TASM05

USER'S MANUAL

A Cross Assembler Program for Motorola 68HC05 Microcomputers

The Engineers Collaborative, Inc.
Webite at www.tec-i.com
Email support@tec-i.com

* * * IMPORTANT WARRANTY AND LIABILITY INFORMATION * * *

The Engineers Collaborative, Inc. makes no warranties, expressed or implied for this software. No warranty of fitness for a particular purpose is offered. The user is advised to test the software thoroughly before relying on it. The user assumes the entire risk of using the product. The total liability of The Engineers Collaborative, Inc. is limited to the purchase price of the product, and does not cover any lost profits, special, incidental or consequential damages, or any claim against the purchaser by any party.

* * * SOFTWARE LICENSE STATEMENT * * *

US copyright law and international treaty provisions protect this software. Therefore, you must treat this software *just like a book*, with the following exception. The Engineers Collaborative, Inc. authorizes you to make archival copies of the software for the sole purpose of backing-up our software and protecting your investment from loss. By saying, *just like a book*, The Engineers Collaborative, Inc. means that the software cannot be used by two different people in two different places at the same time.

* * * OTHER * * *

The information contained in this manual has been carefully checked and is believed to be accurate and complete at the time of printing. However, no responsibility is assumed for errors that might appear. The Engineers Collaborative, Inc. reserves the right to make changes to the product and/or the manual at any time without notice. Furthermore, The Engineers Collaborative, Inc. assumes no liability arising out of the use or application of any of its products. No part of this document may be copied or reproduced in any form or by any means without prior written consent of The Engineers Collaborative, Inc.

Web Site at www.tec-i.com
Email support@tec-i.com

TABLE OF CONTENTS

1.0 INTRODUCTION

TASM05 is a 6805/6305 cross assembler program. It is one of an integrated set of software and hardware tools provided by TECI to facilitate the development of 6805 and 6305 single chip microcomputer applications. These tools are designed to run on the IBM family of personal computers under the PC-DOS disk operating system Version 2.1 or higher. These tools also run on true IBM compatible personal computers.

TASM05 takes its input from a PC-DOS text file and creates as output an object code file in either Intel HEX or Motorola S-record format and a listing file in PC-DOS text format.

When invoking TASM05 a number of option switches may be used to control the output generated. A variety of pseudo-operation codes may be included in the source code text file to control the format of the output generated.

This manual is intended as a reference document for the experienced 6805/6305 assembly language programmer. It is assumed that the user is familiar with the 6805 or 6305 instruction set and chip architecture. Less experienced users should become familiar with the information contained in the following publications available from Motorola Semiconductor Products Inc. 3501 Ed Bluestein Blvd., Austin, Texas 78721 or from Hitachi America, Ltd., Semiconductor and IC Division, 2210 O'Toole Avenue, San Jose, CA 95131 (408) 942-1500.

M6805/M146805 FAMILY USERS MANUAL (M6805UM(AD2))
M6805 HMOS M146805 CMOS FAMILY
Microcomputer/Microprocessor User's Manual (M6805UM(AD2))
MOTOROLA SINGLE-CHIP MICROCOMPUTER DATA (DL132R1)
HITACHI 8-BIT SINGLE CHIP MICROCOMPUTER DATA BOOK (#U71)
HITACHI HD6305 APPLICATIONS NOTES (#U66)

*** * * CAUTION * * * CAUTION * * * CAUTION * * ***

TASM05 has no provision for target processor verification! Some of the opcodes supported by TASM05 are not available on all devices for which TASM05 may be used.

TASM05 may be used to develop code for all of the Motorola 6805 family of processors and all of the Hitachi 6305 family.

TASM05 supports the STOP and WAIT opcodes but these instructions can **not** be used when writing code for the Motorola NMOS 6805 devices.

TASM05 supports the MUL (multiply) opcode but this instruction is only available on the Motorola HCMOS devices.

TASM05 supports the DAA (decimal adjust accumulator) opcode but this instruction is only available on the Hitachi devices.

It is up to the user to assure that the above opcodes are used only for target processors that support them!

2.0 PACKING LIST

You should find these items in the shipping carton:

1. Manual

2. Software Diskette (found in the pocket of the manual)
   Files on the Diskette:
   ```
   TASM05.EXE
   DEMO.ASM
   HEMO.ASM
   ```

3. Warranty Registration (found with the manual)

If any of these items are missing, please call TECI to obtain them.

3.0 SYSTEM REQUIREMENTS

The TECI development tools require an IBM PC, either ISA (Industry Standard Architecture as in the IBM PC XT, AT or compatibles), EISA (Extended Industry Standard Architecture as commonly found in IBM PC compatibles) or MCA (Micro Channel Architecture as found in the IBM PS2). At least 256k RAM memory, one 5 1/4 inch disk drive and PC-DOS Version 2.1 or higher or the equivalent MS-DOS version operating system.

4.0 CREATING A WORKING SOFTWARE ENVIRONMENT

TECI 68HC05 software development tools are supplied on a 5 1/4 inch double sided double density floppy diskette.

The files supplied on this diskette are as follows:

TASM05.EXE	This is the 6805 cross assembler program.
DEMO.ASM	This is a MC68705P3 source code file that can be used to familiarize 1st time users with the format of TASM05 source files for the MC68705P3.
HEMO.ASM	This is a HD63705 source code file that can be used to familiarize 1st time users with the format of TASM05 source files for the HITACHI HD63705VOP.

It is assumed that the developer will be working on a computer with a hard disk. Though all the tools included in this package may be run on a 8088 PC XT class computer with one 5.25" floppy disk drive, inexpensive modern computers are available that will provide a more flexible working environment.

For the XT user with one or two 5.25" floppy drives, make at least one backup copy of the TECI diskette and at least one working copy of the same diskette. To make a backup, use the DISKCOPY command. See your DOS manual for more information. To create a working copy of the diskette, format a diskette with the /s switch to make a bootable floppy. Copy all the files from your backup of the TECI diskette onto the bootable floppy disk and the working copy of the diskette should be ready to use.

For anyone with a hard disk (including the XT user) it is suggested that you work from the hard disk to speed up operations. Create a directory on the hard disk and copy all the files on the TECI diskette to that directory.

For ease of use, it is suggested that (for users of DOS version 3.x and higher) the directory containing the TECI software be added to the PATH in the AUTOEXEC.BAT file in the ROOT directory on the hard disk. This will enable the user to start up TASM05.EXE from any directory that is the current directory. To enable the PATH to the TASM05.EXE the computer must be rebooted. (Read your DOS manual for more information about the PATH command.)

It must be stressed that new users read the manuals. Even the most experienced programmer may not be able to intuitively grasp every aspect of the TECI environment, though it is simple enough to use after a short acquaintance.

5.0 RUNNING TASM05

To invoke TASM05, type at the DOS ready prompt:

```
[d:]TASM05 [d:]fname.asm [option 1] [option 2] . . .
```

[d:] denotes optional drive specifiers where the TASM05 program or the source file is to be found and [option] denotes optional output control switches. The output files will always be directed to the currently logged drive.

fname.asm is the source code filename. There is no default source file extension so the extension, if any, must be included when invoking TASM05.

Any number of output control options may follow the source file name and these options can be in any order. Options must be separated by one or more spaces.

Normally, TASM05 generates two output files, a listing file and an object code file. These files have the same name as the source code file with different file extensions. The listing file has extension '.PRN' and the object code file has extension '.MIK' or optionally '.HEX'. Object code files with extension '.MIK' are in Motorola S record format and object code files with extension '.HEX' are in Intel HEX format. The default format is '.MIK'

TASM05 is a two pass assembler, it must read the source file twice to complete the generation of object code. On the first pass through the source file statements TASM05 does only the work required to associate values with the symbols that appear in the source file. The output files are generated on the second pass. If errors are found on the first pass, error messages are displayed on the screen and TASM05 terminates before attempting the second pass.

5.1 Output Control Switches

The output control switches are 3 or 4 letter mnemonics that are typed on the command line following the source code filename when invoking TASM05. These switches control the output generated by TASM05 as follows:

NLF NO LISTING FILE - NLF can be used to prevent the generation of the listing. This switch might be used when first assembling a source code file to see if the file assembles without errors.

NOF NO OBJECT FILE - NOF can be used to prevent the generation of the object code file. It might be used in the same manner as NLF.

NST NO SYMBOL TABLE - NST prevents the generation of the symbol table. The symbol table is an alphabetical listing of all symbols and their associated values. The symbol table is appended to the end of the listing file.

LTP LIST TO PRINTER - LTP can be used to send a copy of the listing file to the printer.

HEX Changes the object code file format to INTEL HEX.

WOE WAIT ON ERROR - WOE is used to prevent error messages from scrolling off the screen before they can be read. When used, assembly is halted after an error message has been sent to the screen until the operator presses any key.

SPF SUPPRESS PAGE FORMATTING - This switch is used to suppress normal page formatting. TASM05 normally inserts a form feed character into the listing file after each 64 lines (or after the number of lines specified by the PGLEN pseudo-op) then it inserts

5 lines of page header information into the listing file. See section 5.5. The SPF switch is used to prevent this page formatting.

5.2 Examples

Some examples of invoking TASM05 follow. The examples assume that a source code file named WIDGET.ASM exists.

*Type at the DOS ready prompt:

```
TASM05 WIDET.ASM   or   TASM05 widget.asm
```

In this example no output control switches are used therefore a listing file will be created on the currently logged drive named WIDGET.PRN and an object code file will be created in Motorola S record format named WIDGET.MIK.

*If we had typed at the DOS ready prompt:

```
TASM05 WIDGET.ASM HEX LTP
```

a listing file named WIDGET.PRN and an object code file named WIDGET.HEX in Intel HEX format would have been created and a copy of the listing file would have been sent to the printer.

