TSC 6800 Mnemonic Assembler
SL68-26

Technical Systems Consultants
Box 2574
West Lafayette, IN 47906

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 M68ØØ assembly language. Those who are not are referred to the "M68ØØ Microprocessor Programming Manual" or the "M68ØØ Programming Reference Manual," both available from your Motorola distributor.

The source language (input) for the TSC 68ØØ 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 Ø. This restriction, of course, does not apply to line numbers, if present.

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 ØD). 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 first column and must be unique.
2) Labels consist of letters (A-Z) and numerals (Ø-9).
3) Every label must begin with a letter (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) which must be followed by a space. The exception to this is number 2, below.
2) Mnemonics such as LDA A and AND B may be written as LDAA and ANDB, respectively. In this case fourth character must be followed by a space.

operand field

1) The operand field may consist of an addressing mode indicator and an 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. (Expressions defined later.)
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 ($2Ø) to DEL ($7F).

Expressions

Expressions consist of combinations of numbers and symbols seperated 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 Ø-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 ($2Ø-$5F) to be used as an operand when preceeded by a single quote.

Base	Prefix	Postfix	Comment
Decimal	none	none	decimal assumed
Binary	%	B	Ø, 1 allowed
Octal	@	O or Q	Ø-7 allowed
Hexadecimal	$	H	Ø-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).

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 M68ØØ mnemonics this assembler supports 11 assembler directives 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	activate 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

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 legal expression. 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 seperated by a comman 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 all 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)
MEM	enable storing of object code in memory
NOM	disable storing of object code in memory (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 seperated by a comma. Some of the consequences and uses of the options will be explained later.

PAG

The PAG operator, if the PAG option is on, causes a page eject and subsequently causes the title (if any) 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 Ø and no title is printed on that page. The PAG operator itself will not appear in the listing.

The usual procedure is to have all the options and the title declaration followed by a PAG be 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 ØØØØ 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.

NAM or TTL

These operators are used to assign a title to be printed at the top of all pages (other than page Ø) 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.

** Description of assembler operation

Pass 1 - PASONE ($03B1)

Pass 1 is used to build the symbol table which is used to resolve forward references. Nothing is printed unless the error limit is exceeded (85). Pass 1 must be run before PASS 2 and again before PASS 3.

Pass 2 - PASTWO ($03D9)

During pass 2 several things may happen.

1) If the LIST option is on, the assembled source listing is printed with error messages, if any.

2) If the LIST option is off only offending source lines and their corresponding error messages are printed.

3) If the TAPE option is on, a MIKBUG formatted object record is outputted (through a different control point than the source listing).

4) If the MEMORY option is on, object code is placed in memory in the following form:

COUNT ADDRESS DATA ... DATA COUNT ADDRESS DATA ... DATA TERM

where ADDRESS is the destination address of the first data following

COUNT is a 16 bit byte count indicating how many data bytes follow

DATA is the actual data

TERM is the record terminator (a COUNT of ØØØØ)

When a count of ØØØØ occurs this signifies the end of the program.

5) If the SYMBOL option is on, a sorted symbol table will be printed after the assembly listing (if any). Pass 1 must be run before PASS 2.

Pass 3 - PASTHR ($05BB)

Pass 3 is used when the user does not have a "punch" device, on which to save the MIKBUG formatted records, which operates independently from the list device. Pass 3 is identical in operation to pass 2 except that NOSYM, NOLIST, NOMEM 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 initiallization routine for each of the passes which must be run once before that pass in 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 customize 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.

Output Character Routine

The address at $0321 must be changed to that of your Output Character routine. This routine must print the ASCII character in the A register whose parity bit (most significant bit) is zero. The B and X registers must not be altered. If you have a printer or a disk you will likely want to specify the address of a program which handles these peripherals as well as the control terminal.

Tape Output Character Routine

The address at $0324 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 seperate 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 $04C0 and $04C4 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 the four characters, change the byte at $04B3 to the appropriate value (even Ø). This will, of course, affect both turn on and turn off simultaneously.

Tape Control Delay

The byte at $04C9 controls the number of half-seconds (1MHz clock) 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 ØØ.

Page Control

Page Eject

The four bytes at $11D1 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 Ø4 after that character in the string. The control character is currently set to $ØA (line feed).

Top Margin Control

The byte at $1143 controls the number of lines from the form feed position to the title and page number line (can be Ø).

Page Length Control

The byte at $07C5 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.

Controller Routine

The routine MAIN ($300) is an example of how to use the assembler subroutines. It assumes the user has no independent punch device and therefore must run PASS 3 in order to output the object code. Also, MAIN assumes the source program resides entirely in RAM and that the necessary pointers (to be described) are set.

Disk users will, of course, want to write their own MAIN routine which will bring in each section of source code and run PASS 1 on each, then bring in each section again and run PASS 2, similarly for PASS 3. Naturally, the initiatization routine for each pass need be run only once before each series of passes of the same type. Be reminded that PASS 1 needs to be run before PASS 2 and again before PASS 3. This procedure will allow assembly of files too large to reside entirely in RAM.

One note of caution: the END operator is not strictly necessary at the end of a program as the pass in effect will terminate at the end of the source area. However, if you are generating object code, only an END statement will flush the code buffer or fix the memory count. Likewise, only an END operator will cause the symbol table to be printed (if SYM is on). The byte ENDFLG ($0058) will be set ($FF) if the END operator occurs, which can be detected by your MAIN routine.

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.

LBLBEG - $ØØ4Ø

LBLEND - $ØØ42

These are the pointers to the area which will 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.)

If you want a 1K symbol table (a recommended minimum, enough for 60-80 labels) you might set LBLBEG to $2ØØØ and LBLEND to $23FF. Notice that the pointers do point to the actual beginning and end of the table.

SRCBEG - $ØØ44

SRCEND - $ØØ46

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 ($ØD) of the last line of the source section to be assembled.

LINBYT - $ØØ48

Although not actually a pointer LINBYT is related to the source pointers. It tells the assembler how many bytes to ignore from 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 Ø.

MEMPTR - $ØØ49

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 PIINIT) will be set if any errors have occurred in any of the passes.

Note: The ASCII characters ØØ - $ØC, $ØE - $1F, and $8Ø - $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 characters may appear subject to all of the foregoing restrictions.

Additional Feature

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.

Final Note

Please be reminded, when using the MEMORY option, that in most cases the object code will not be put in memory where it can be executed. It is up to the user to write the simple routine necessary to move the code to its proper executable location.

Important: The address at $031C is the address to which control returns when the assembly is complete. This should be modified to suit your needs.

**** USING THE TSC EDITOR ****

The TSC Text Editing System and the TSC Mnemonic Assembler have not been written to be used co-resident. It is possible to use them one after the other without reloading the source. Following is the procedure to be used:

1. Load the editor but before running, change BEGPNT (location $0359) presently $1492 to $1600. This moves the starting location of the text. Put a $ØD at location $15FF.
2. Run the editor and create your file.
3. When finished, exit the editor and write down the contents of
   a.) FILBEG ($0097- 0098) Shows the source beginning.
   b.) FILEND ($0099-009A) Shows one past the source end.
4. Load the assembler but before running be sure to set all pointers.
   a.) Symbol Table. limits ($0040-0043)
   b.) Source beginning ($0044-0045) contents of edit FILBEG
   c.) Source ending ($0046-0047) "contents-1" of edit FILEND
       ***** Be sure to subtract one from FILEND !!
   d.) Skip count ($0048) Set this to 03 (3 line no. in editor)
   e.) Memory pointer ($0049) Set if used.
5. Run the assembler.