

These are the pointers to the area which wil be used for the label table (symbol table). Each entry (symbol) in the table requires 8 bytes. A large table will result in the Put Label and Find Label routines running faster but the Shell (sort) routine will run slower. A small table will have the opposite effect. Of course, the table needs to be large enough to accomodate the number of symbols in your program. A reasonable formula for determining the size necessary is:

```
        SIZE = N * 8 * 2 = N * 16 bytes
```

where N is an estimate of the number of symbols expected. When the table is full an error message will be inserted in the listing. (The table may not be completely full due to the algorithm used for creating the table - hashing, or scatter storage). It should be noted that the symbol table must be an exact multiple of 8 bytes in length.

If you want a 1K symbol table (a recommended minimum, enough for 60-80 labels) you might set LBLBEG to $2000 and LBLEND to $23FF. Notice that the pointers do point to the actual beginning and end of the table. The disk version automatically sets the size of the symbol table.

```
        SRCBEG - $0044
        SRCEND - $0046
```

These two pointers indicate the beginning and end of the section of source code to be assembled, which may be as small as one line of source. SRCEND must point to the carriage return ($0D) of the last line of the source section to be assembled.

```
         LINBYT - $0048
```

Although not actually a pointer LINBYT is related to the source pointers

It tells the assembler how many bytes to ignore from the carriage return of the previous line (or SRCBEG) before actually processing text. This allows direct output of text editors to be assembled without removing the preceeding line numbers. If you have no line number bytes, set LINBYT to 0.

MEMPTR - $0049

This pointer tells the assembler where in memory, if the MEMORY option is on, to put the assembled object code. Recall that four extra bytes (address and count) are required for each contiguous block of code.

ERROR MESSAGES

This assembler supports 12 error messages which are printed after the offending line. The error messages announce violations of any of the restrictions set forth in this manual and are, therefore, self-explanatory.

Additionally, the byte 'ERRORS' (cleared by P1INIT) will be set if any errors have occurred in any of the passes. Note: The ASCII characters 00 - $0C, $0E - $1F, and $80 - $8F, inclusively are explicitly prohibited from being in any area of the source program with the exception of the bytes which are skipped by the assembler (line number bytes). Their existence will cause undefined results. The remaining ASCII character may appear subject to all of the foregoing restrictions.

ADDITIONAL FEATURES

This assembler supports 2 extra mnemonics namely BHS and BLO which are the logical equivalents of BCC and BCS respectively. However, Branch if Higher or Same and Branch if Lower are much easier to remember and use.