*If we had typed:

```
TASM05 WIDGET.ASM NOF NLF WOE
```

no output files would be created and if errors were detected in the source file the program would pause after displaying the error message until the operator pressed any key.

6.0 TASM05 SOURCE FILES

Source code files compatible with TASM05 are standard PC-DOS ASCII text files. The only control codes that can be present in the files are carriage return (CR), line feed (LF), and horizontal tab characters.

These files may be created by the EDLIN text editor supplied with DOS but it is recommended that a more sophisticated editor such as Personal Editor from IBM, Wordstar used in the text mode, the Sidekick notepad editor, etc be used.

The reader is encouraged to study the example listing in Appendix A to acquire an understanding of source code syntax and format acceptable to TASM05.

Source files consist of a series of one line statements in the following forms where [] denotes optional statement fragments:

Field 1	**Field 2**	**Field 3**	**Field 4**
[*comment] or [;comment]			
	MNEMONIC	[arguments]	[comment]
symbol	MNEMONIC	[arguments]	[comment]
	PSEUDO-OP	[arguments]	[comment]
symbol	PSEUDO-OP	[arguments]	[comment]

MNEMONIC is any standard mnemonic for a 6805/6305 operation code and must not start in the first column of a statement. Mnemonics can be either upper or lower case characters or combinations of both. BRCLR and BrClr are considered the same by TASM05.

PSEUDO-OP is any TASM05 pseudo-operation and must not start in the first column of a statement. TASM05 pseudo-ops are described later in this document. They can be either upper or lower case characters or combinations of both.

Symbols, if present, must start in column 1 of the statement, consist of 1 to 15 ASCII alphanumeric characters and begin with a letter, a period, or an underscore. Capital letters are significant in symbol names. That is the assembler will treat 'Dog', 'dog' and 'doG' as unique symbols.

Comments must be preceded by a semicolon (;) or an asteric (*) when they appear on a line by themselves. Comments that appear as the last field of a line do not have to be preceded by a semicolon or asteric but must be preceded by a space or horizontal tab character and can not start with an expression operator character (+ - * /....).

The fields of a statement must be delimited by at least one space or a horizontal tab character.

The arguments portion of a statement consists of from 1 to 3 symbols, numeric constants, character constants, expressions or combinations depending on the MNEMONIC or PSEUDO-OP.

Constants consist of a sequence of characters optionally preceded by a radix specifier. The default radix is ten. A radix of 16 may be specified by a leading $ or by a trailing H. If a trailing H is used the first digit of the constant must be a decimal digit.

As an example the numeric constant 15 may be specified as 15, $F, or as 0FH.

Constant examples:

```
One_hundred   EQU   100         a decimal constant
One_hundred   EQU   $64         a hexadecimal constant
One_hundred   EQU   @144        an octal constant
One_hundred   EQU   %1100100    a binary constant
Two_hundred   EQU   0C8H        another form of hex constant
```

Character constants are enclosed in single or double quotes (' or ").

As an example, the statement:

```
TEXT    DB    "The quick brown fox."
```

is equivalent to the statement:

```
TEXT    DB    'The quick brown fox.'
```

7.0 EXPRESSIONS

Expressions may consist of symbols, constants or the characters * or $ (* or $ denote the current value of the program counter) joined together by one of the operators: +-*/%&|^. The operators are the same as in C:

+	add
-	subtract
*	multiply
/	divide
%	remainder after division
&	bitwise logical and
|	bitwise logical or
^	bitwise logical exclusive or

Expressions are evaluated left to right and there is no provision for parenthesized expressions. Arithmetic is carried out in signed two's compliment integer precision (16 bits on the IBM PC).

Constants are constructed as follows:

'	followed by an ASCII character
$	followed by hexadecimal constant
@	followed by an octal constant
%	followed by a binary constant
digits	decimal constant

Refer to the listing in Appendix A for examples of correct expression syntax.

8.0 TASM05 PSEUDO-OPERATIONS

Pseudo-operations look like normal mnemonics, but instead of creating code, they tell the assembler to do something. They tell the assembler things such as where to start the program, where it ends, to set aside memory for variables, give symbolic names to a numeric value, etc.

8.1 Storage Definition Pseudo-Ops

DEFINE BYTES

```
        DB      Argument [,Argument_2] [,Argument_N]
        or
Symbol  DB      Argument [,Argument_2] [,Argument_N]
```

The define bytes pseudo-op is used to initialize memory locations to the value of the argument. It can also be used to associate a symbolic name to the memory location. The argument must be either a numeric constant or a character string constant. Numeric constants must be a value between 0H (0) and 0FFH (255).

Examples:

```
        DB      $FF
        DB      0B3H
```

```
            DB      255
            DB      'How now brown cow'
symbol      DB      "How now brown cow"
```

DEFINE WORD

```
            DW      Argument [,Argument_2] [,Argument_N]
            or
Symbol      DW      Argument [,Argument_2] [,Argument_N]
```

The define word pseudo-op is used to initialize two consecutive memory locations to the value of the argument. It can also be used to associate a symbolic name to the first of 2 memory locations. The most significant byte is stored first then the least signifigant byte. The argument must be a numeric constant with value between 0H and 0FFFFH (65535).

Examples:

```
            DW      0
            DW      257
symbol      DW      $FFFF
            DW      0FC49H ;this is the same as DB $FC, $49
```

DEFINE STORAGE

```
            DS      Argument
            or
Symbol      DS      Argument
```

The define storage pseudo-op is used to reserve the number of consecutive memory locations specified by the argument. It can also be used to associate a symbolic name to the first of these memory locations. The argument must be a numeric constant with value between 0 and 255. The bytes reserved are not initialized.

Examples:

```
            DS      10      ;reserves 10 bytes of storage
symbol      DS      10H     ;reserves 16 bytes of storage
```

FORM CONSTANT BYTES

```
            FCB     Argument [,Argument_2] [,Argument_N]
            or
Symbol      FCB     Argument [,Argument_2] [,Argument_N]
```

The form constant bytes pseudo-op is used to initialize one or more memory locations to the values specified by the argument. It can also be used to associate a symbolic name to the memory location. The values of the argument must be either numeric constants, character constants or symbol names that have been previously defined. The values of the argument must be separated by commas. Numeric constants must be a value between 0H and 0FFH.

Examples:

```
            FCB     $FF
            FCB     0B3H, %01110000
symbol      FCB     1, 2, 3, 4
```

FORM DOUBLE BYTES

```
            FDB     Argument [,Argument_2] [,Argument_N]
            or
Symbol      FDB     Argument [,Argument_2] [,Argument_N]
```

The form double bytes pseudo-op is used to initialize two or more consecutive memory locations to the values specified by the argument. It can also be used to associate a symbolic name to the memory location. The values of the argument must be either numeric constants, character constants or symbol names that have been previously defined. The values of the argument must be separated by commas. Numeric constants must be a value between 0H and 0FFFFH.

Examples:

```
            FDB     $FFFF
            FDB     0B333H, %1110111001110000
symbol      FDB     1, 2, 3, 4
```

RESERVE MEMORY BYTES

```
            RMB     Argument
            or
Symbol      RMB     Argument
```

The reserve memory bytes pseudo-op is used to reserve the number of consecutive memory locations specified by the argument. It can also be used to associate a symbolic name to the first of these memory locations. The argument must be a numeric constant with value between 0 and 255. The bytes reserved are not initialized.

Examples:

```
            RMB     10      ;reserves 10 bytes of storage
symbol      RMB     10H     ;reserves 16 bytes of storage
```

BLOCK STORE ZEROS

```
            BSZ     Argument
            or
Symbol      BSZ     Argument
```

The block store zeros pseudo-op is used to reserve the number of consecutive memory locations specified by the argument. It can also be used to associate a symbolic name to the first of these memory locations. The argument must be a numeric constant with value between 0 and 255. The bytes reserved are each initialized to zero.

Examples:

```
            BSZ     10      ;reserves 10 bytes of storage initialized to zero
symbol      BSZ     10H     ;reserves 16 bytes of storage initialized to zero
```

ZERO MEMORY BYTES

```
            ZMB     Argument
            or
Symbol      ZMB     Argument
```

The zero memory bytes pseudo-op is the same as the BSZ pseudo-op described above. It is included in TASM05 for compatibility with other assemblers.

FORM CONSTANT CHARACTERS

```
            FCC     Argument
            or
Symbol      FCC     Argument
```

The form constant characters pseudo-op is used to initialize one or more consecutive memory locations to the string characters specified by the argument. The characters of the string must be delimited by (') or (").

Examples:

```
            FCC     'Now is the time for all good men. . .'
Symbol      FCC     "How now brown cow."
```

FILL

```
            FILL    Argument1,Argument2
            or
Symbol      FILL    Argument1,Argument2
```

The fill pseudo-op is used to fill the number of consecutive memory locations specified by argument2 with the value specified by argument1. The arguments must be separated by a comma. The value of argument1 must be between 0 and 255.

Examples:

```
            FILL    $FF, 10  fills ten memory locations with 0FFH
Symbol      FILL    1,3              fills three memory locations with one's.
```

8.2 ORG and END Pseudo-Ops

ORG argument

Set program origin. Sets the program counter to the value of the argument. This statement must precede the first code generating statement in a source file. Additional ORG pseudo-ops may be used throughout the source file to create separate program fragments. The argument may be a numeric constant or an expression as shown in the examples below.

Examples:

```
                ORG     100H            ;sets origin to 256
                ORG     10 * 2          ;sets origin to 20
                ORG     10 / 2          ;sets origin to 5
                ORG     10 - 2          ;sets origin to 8
                ORG     10 + 2          ;sets origin to 12
                ORG     $ + 2           ;sets origin to current value of pc + 2
                ORG     * + 2           ;sets origin to current value of pc + 2
                                        ;Note that either $ or * may be used to
                                        ;specify current program counter value.

memsize         EQU     2 * 1024        ;memsize = 2048

                ORG     memsize - 8     ;sets origin to 2040
```

END argument

The END pseudo-op is an optional statement that can be included as the last statement of the primary source file, (INCLUDE files must not have an END statement). If used, the value of the argument becomes the start address specified by the end record of the object code file. The argument must be a numeric or symbolic constant with value between 0H and 0FFFFH.

Examples:

```
                END     80H

start           EQU     80H
                END     start
```

8.3 Symbolic Constants

symbol EQU argument

The EQU pseudo-op assigns the value of the argument to the symbol name. The argument must be a numeric constant or an expression. Once defined by EQU the symbolic name may be used throughout the source file in place of its value. Once a symbol has been assigned a value by the EQU pseudo-op the same symbol can not be assigned another value elsewhere in the source file. Symbolic constants are constants!

Examples:

```
PORTA     EQU     0
memsize   EQU     2 * 1024

          BSET    7,PORTA
          ORG     memsize - 10
```

8.4 Include Pseudo-OP

The Include pseudo-op causes the specified file to be processed as if its source code statements were present in the main source file at the location of the INCLUDE pseudo-op. Included files must not contain INCLUDE pseudo-ops. Include files can not be nested! The main source file can contain any number of INCLUDE pseudo-ops. Included files must not contain an END pseudo-op. If the file to be included does not reside on the currently logged drive, the filename must contain the drive specification.

Examples:

```
          INCLUDE     B:MYFILE.ASM
          INCLUDE     EQUATES.ASM
          INCLUDE     A:TIMERISR.ASM
```

8.5 Listing Control Pseudo-Ops

PAGE

The PAGE (or optionally just PAG) pseudo-op causes a form feed character to be inserted into the listing output file. This causes the printed listing to skip to the top of the next page.

PGLEN argument

The PGLEN pseudo-op sets the number of lines that will be printed on each page of the listing to the value specified by the argument. The argument must be a numeric constant.

Examples:

```
PGLEN 55
PGLEN 66
```

TITLE 'text'
SBTTL 'text'

Each page of the listing file contains a heading at the top of the page that is of the following form:

blank line
TECI 6805/6305 Cross Assembler Version #.#
blank line
User specified title
User specified subtitle
blank line

The user specifies the text of the title and subtitle by means of the TITLE and SBTTL pseudo-ops as shown in the following examples:

```
TITLE   'Text of users title goes here!'
SBTTL  'The text of the users subtitle goes here!'
```

NOLIST

The NOLIST pseudo-op can be used to prevent certain sections of the source file from appearing in the listing file. All lines of the source file after the NOLIST statement is encountered will not appear in the listing file unless a LIST pseudo-op is encountered.

LIST

The LIST pseudo-op is used to resume the listing of source file lines in the listing file after listing has been stopped with the NOLIST pseudo-op.

8.6 OPT Pseudo-Op

The option (OPT) pseudo-op is included for compatibility with other assemblers. Generally the same functions can be performed with other pseudo-ops. The syntax of the OPT pseudo-op is as follows:

```
OPT    option
```

where the option is any of the following:

LIST or L	same as the LIST pseudo-op
NOLIST or NOL	same as the NOLIST pseudo-op
CYCLES or C	does nothing in TASM05
NOCYCLES or NOC	does nothing in TASM05
CONTCYCLES or CONTC	does nothing in TASM05

Examples:

```
OPT    LIST
OPT    NOLIST
```

8.7 Null Pseudo-Ops

Null pseudo-ops are included in TASM05 for compatibility with other assemblers. TASM05 recognizes them so that errors will not be generated but the program performs no functions when thay are encountered.

The NULL Pseudo-Ops are:

NAME
NAM
SPC
CYCLES
NOCYCLES
CONTCYCLES

9.0 6805/6305 ADDRESSING MODES

A summary of 6805/6305 addressing modes and instruction syntax is provided in this section. The reader is referred to the publications listed in the introduction for additional information. Appendix A contains an actual assembly listing of all the 6805/6305 instructions and the addressing modes available for each instruction. This listing should be used to provide more comprehensive examples of proper 6805/6305 source code syntax.

9.1 Inherent Addressing

Inherent addressing instructions contain the operand information implicitly. These are one byte instructions such as:

```
ROLA        ;Rotate the A register left thru carry
ROLX        ;Rotate the X register left thru carry
ASLA        ;Arithmetic shift left the A register
```

9.2 Immediate Addressing

In immediate addressing the operand data is contained in the byte following the opcode byte. Immediate addressing is designated by a leading "#" and the immediate data must have a value in the range 0H - 0FFH (255).

```
LDA   #7        ;load the A register with the number 7.
LDX   #SMALL    ;load the X register with the value of the symbol
                ;SMALL
```

9.3 Direct Addressing

In direct addressing the address of the operand is contained in the byte following the instruction opcode. Since the address in contained in one byte, direct addressing can only be used to access memory locations in the first page of memory (addresses 0 - 255). The operand field must contain a numeric or symbolic constant.

```
LDA     0       ;load the A register with the contents of
                ;memory location 0
LDX     small   ;load the X register with the contents of
                ;the memory location specified by small.
```

9.4 Extended Addressing

In extended addressing the address of the operand is contained in the two bytes following the instruction opcode. Extended addressing can be used to access any memory location. The operand field must contain a numeric or symbolic constant.

```
        LDA     256     ;load the A register with the contents of
                        ;memory location 256
        LDX     large   ;load the X register with the contents of
                        ;the memory location specified by large.
```

9.5 Indexed Addressing

In indexed addressing the contents of the X register is used to calculate the address of the operand. There are 3 types of indexed addressing, indexed - no offset, indexed - one byte offset and indexed two byte offset. The address of the operand is the offset plus the contents of the X register. In the case of indexed - no offset, the contents of the X register is the address of the operand.

```
SMALL   EQU     20
LARGE   EQU     500
DATADB      10

        LDX     #7
        LDA     ,X          ;indexed - no offset, load the A reg
                            ;with the contents of address 7
        LDX     #DATA       ;loads X with the address of DATA
        LDA     ,X          ;loads A with the contents of DATA
                            ; which was initialized to 10
        LDA     SMALL,X     ;indexed - one byte offset, load the
                            ;A reg with the contents of address 27
        LDA     LARGE,X     ;indexed - two byte offset, load the
                            ;A reg with the contents of address 507
```

9.6 Relative Addressing

Relative addressing is used by the branch instructions and by the bit test and branch instructions BRCLR and BRSET. In relative addressing, the last byte of the instruction is added to the program counter to obtain the address of the next instruction if the branch is taken. The target address of the branch must be within -126 to +129 bytes (-125 to +130 for the BRCLR and BRSET inst) of the program counter value following the branch instruction. TASM05 calculates the value of the byte to added to the pc value and generates a **branch out of range** error message if these limits are exceed.

```
        BCS     timer           ;branch to the inst labeled "timer"
                                ;if the carry flag is set
        BRCLR 7,porta,timer     ;branch to the inst labeled "timer" if
            ;bit 7 of porta is a zero.
```

10.0 ERROR MESSAGES

Source files must assemble with no errors before the generated code can be trusted. When no errors are found, TASM05 prints the message

```
Assembly complete . . . errors = 0, warnings = 0.
```

on the video screen and includes this message at the end of the listing file.

When errors are detected a message is displayed on the video screen and an error message is included in the listing file on the line above the line in which the error was detected. Error messages sent to the video screen include a description of the error, the line number of the line in error and the name of the file that contains the line in error. Only one error is detected per line. Before the program ends it prints a message that indicates how many lines contained errors.

TASM05 is a two pass assembler. Pass 1 must be completed without errors before pass 2 is started. Errors detected on pass 1 are displayed on the video screen only since the output files are generated on pass 2. If no errors are detected during pass 1 TASM05 prints the message

```
Starting Pass 2 . . .
```

on the video screen.

10.1 Fatal Errors

Fatal errors are errors serious enough that the assembler can not continue to assemble the source statements. When a fatal error is detected TASM05 prints the error message on the video screen then exits to DOS.

Fatal error messages and causes are listed below.

```
Fatal error - Error - can't open fname
```

This message is displayed when the main source file typed when invoking TASM05 can not be opened. The file name may have been typed incorrectly or the file may not reside on the currently logged drive or the specified drive.

```
FATAL ERROR, couldn't open include file fname
```

This message is displayed when an include file can't be opened. The file name may have been typed incorrectly in the main source file or the file may not reside on the currently logged drive or the specified drive.

```
Fatal error - Error - source file name must follow program name!
```

This message is displayed when the operator fails to type the name of the source code file after TASM05 when invoking the assembler.

```
Fatal error - error writing listing file, check disk space!
Fatal ERROR writing to disk - check disk space. . .
Fatal error - Can't open output listing file - check disk space
Fatal error - Can't open output object code file - check disk space
```

These errors may occur part of the way through an assembly if all of the available disk space on the currently logged drive is used. To correct this error more space will have to be made available on the currently logged drive.

```
Fatal error - No mem for fwd. refs
Fatal error - No space for fwd. refs
```

These errors may occur if your machine does not have enough available memory to store the required information for the forward references in the source file being assembled. To correct the problem the number of forward references in the source file must be reduced. Currently, TASM05 allocates space for 500 forward references.

```
Fatal error - Error in Mnemonic table"
```

If this error appears, the TASM05.EXE program file has been corrupted. A new copy of the program will have to be obtained from a backup copy or from TECI.

10.2 Non-Fatal Errors

Non-fatal errors are errors that are detected but not serious enough to cause termination of the assembly. They most certainly are serious enough to prevent the program under development from performing its intended function however.

A list of non-fatal errors and probable causes are as follows:

These error messages will appear in the listing file on the line above the source statement in which the error was detected. When displayed on the video screen the line number in which the error occurred and the name of file containing the line in error will be printed in front of the error message.

```
Unrecognized Mnemonic
Undefined Pseudo Operation
```

Indicates an invalid opcode field in the source code statement. The opcode or pseudo-op was misspelled or not separated from other fields of the source statement by spaces or tab characters.

```
Symbol Redefined
```

An attempt was made to change the definition of an already defined symbol. This can happen if more than one source statement has been given the same label or symbol name.

```
Illegal Symbol Name
```

The symbol field contains a symbol that contains characters other than the alphanumeric ASCII characters or the symbol name starts with a character other than the alpha characters or the underscore or the period.

```
Out of memory space for the symbol table.
```

Your machine does not contain enough available memory space to contain the symbol table. The number of symbols must be reduced or the amount of available memory increased.

Symbol Undefined on Pass 2

The most probable cause of this error is that a symbol name was referred to in the arguments field of a source statement but never appeared in the symbol field of another source statement.
As an example, if we had the source statement

```
        JMP         WATCHDOG
```

and no source statement had WATCHDOG in the symbol field then we would get an undefined symbol error when TASM05 read the JMP statement on pass 2 of the assembly.

Undefined operand on Pass One

The operands for the EQU and ORG statements must be defined on pass 1 and must occur in the source file before they are used.

Branch out of Range

Target addresses for the relative branch instructions must be within -126 to +129 bytes of the first byte of the branch instruction or within -125 to +130 bytes of a BRCLR or BRSET instruction. This error indicates that the target address of the branch is outside these limits.

Immediate Addressing Illegal
Extended Addressing not allowed
Unknown Addressing Mode

These errors are generated when an illegal or unrecognizeable addressing mode is detected for certain operation codes.

SYNTAX

This is a catch all error message that indicates that the assembler found something that it did not understand but is so confused that it can not give a more specific error message.

Phasing error

Lets hope this error never occurs. It indicates that assumptions made by the assembler on pass 1 were proven invalid on pass 2. If this error occurs check for possible problems related to the size of forward references. Rearrange your code and try again!

```
Missing Delimiter
Missing Delimiter Character
```

These errors occur with the DB or FCC pseudo-ops when the delimiter character is missing from the end of a string constant.

```
EQU requires label
```

An EQU pseudo-op was found that did not have a symbol field. EQU statements must have a symbol with which to equate the value specified by the operand field.

```
Bad fill
```

A FILL pseudo-op was found that did not have the proper operands. The FILL pseudo-op requires two operands separated by a comma.

```
Bit Number must be 0-7
```

The field that specifies the bit number for the BSET, BCLR, BRSET and BRCLR opcodes must specify a number in the range 0-7.

```
Unrecognized OPT
```

An OPT pseudo-op was found with an invalid argument or operand.

10.3 Warning Errors

Warning errors are generated when the assembler finds unexpected values or statement syntax in the source file. The resulting code generated may or may not be correct. Source files should be corrected so that no warnings errors occur before the generated code is trusted.

A list of warning errors and possible causes are as follows:

```
Value Truncated
```

An operand was encountered that was larger than 255 when a byte value was expected. The value was truncated to fit in a byte.

```
Indexed Addressing Assumed
```

Incorrect indexed addressing syntax was found and the indexed addressing mode was assumed.

```
Continuation too long
```

To eliminate this warning make sure that each source statement is complete on one line.

11.0 OUTPUT FILE FORMAT

11.1 Listing file format

Each line of the listing file generated by TASM05 can be a maximum of 132 characters long. The text from each line of the source code file is copied to the corresponding line of the listing file starting at column 16. To create neat appearing printed listing files the user must be aware that each listing file line will be 16 columns longer than the source file lines. If listing files will be printed with 80 column lines then the text of the source file lines should occupy 64 columns or less. If listing files will be printed with 132 column lines then the text of the source file lines should occupy 116 columns or less. It is up to the user to set up his printer for the most appropriate line length.

Columns 2 - 5 of a listing file line contain a 4 character hexadecimal number that corresponds to the current value of the program counter.
Columns 7 - 14 contain the hexadecimal representation of the code generated by TASM05.

Each page of the listing file contains a 6 line heading as explained in the description of the TITLE and SBTTL pseudo-ops.

At the end of the listing file is the alphabetical listing of all symbols and their associated values in hexadecimal representation.

11.2 Motorola S-record Object File Format

Each line of this file has the following format:

Bytes 1 & 2	"S1"	
	3 & 4	The number of data bytes in this record + 3.
	5 & 6	Load Address - high byte
	7 & 8	Load Address - low byte
	9 . . X	Data bytes, 2 characters each
	X+1 & X+2	checksum
	X+3 & X+4	carriage return and linefeed

The last record in the file is the same as above except that it starts with "S9" and the load address field contains the program starting address when specified by an END pseudo-op in the source file. This field contains zero otherwise.

The checksum is the ones-compliment of the 8 bit sum without carry of the byte count, the load address and the data bytes.

11.3 Intel HEX Object File Format

Each line of this file has the following format:

Bytes	1	Colon (:)
	2 & 3	Number of data bytes in this record
	4 & 5	Load Address - high byte
	6 & 7	Load Address - low byte
	8 & 9	Unused - "00"
	10 - X	Data bytes - 2 characters each
	X+1 & X+2	checksum
	X+3 & x+4	carriage return and linefeed

The last line of the file is similar to that described above except the number of data bytes is 0 and the load address is the program starting address if specified by an END pseudo-op in the source file. This field is zero otherwise.

The checksum is the two's compliment of the 8 bit sum without carry of all the data bytes, the load address and the byte count.

APPENDIX A - TEST05 LISTING FILE
TASM05 SOURCE FILE SYNTAX, OPCODES & ADDRESSING MODE EXAMPLES

```
                ;*******************************************************************************
                ;**                      Test File - Fname = TEST05.ASM                       **
                ;**                              10/01/86                                     **
                ;*******************************************************************************
                ; This is a comment line!
                * This is another form of comment line!
                ; The following show the syntax of pseudo-ops whose source lines do not show
                ; up in the listing file. Use as shown without comment char to start the line.
                ;
                ;               INCLUDE A:MYFILE.ASM            ;include MYFILE.ASM
                ;               TITLE   'Title text'            ;specify the page header title
                ;               SBTTL   'Subtitle text'         ;specify the page header subtitle
                ;               PGLEN   64                      ;print 64 lines per page
                ;               NOLIST                          ;turn listing off
                ;               LIST                            ;turn listing on
                ;               OPT     NOLIST                  ;another way to turn listing off
                ;               PAGE                            ;force a page break
                ;

1000            MEMSIZE         EQU     4 * 1024                ;EXPRESSION EXAMPLE
0F38            MOR             equ     $F38                    ;MASK OPTION REGISTER AT 0F38 HEX
1122            WORD            EQU     1122H                   ;16 BIT CONSTANT
0003            BYTE            EQU     3                       ; 8 BIT CONSTANT
0000            PORTA           equ     0                       ;I/O port a address 0
0001            PORTB           equ     PORTA+1                 ;I/O port b address 1
0002            PORTC           equ     PORTA+2                 ;I/O port c address 2
0003            PORTD           equ     PORTA+3                 ;I/O port d address 3
                ;                                               ;expression examples
0001            ONE             EQU     3-2                     this is a comment
0001            ONE_1           EQU     3 - 2                   another comment
0001            One             EQU     1 + 0                   still another comment
0001            One_1           EQU     $10 - 15                still another comment
0006            Six             EQU     3*2                     multiply
0007            SEVEN           EQU     Six+1                   add
000A            TEN             EQU     20/2                    divide
0002            TWO             EQU     8% 3                    remainder after division
0080            _128            EQU     $FF & $80               bitwise AND
0005            FIVE            EQU     4 |1                    bitwise OR
0007            SEVEN_1         EQU     5 ^ 2                   bitwise XOR
0006            Complex         equ     ONE+ONE_1 * Six/TWO     =6 (expressions are evaluated left to right)
                ;
0000                            ORG     0                       ORIGINATE AT ADDRESS 0
                                                                ;EXAMPLES OF STORAGE DEFINITION PSEUDO-OPS
0000 00                         DB      0                       define byte example
0001 0000                       DW      0                       define word example
0003                            DS      2                       define storage example, reserves 2 bytes
0005                            RMB     10                      reserve 10 memory bytes
000F 00010203                   FCB     0,1,2,3                 form constant bytes
0013 00000001                   FDB     0,1,2,3                 form double bytes
     00020003
001B 48656C6C                   FCC     'Hello'                 form constant characters
     6F
0020 48656C6C                   DB      'Hello'                 another way to do it!
     6F
0025 00000000                   ZMB     8                       zero 8 memory bytes
     00000000
002D 00000000                   BSZ     8                       another way to do it!
     00000000
0035 FFFFFFFF                   FILL    0FFH, 10                fill ten bytes with 0ffH
     FFFFFFFF
     FFFF
                ;
0049                            ORG     $ + 10                  ;ORG WITH EXPRESSION
0053                            ORG     * + 10                  another way to do it!
0053 4E6F7720   MESSAGE         DB      'Now is the time for all good men . . .'  a long message
     69732074
     68652074
     696D6520
     666F7220
     616C6C20
     676F6F64
     206D656E
     202E202E
     202E
```

APPENDIX A - TEST05 LISTING FILE
TASM05 SOURCE FILE SYNTAX, OPCODES & ADDRESSING MODE EXAMPLES

```
                ;*******************************************************************************
                ;**THE FOLLOWING IS AN ASSEMBLY OF ALL 6805/6305 OPCODES AND ADDRESSING MODES**
                ;*******************************************************************************
                ;
0080                            ORG     80H                     ;START OF EPROM AREA
                                                                ;ADD MEMORY WITH ACCUM WITH CARRY
0080 A903       START           ADC     #BYTE                   ;IMMEDIATE
```

```
0082 B903                       ADC     BYTE                    ;DIRECT
0084 C91122                     ADC     WORD                    ;EXTENDED
0087 F9                         ADC     ,X                      ;INDEXED 0 BYTE OFFSET
0088 E903                       ADC     BYTE,X                  ;INDEXED 1 BYTE OFFSET
008A D91122                     ADC     WORD,X                  ;INDEXED 2 BYTE OFFSET
008D AB03                       ADD     #BYTE                   ;IMMEDIATE
008F BB03                       ADD     BYTE                    ;DIRECT
0091 CB1122                     ADD     WORD                    ;EXTENDED
0094 FB                         ADD     ,X                      ;INDEXED 0 BYTE OFFSET
0095 EB03                       ADD     BYTE,X                  ;INDEXED 1 BYTE OFFSET
0097 DB1122                     ADD     WORD,X                  ;INDEXED 2 BYTE OFFSET

                                                                ;LOGICAL AND ACCUM WITH MEMORY
009A A403                       AND     #BYTE                   ;IMMEDIATE
009C B403                       AND     BYTE                    ;DIRECT
009E C41122                     AND     WORD                    ;EXTENDED
00A1 F4                         AND     ,X                      ;INDEXED 0 BYTE OFFSET
00A2 E403                       AND     BYTE,X                  ;INDEXED 1 BYTE OFFSET
00A4 D41122                     AND     WORD,X                  ;INDEXED 2 BYTE OFFSET

                                                                ;ARITHMETIC SHIFT LEFT
00A7 48                         ASLA                            ;ACCUMULATOR
00A8 58                         ASLX                            ;INDEX REGISTER
00A9 3803                       ASL     BYTE                    ;DIRECT
00AB 78                         ASL     ,X                      ;INDEXED 0 BYTE OFFSET
00AC 6803                       ASL     BYTE,X                  ;INDEXED 1 BYTE OFFSET

                                                                ;ARITHMETIC SHIFT RIGHT
00AE 47                         ASRA                            ;ACCUMULATOR
00AF 57                         ASRX                            ;INDEX REGISTER
00B0 3703                       ASR     BYTE                    ;DIRECT
00B2 77                         ASR     ,X                      ;INDEXED 0 BYTE OFFSET
00B3 6703                       ASR     BYTE,X                  ;INDEXED 1 BYTE OFFSET

00B5 24FE                       BCC     $                       ;BRANCH IF CARRY CLEAR

00B7 1100                       BCLR    0,PORTA                 ;BIT CLEAR 0 PORTA
00B9 1300                       BCLR    1,PORTA                 ;BIT CLEAR 1 PORTA
00BB 1500                       BCLR    2,PORTA                 ;BIT CLEAR 2 PORTA
00BD 1701                       BCLR    3,PORTB                 ;BIT CLEAR 3 PORTB
00BF 1901                       BCLR    4,PORTB                 ;BIT CLEAR 4 PORTB
00C1 1B01                       BCLR    5,PORTB                 ;BIT CLEAR 5 PORTB
00C3 1D02                       BCLR    6,PORTC                 ;BIT CLEAR 6 PORTC
00C5 1F02                       BCLR    7,PORTC                 ;BIT CLEAR 7 PORTC


00C7 25FE                       BCS     $                       ;BRANCH IF CARRY SET
00C9 27FE                       BEQ     $                       ;BRANCH IF EQUAL
00CB 28FE                       BHCC    $                       ;BRANCH IF HALF CARRY CLEAR
00CD 29FE                       BHCS    $                       ;BRANCH IF HALF CARRY SET
00CF 22FE                       BHI     $                       ;BRANCH IF HIGHER
00D1 24FE                       BHS     $                       ;BRANCH IF HIGHER OR SAME
00D3 2FFE                       BIH     $                       ;BRANCH IF INTERRUPT LINE HIGH
00D5 2EFE                       BIL     $                       ;BRANCH IF INTERRUPT LINE LOW

                                                                ;BIT TEST MEMORY WITH ACCUMULATOR
00D7 A503                       BIT     #BYTE                   ;IMMEDIATE
00D9 B503                       BIT     BYTE                    ;DIRECT
00DB C51122                     BIT     WORD                    ;EXTENDED
00DE F5                         BIT     ,X                      ;INDEXED 0 BYTE OFFSET
00DF E503                       BIT     BYTE,X                  ;INDEXED 1 BYTE OFFSET
00E1 D51122                     BIT     WORD,X                  ;INDEXED 2 BYTE OFFSET
```

APPENDIX A - TEST05 LISTING FILE
TASM05 SOURCE FILE SYNTAX, OPCODES & ADDRESSING MODE EXAMPLES

```
00E4 25FE                       BLO     $                       ;BRANCH IF LOWER
00E6 23FE                       BLS     $                       ;BRANCH IF LOWER OR SAME
00E8 2CFE                       BMC     $                       ;BRANCH IF INTERRUPT MASK CLEAR
00EA 2BFE                       BMI     $                       ;BRANCH IF MINUS
00EC 2DFE                       BMS     $                       ;BRANCH IF INTERRUPT MASK SET
00EE 26FE                       BNE     $                       ;BRANCH IF NOT EQUAL
00F0 2AFE                       BPL     $                       ;BRANCH IF PLUS
00F2 20FE                       BRA     $                       ;BRANCH ALWAYS

00F4 0100FD                     BRCLR   0,PORTA,$               ;BRANCH IF PORTA BIT 0 CLEAR
00F7 0300FD                     BRCLR   1,PORTA,$               ;BRANCH IF PORTA BIT 1 CLEAR
00FA 0500FD                     BRCLR   2,PORTA,$               ;BRANCH IF PORTA BIT 2 CLEAR
00FD 0700FD                     BRCLR   3,PORTA,$               ;BRANCH IF PORTA BIT 3 CLEAR
0100 0900FD                     BRCLR   4,PORTA,$               ;BRANCH IF PORTA BIT 4 CLEAR
0103 0B00FD                     BRCLR   5,PORTA,$               ;BRANCH IF PORTA BIT 5 CLEAR
0106 0D00FD                     BRCLR   6,PORTA,$               ;BRANCH IF PORTA BIT 6 CLEAR
0109 0F00FD                     BRCLR   7,PORTA,$               ;BRANCH IF PORTA BIT 7 CLEAR

010C 21FE                       BRN     $                       ;BRANCH NEVER

010E 0001FD                     BRSET   0,PORTB,$               ;BRANCH IF PORTB BIT 0 SET
0111 0201FD                     BRSET   1,PORTB,$               ;BRANCH IF PORTB BIT 1 SET
0114 0401FD                     BRSET   2,PORTB,$               ;BRANCH IF PORTB BIT 2 SET
0117 0601FD                     BRSET   3,PORTB,$               ;BRANCH IF PORTB BIT 3 SET
011A 0801FD                     BRSET   4,PORTB,$               ;BRANCH IF PORTB BIT 4 SET
011D 0A01FD                     BRSET   5,PORTB,$               ;BRANCH IF PORTB BIT 5 SET
0120 0C01FD                     BRSET   6,PORTB,$               ;BRANCH IF PORTB BIT 6 SET
0123 0E01FD                     BRSET   7,PORTB,$               ;BRANCH IF PORTB BIT 7 SET

0126 1000                       BSET    0,PORTA                 ;BIT SET 0 PORTA
0128 1200                       BSET    1,PORTA                 ;BIT SET 1 PORTA
012A 1400                       BSET    2,PORTA                 ;BIT SET 2 PORTA
012C 1600                       BSET    3,PORTA                 ;BIT SET 3 PORTA
012E 1800                       BSET    4,PORTA                 ;BIT SET 4 PORTA
0130 1A00                       BSET    5,PORTA                 ;BIT SET 5 PORTA
0132 1C00                       BSET    6,PORTA                 ;BIT SET 6 PORTA
0134 1E00                       BSET    7,PORTA                 ;BIT SET 7 PORTA


0136 ADFE                       BSR     $                       ;BRANCH TO SUBROUTINE
0138 98                         CLC                             ;CLEAR CARRY BIT
0139 9A                         CLI                             ;CLEAR INTERRUPT MASK BIT

013A 4F                         CLRA                            ;CLEAR ACCUMULATOR
013B 5F                         CLRX                            ;CLEAR INDEX REGISTER
013C 3F03                       CLR     BYTE                    ;CLEAR MEMORY
013E 7F                         CLR     ,X                      ;CLEAR MEMORY INDEXED 0 BYTE OFFSET
013F 6F03                       CLR     BYTE,X                  ;CLEAR MEMORY INDEXED 1 BYTE OFFSET

                                                                ;COMPARE ACCUMULATOR WITH MEMORY
0141 A103                       CMP     #BYTE                   ;IMMEDIATE
0143 B103                       CMP     BYTE                    ;DIRECT
0145 C11122                     CMP     WORD                    ;EXTENDED
0148 F1                         CMP     ,X                      ;INDEXED 0 BYTE OFFSET
0149 E103                       CMP     BYTE,X                  ;INDEXED 1 BYTE OFFSET
014B D11122                     CMP     WORD,X                  ;INDEXED 2 BYTE OFFSET

014E 43                         COMA                            ;COMPLEMENT ACCUMULATOR
014F 53                         COMX                            ;COMPLEMENT INDEX REGISTER

0150 3303                       COM     BYTE                    ;COMPL MEMORY
0152 73                         COM     ,X                      ;COMPL MEMORY INDEXED 0 BYTE OFFSET
0153 6303                       COM     BYTE,X                  ;COMPL MEMORY INDEXED 1 BYTE OFFSET

                                                                ;COMPARE INDEX REGISTOR WITH MEMORY
0155 A303                       CPX     #BYTE                   ;IMMEDIATE
0157 B303                       CPX     BYTE                    ;DIRECT
0159 C31122                     CPX     WORD                    ;EXTENDED
015C F3                         CPX     ,X                      ;INDEXED 0 BYTE OFFSET
015D E303                       CPX     BYTE,X                  ;INDEXED 1 BYTE OFFSET
015F D31122                     CPX     WORD,X                  ;INDEXED 2 BYTE OFFSET

0162 8D                         DAA                             ;DECIMAL ADJUST ACCUMULATOR
```

APPENDIX A - TEST05 LISTING FILE
TASM05 SOURCE FILE SYNTAX, OPCODES & ADDRESSING MODE EXAMPLES

```
0163 4A                         DECA                            ;DECREMENT ACCUMULATOR
0164 5A                         DECX                            ;DECREMENT INDEX REGISTER
0165 3A03                       DEC     BYTE                    ;DEC MEMORY
0167 7A                         DEC     ,X                      ;DEC MEMORY INDEXED 0 BYTE OFFSET
0168 6A03                       DEC     BYTE,X                  ;DEC MEMORY INDEXED 1 BYTE OFFSET

                                                                ;EXCLUSIVE OR MEMORY WITH ACCUMULATOR
016A A803                       EOR     #BYTE                   ;IMMEDIATE
```

```
016C B803                EOR     BYTE            ;DIRECT
016E C81122              EOR     WORD            ;EXTENDED
0171 F8                  EOR     ,X              ;INDEXED 0 BYTE OFFSET
0172 E803                EOR     BYTE,X          ;INDEXED 1 BYTE OFFSET
0174 D81122              EOR     WORD,X          ;INDEXED 2 BYTE OFFSET

0177 4C                  INCA                    ;INCREMENT ACCUMULATOR
0178 5C                  INCX                    ;INCREMENT INDEX REGISTER
0179 3C03                INC     BYTE            ;INC MEMORY
017B 7C                  INC     ,X              ;INC MEMORY INDEXED 0 BYTE OFFSET
017C 6C03                INC     BYTE,X          ;INC MEMORY INDEXED 1 BYTE OFFSET
                                                 ;JUMP
017E BC03                JMP     BYTE            ;DIRECT
0180 CC1122              JMP     WORD            ;EXTENDED
0183 FC                  JMP     ,X              ;INDEXED 0 BYTE OFFSET
0184 EC03                JMP     BYTE,X          ;INDEXED 1 BYTE OFFSET
0186 DC1122              JMP     WORD,X          ;INDEXED 2 BYTE OFFSET

                                                 ;JUMP TO SUBROUTINE
0189 BD03                JSR     BYTE            ;DIRECT
018B CD1122              JSR     WORD            ;EXTENDED
018E FD                  JSR     ,X              ;INDEXED 0 BYTE OFFSET
018F ED03                JSR     BYTE,X          ;INDEXED 1 BYTE OFFSET
0191 DD1122              JSR     WORD,X          ;INDEXED 2 BYTE OFFSET

                                                 ;LOAD ACCUMULATOR FROM MEMORY
0194 A603                LDA     #BYTE           ;IMMEDIATE
0196 B603                LDA     BYTE            ;DIRECT
0198 C61122              LDA     WORD            ;EXTENDED
019B F6                  LDA     ,X              ;INDEXED 0 BYTE OFFSET
019C E603                LDA     BYTE,X          ;INDEXED 1 BYTE OFFSET
019E D61122              LDA     WORD,X          ;INDEXED 2 BYTE OFFSET

                                                 ;LOAD INDEX REGISTER FROM MEMORY
01A1 AE03                LDX     #BYTE           ;IMMEDIATE
01A3 BE03                LDX     BYTE            ;DIRECT
01A5 CE1122              LDX     WORD            ;EXTENDED
01A8 FE                  LDX     ,X              ;INDEXED 0 BYTE OFFSET
01A9 EE03                LDX     BYTE,X          ;INDEXED 1 BYTE OFFSET
01AB DE1122              LDX     WORD,X          ;INDEXED 2 BYTE OFFSET

                                                 ;LOGICAL SHIFT LEFT
01AE 48                  LSLA                    ;ACCUMULATOR
01AF 58                  LSLX                    ;INDEX REGISTER
01B0 3803                LSL     BYTE            ;MEMORY DIRECT
01B2 78                  LSL     ,X              ;MEMORY INDEXED 0 BYTE OFFSET
01B3 6803                LSL     BYTE,X          ;MEMORY INDEXED 1 BYTE OFFSET

                                                 ;LOGICAL SHIFT RIGHT
01B5 44                  LSRA                    ;ACCUMULATOR
01B6 54                  LSRX                    ;INDEX REGISTER
01B7 3403                LSR     BYTE            ;MEMORY DIRECT
01B9 74                  LSR     ,X              ;MEMORY INDEXED 0 BYTE OFFSET
01BA 6403                LSR     BYTE,X          ;MEMORY INDEXED 1 BYTE OFFSET

01BC 42                  MUL                     ;MULTIPLY A TIMES X
```

```
The Engineers Collaborative, Inc. 6805/6305 Cross Assembler Version 2.3

APPENDIX A - TEST05 LISTING FILE
TASM05 SOURCE FILE SYNTAX, OPCODES & ADDRESSING MODE EXAMPLES

                                                    ;NEGATE
01BD 40                         NEGA                ;ACCUMULATOR
01BE 50                         NEGX                ;INDEX REGISTER
01BF 3003                       NEG     BYTE        ;MEMORY DIRECT
01C1 70                         NEG     ,X          ;MEMORY INDEXED 0 BYTE OFFSET
01C2 6003                       NEG     BYTE,X      ;MEMORY INDEXED 1 BYTE OFFSET

01C4 9D                         NOP                 ;NO OPERATION
                                                    ;INCLUSIVE OR ACCUMULATOR WITH MEMORY
01C5 AA03                       ORA     #BYTE       ;IMMEDIATE
01C7 BA03                       ORA     BYTE        ;DIRECT
01C9 CA1122                     ORA     WORD        ;EXTENDED
01CC FA                         ORA     ,X          ;INDEXED 0 BYTE OFFSET
01CD EA03                       ORA     BYTE,X      ;INDEXED 1 BYTE OFFSET
01CF DA1122                     ORA     WORD,X      ;INDEXED 2 BYTE OFFSET

                                                    ;ROTATE LEFT THRU CARRY
01D2 49                         ROLA                ;ACCUMULATOR
01D3 59                         ROLX                ;INDEX REGISTER
01D4 3903                       ROL     BYTE        ;MEMORY DIRECT
01D6 79                         ROL     ,X          ;MEMORY INDEXED 0 BYTE OFFSET
01D7 6903                       ROL     BYTE,X      ;MEMORY INDEXED 1 BYTE OFFSET

                                                    ;ROTATE RIGHT THRU CARRY
01D9 46                         RORA                ;ACCUMULATOR
01DA 56                         RORX                ;INDEX REGISTER
01DB 3603                       ROR     BYTE        ;MEMORY DIRECT
01DD 76                         ROR     ,X          ;MEMORY INDEXED 0 BYTE OFFSET
01DE 6603                       ROR     BYTE,X      ;MEMORY INDEXED 1 BYTE OFFSET

01E0 9C                         RSP                 ;RESET STACK POINTER
01E1 80                         RTI                 ;RETURN FROM INTERRUPT
01E2 81                         RTS                 ;RETURN FROM SUBROUTINE

                                                    ;SUBTRACT WITH CARRY
01E3 A203                       SBC     #BYTE       ;IMMEDIATE
01E5 B203                       SBC     BYTE        ;DIRECT
01E7 C21122                     SBC     WORD        ;EXTENDED
01EA F2                         SBC     ,X          ;INDEXED 0 BYTE OFFSET
01EB E203                       SBC     BYTE,X      ;INDEXED 1 BYTE OFFSET
01ED D21122                     SBC     WORD,X      ;INDEXED 2 BYTE OFFSET

01F0 99                         SEC                 ;SET CARRY BIT
01F1 9B                         SEI                 ;SET INTERRUPT MASK BIT

                                                    ;STORE ACCUMULATOR IN MEMORY
01F2 B703                       STA     BYTE        ;DIRECT
01F4 C71122                     STA     WORD        ;EXTENDED
01F7 F7                         STA     ,X          ;INDEXED 0 BYTE OFFSET
01F8 E703                       STA     BYTE,X      ;INDEXED 1 BYTE OFFSET
01FA D71122                     STA     WORD,X      ;INDEXED 2 BYTE OFFSET

01FD 8E                         STOP                ;CMOS VERSIONS ONLY

                                                    ;STORE INDEX REGISTER IN MEMORY
01FE BF03                       STX     BYTE        ;DIRECT
0200 CF1122                     STX     WORD        ;EXTENDED
0203 FF                         STX     ,X          ;INDEXED 0 BYTE OFFSET
0204 EF03                       STX     BYTE,X      ;INDEXED 1 BYTE OFFSET
0206 DF1122                     STX     WORD,X      ;INDEXED 2 BYTE OFFSET
                                                    ;SUBTRACT
0209 A003                       SUB     #BYTE       ;IMMEDIATE
020B B003                       SUB     BYTE        ;DIRECT
020D C01122                     SUB     WORD        ;EXTENDED
0210 F0                         SUB     ,X          ;INDEXED 0 BYTE OFFSET
0211 E003                       SUB     BYTE,X      ;INDEXED 1 BYTE OFFSET
0213 D01122                     SUB     WORD,X      ;INDEXED 2 BYTE OFFSET

0216 83                         SWI                 ;SOFTWARE INTERRUPT
0217 97                         TAX                 ;TRANSFER ACCUM TO INDEX REGISTER
```

```
The Engineers Collaborative, Inc. 6805/6305 Cross Assembler Version 2.3

APPENDIX A - TEST05 LISTING FILE
TASM05 SOURCE FILE SYNTAX, OPCODES & ADDRESSING MODE EXAMPLES

                                                          ;TEST FOR NEGITIVE OR ZERO
0218 4D                         TSTA                      ;ACCUMULATOR
0219 5D                         TSTX                      ;INDEX REGISTER
021A 3D03                       TST     BYTE              ;DIRECT
021C 7D                         TST     ,X                ;INDEXED 0 BYTE OFFSET
021D 6D03                       TST     BYTE,X            ;INDEXED 1 BYTE OFFSET


021F 9F                         TXA                       ;TRANSFER INDEX REG TO ACCUM
0220 8F                         WAIT                      ;CMOS VERSIONS ONLY

0F38                            ORG     MOR               ;MASK OPTIONS
0F38 01                         DB      01                ;PROGRAM 01 INTO MOR

0FF8                            ORG     MEMSIZE - 8       ;VECTORS
0FF8 0080                       DW      START             ;TIMER INTERRUPT VECTOR
0FFA 0080                       DW      START             ;EXTERNAL INTERRUPT
0FFC 0080                       DW      START             ;SOFTWARE INTERRUPT
0FFE 0080                       DW      START             ;POWER UP RESET VECTOR
0080                            END     START             ;


Assembly complete - Errors = 0 , Warnings = 0




  ---Symbol Table---
BYTE            0003
Complex         0006
FIVE            0005
MEMSIZE         1000
MESSAGE         0053
MOR             0F38
ONE             0001
ONE_1           0001
One             0001
One_1           0001
PORTA           0000
PORTB           0001
PORTC           0002
PORTD           0003
SEVEN           0007
SEVEN_1         0007
START           0080
Six             0006
TEN             000A
TWO             0002
WORD            1122
_128            0080
```

APPENDIX B -- MC6805 AND MC68HC05 Instruction Set

The table that follows on the next few pages includes all the instructions the MC68HC05 can execute. Below are the meanings for the abbreviations used in the table.

Boolean Expression

=	The symbol to the left of this is loaded with what is the the right.
&	Logical AND
+	Inclusive OR, Addition
+	Exclusive OR
-	Subtraction (two's complement)
*	Multiplication
[ ]	Contents of memory location. Example: [short] means contents of memory location called short.

MCU Registers

A	Accumulator
CC	Condition Code Register
X	Index Register
PC	Program Counter
PCH	PC High Byte
PCL	PC Low Byte
SP	Stack Pointer

Condition Code Symbols

H	Half Carry	Result caused a carry from bit 3 to bit 4	(CC register bit 4)
I	interrupt Mask	When Set interrupts are disabled	(bit 3)
N	Negate (sign)	Result has bit 7 set	(bit 2)
Z	Zero	Result is a zero	(bit 1)
C	Carry/Borrow	Result is too large for one byte	(bit 0)
#	bit is affected		
-	bit is unaffected		
0	bit is cleared		
1	bit is set		

Addressing Modes

Mode	Abbreviation	Operands
Inherent	INH	none
Immediate	IMM	ii
Direct	DIR	dd
Direct(n)	DIR(n)	dd rr
Extended	EXT	hi lo
Indexed no offset	IX	none
Indexed byte offset	IX1	LO
Indexed 2 byte offset	IX2	HI LO
Relative	REL	rr

SOURCE FORMS	OPERATION	BOOLEAN EXPRESSION	ADDRESSING MODES	MACHINE CODE OPCODE	MACHINE CODE OPERAND	BYTES	CYCLES	CONDITION CODES H	I	N	Z	C
ADC #byte	Add with carry	A = A + byte + C	IMM	A9	ii	2	2	#	-	#	#	#
ADC short		A = A + [short] + C	DIR	B9	dd	2	3					
ADC long		A = A + [long] + C	EXT	C9	hi lo	3	4					
ADC long,x		A = A + [long+x] + C	IX2	D9	HI LO	3	5					
ADC short,x		A = A + [short+x] + C	IX1	E9	LO	2	4					
ADC ,x		A = A + [x] + C	IX	F9		1	3					
ADD #byte	Add	A = A + byte	IMM	AB	ii	2	2	#	-	#	#	#
ADD short		A = A + [short]	DIR	BB	dd	2	3					
ADD long		A = A + [long]	EXT	CB	hi lo	3	4					
ADD long,x		A = A + [long+x]	IX2	DB	HI LO	3	5					
ADD short,x		A = A + [short+x]	IX1	EB	LO	2	4					
ADD ,x		A = A + [x]	IX	FB		1	3					
AND #byte	Logical AND	A = A & byte	IMM	A4	ii	2	2	-	-	#	#	-
AND short		A = A & [short]	DIR	B4	dd	2	3					
AND long		A = A & [long]	EXT	C4	hi lo	3	4					
AND long,x		A = A & [long+x]	IX2	D4	HI LO	3	5					
AND short,x		A = A & [short+x]	IX1	E4	LO	2	4					
AND ,x		A = A & [x]	IX	F4		1	3					
ASL short	Arithmatic Shift Left	<---------- 0	DIR	38	dd	2	5	-	-	#	#	#
ASLA		+-++----------+	INH(A)	48		1	3					
ASLX		+-++----------+	INH(X)	58		1	3					
ASL short,x		C 7 6 5 4 3 2 1 0	IX1	68	LO	2	6					
ASL ,x			IX	78		1	5					
ASR short	Arithmatic Shift Right	0 ---------->	DIR	37	dd	2	5	-	-	#	#	#
ASRA		+----------++-+	INH(A)	47		1	3					
ASRX		+----------++-+	INH(X)	57		1	3					
ASR short,x		7 6 5 4 3 2 1 0 C	IX1	67	LO	2	6					
ASR ,x			IX	77		1	5					
BCC rel	Branch if Carry Clear	? C = 0	REL	24	rr	2	3	-	-	-	-	-
BCLR 0,byte	Clear Bit n in memory location byte	Bit 0 of [byte] = 0	DIR(0)	11	dd	2	5	-	-	-	-	-
BCLR 1,byte		Bit 1 of [byte] = 0	DIR(1)	13	dd	2	5					
BCLR 2,byte		Bit 2 of [byte] = 0	DIR(2)	15	dd	2	5					
BCLR 3,byte		Bit 3 of [byte] = 0	DIR(3)	17	dd	2	5					
BCLR 4,byte		Bit 4 of [byte] = 0	DIR(4)	19	dd	2	5					
BCLR 5,byte		Bit 5 of [byte] = 0	DIR(5)	1B	dd	2	5					
BCLR 6,byte		Bit 6 of [byte] = 0	DIR(6)	1D	dd	2	5					
BCLR 7,byte		Bit 7 of [byte] = 0	DIR(7)	1F	dd	2	5					
BCS rel	Branch if Carry Set	? C = 1	REL	25	rr	2	3	-	-	-	-	-
BEQ rel	Branch if Equal	? Z = 1	REL	27	rr	2	3	-	-	-	-	-
BHCC rel	Branch if Half Carry Clear	? H = 0	REL	28	rr	2	3	-	-	-	-	-
BHCS rel	Branch if Half Carry Set	? H = 1	REL	29	rr	2	3	-	-	-	-	-
BHI rel	Branch if Higher	? (C+Z) = 0	REL	22	rr	2	3	-	-	-	-	-
BHS rel	Branch if Higher or Same	? C = 0	REL	24	rr	2	3	-	-	-	-	-
BIH rel	Branch if !IRQ pin is High	? !IRQ pin = 1	REL	2F	rr	2	3	-	-	-	-	-

SOURCE FORMS	OPERATION	BOOLEAN EXPRESSION	ADDRESSING MODES	MACHINE CODE		BYTES	CYCLES	CONDITION CODES				
				OPCODE	OPERAND			H	I	N	Z	C
BIL rel	Branch if !IRQ pin is Low	? !IRQ pin = 0	REL	2E	rr	2	3	-	-	-	-	-
BIT #byte	Bit test Memory with A	A & byte	IMM	A5	ii	2	2	-	-	#	#	-
BIT short		A & [short]	DIR	B5	dd	2	3					
BIT long		A & [long]	EXT	C5	hi lo	3	4					
BIT long,x		A & [long+x]	IX2	D5	HI LO	3	5					
BIT short,x		A & [short+x]	IX1	E5	LO	2	4					
BIT ,x		A & [x]	IX	F5		1	3					
BLO rel	Branch if Lower	? C = 1	REL	25	rr	2	3	-	-	-	-	-
BLS rel	Branch if Lower or Same	? (C + Z) = 1	REL	23	rr	2	3	-	-	-	-	-
BMC rel	Branch if I bit is Clear	? I = 0	REL	2C	rr	2	3	-	-	-	-	-
BMI rel	Branch if Minus	? N = 1	REL	2B	rr	2	3	-	-	-	-	-
BMS rel	Branch if I bit is Set	? I = 1	REL	2D	rr	2	3	-	-	-	-	-
BNE rel	Branch if Not Equal	? Z = 0	REL	26	rr	2	3	-	-	-	-	-
BPL rel	Branch if Plus	? N = 0	REL	2A	rr	2	3	-	-	-	-	-
BRA rel	Branch Always	? 1 = 1	REL	20	rr	2	3	-	-	-	-	-
BRCLR 0,byte,rel	Branch if bit N of byte is clear	? bit 0 of [byte] = 0	DIR(b0)	01	dd rr	3	5	-	-	-	-	#
BRCLR 1,byte,rel		? bit 1 of [byte] = 0	DIR(b1)	03	dd rr	3	5					
BRCLR 2,byte,rel		? bit 2 of [byte] = 0	DIR(b2)	05	dd rr	3	5					
BRCLR 3,byte,rel		? bit 3 of [byte] = 0	DIR(b3)	07	dd rr	3	5					
BRCLR 4,byte,rel		? bit 4 of [byte] = 0	DIR(b4)	09	dd rr	3	5					
BRCLR 5,byte,rel		? bit 5 of [byte] = 0	DIR(b5)	0B	dd rr	3	5					
BRCLR 6,byte,rel		? bit 6 of [byte] = 0	DIR(b6)	0D	dd rr	3	5					
BRCLR 7,byte,rel		? bit 7 of [byte] = 0	DIR(b7)	0F	dd rr	3	5					
BRN rel	Branch Never	? 1 = 0	REL	21	rr	2	3	-	-	-	-	-
BRSET 0,byte,rel	Branch if bit N of byte is Set	? bit 0 of [byte] = 1	DIR(b0)	00	dd rr	3	5	-	-	-	-	#
BRSET 1,byte,rel		? bit 1 of [byte] = 1	DIR(b1)	02	dd rr	3	5					
BRSET 2,byte,rel		? bit 2 of [byte] = 1	DIR(b2)	04	dd rr	3	5					
BRSET 3,byte,rel		? bit 3 of [byte] = 1	DIR(b3)	06	dd rr	3	5					
BRSET 4,byte,rel		? bit 4 of [byte] = 1	DIR(b4)	08	dd rr	3	5					
BRSET 5,byte,rel		? bit 5 of [byte] = 1	DIR(b5)	0A	dd rr	3	5					
BRSET 6,byte,rel		? bit 6 of [byte] = 1	DIR(b6)	0C	dd rr	3	5					
BRSET 7,byte,rel		? bit 7 of [byte] = 1	DIR(b7)	0E	dd rr	3	5					
BSET 0,byte	Set Bit n in memory location byte	Bit 0 of [byte] = 1	DIR(0)	10	dd	2	5	-	-	-	-	-
BSET 1,byte		Bit 1 of [byte] = 1	DIR(1)	12	dd	2	5					
BSET 2,byte		Bit 2 of [byte] = 1	DIR(2)	14	dd	2	5					
BSET 3,byte		Bit 3 of [byte] = 1	DIR(3)	16	dd	2	5					
BSET 4,byte		Bit 4 of [byte] = 1	DIR(4)	18	dd	2	5					
BSET 5,byte		Bit 5 of [byte] = 1	DIR(5)	1A	dd	2	5					
BSET 6,byte		Bit 6 of [byte] = 1	DIR(6)	1C	dd	2	5					
BSET 7,byte		Bit 7 of [byte] = 1	DIR(7)	1E	dd	2	5					

SOURCE FORMS	OPERATION	BOOLEAN EXPRESSION	ADDRESSING MODES	MACHINE CODE		BYTES	CYCLES	CONDITION CODES				
				OPCODE	OPERAND			H	I	N	Z	C
BSR rel	Branch to SubRoutine	PC = PC + 2 [SP] = PCL;SP = SP - 1 [SP] = PCH;SP = SP - 1 PC = PC + rel	REL	AD	rr	2	6	-	-	-	-	-
CLC	CLear Carry bit	C bit = 0	INH	98		1	2	-	-	-	-	0
CLI	CLear I bit	I bit = 0	INH	9A		1	2	-	0	-	-	-
CLR short	CLeaR	[short] = 0	DIR	3F	dd	2	5	-	-	0	1	-
CLRA		A = 0	INH(A)	4F		1	3					
CLRX		X = 0	INH(X)	5F		1	3					
CLR short,x		[short+x] = 0	IX1	6F	LO	2	6					
CLR ,x		[x] = 0	IX	7F		1	5					
CMP #byte	CoMPare	A - byte	IMM	A1	ii	2	2	-	-	#	#	#
CMP short	A with	A - [short]	DIR	B1	dd	2	3					
CMP long	memory	A - [long]	EXT	C1	hi lo	3	4					
CMP long,x		A - [long+x]	IX2	D1	HI LO	3	5					
CMP short,x		A - [short+x]	IX1	E1	LO	2	4					
CMP ,x		A - [x]	IX	F1		1	3					
COM short	1's	[short] = $FF - [short]	DIR	33	dd	2	5	-	-	#	#	1
COMA	Complement	A = $FF - A	INH(A)	43		1	3					
COMX		X = $FF - X	INH(X)	53		1	3					
COM short,x		[short+x]=$FF-[short+x]	IX1	63	LO	2	6					
COM ,x		[x] = $FF - [x]	IX	73		1	5					
CPX #byte	ComPare	X - byte	IMM	A3	ii	2	2	-	-	#	#	#
CPX short	X with	X - [short]	DIR	B3	dd	2	3					
CPX long	memory	X - [long]	EXT	C3	hi lo	3	4					
CPX long,x		X - [long+x]	IX2	D3	HI LO	3	5					
CPX short,x		X - [short+x]	IX1	E3	LO	2	4					
CPX ,x		X - [x]	IX	F3		1	3					
DEC short	DECrement	[short] = $FF - [short]	DIR	3A	dd	2	5	-	-	#	#	-
DECA		A = $FF - A	INH(A)	4A		1	3					
DECX (or DEX)		X = $FF - X	INH(X)	5A		1	3					
DEC short,x		[short+x]=$FF-[short+x]	IX1	6A	LO	2	6					
DEC ,x		[x] = $FF - [x]	IX	7A		1	5					
EOR #byte	Exclusive	A + byte	IMM	A8	ii	2	2	-	-	#	#	-
EOR short	Or A with	A + [short]	DIR	B8	dd	2	3					
EOR long	memory	A + [long]	EXT	C8	hi lo	3	4					
EOR long,x		A + [long+x]	IX2	D8	HI LO	3	5					
EOR short,x		A + [short+x]	IX1	E8	LO	2	4					
EOR ,x		A + [x]	IX	F8		1	3					
INC short	INCrement	[short] = [short] + 1	DIR	3C	dd	2	5	-	-	#	#	-
INCA		A = A + 1	INH(A)	4C		1	3					
INCX (or INX)		X = X + 1	INH(X)	5C		1	3					
INC short,x		[short+x]=[short+x] + 1	IX1	6C	LO	2	6					
INC ,x		[x] = [X] + 1	IX	7C		1	5					
JMP short	JuMP	PC = short	DIR	BC	dd	2	2	-	-	-	-	-
JMP long		PC = long	EXT	CC	hi lo	3	3					
JMP long,x		PC = [long+x]	IX2	DC	HI LO	3	4					
JMP short,x		PC = [short+x]	IX1	EC	LO	2	3					
JMP ,x		PC = [x]	IX	FC		1	2					
JSR short	Jump	PC = PC + BYTES	DIR	BD	dd	2	5	-	-	-	-	-
JSR long	to	[SP] = PCL;SP = SP - 1	EXT	CD	hi lo	3	6					
JSR long,x	Subroutine	[SP] = PCH;SP = SP - 1	IX2	DD	HI LO	3	7					
JSR short,x		PC = as shown in JMP	IX1	ED	LO	2	6					
JSR ,x			IX	FD		1	5					

SOURCE FORMS	OPERATION	BOOLEAN EXPRESSION	ADDRESSING MODES	MACHINE CODE OPCODE	MACHINE CODE OPERAND	BYTES	CYCLES	CONDITION CODES H	I	N	Z	C
LDA #byte	LoaD A	A = byte	IMM	A6	ii	2	2	-	-	#	#	-
LDA short		A = [short]	DIR	B6	dd	2	3					
LDA long		A = [long]	EXT	C6	hi lo	3	4					
LDA long,x		A = [long+x]	IX2	D6	HI LO	3	5					
LDA short,x		A = [short+x]	IX1	E6	LO	2	4					
LDA ,x		A = [x]	IX	F6		1	3					
LDX #byte	LoaD X	X = byte	IMM	AE	ii	2	2	-	-	#	#	-
LDX short		X = [short]	DIR	BE	dd	2	3					
LDX long		X = [long]	EXT	CE	hi lo	3	4					
LDX long,x		X = [long+x]	IX2	DE	HI LO	3	5					
LDX short,x		X = [short+x]	IX1	EE	LO	2	4					
LDX ,x		X = [x]	IX	FE		1	3					
LSL short	Logical	<---------0	DIR	38	dd	2	5	-	-	#	#	#
LSLA	Shift	+-++---------+	INH(A)	48		1	3					
LSLX	Left	+-++---------+	INH(X)	58		1	3					
LSL short,x		C 7 6 5 4 3 2 1 0	IX1	68	LO	2	6					
LSL ,x			IX	78		1	5					
LSR short	Logical	0--------->	DIR	34	dd	2	5	-	-	0	#	#
LSRA	Shift	+---------++-+	INH(A)	44		1	3					
LSRX	Right	+---------++-+	INH(X)	54		1	3					
LSR short,x		7 6 5 4 3 2 1 0 C	IX1	64	LO	2	6					
LSR ,x			IX	74		1	5					
MUL	unsigned MULtiply	X:A = X * A	INH	42		1	11	0	-	-	-	0
NEG short	NEGate	[short] = 0 - [short]	DIR	30	dd	2	5	-	-	#	#	#
NEGA	(2's	A = 0 - A	INH(A)	40		1	3					
NEGX	complement)	X = 0 - X	INH(X)	50		1	3					
NEG short,x		[short+x] = -[short+x]	IX1	60	LO	2	6					
NEG ,x		[x] = 0 - [x]	IX	70		1	5					
NOP	No Operation		INH	9D		1	2	-	-	-	-	-
ORA #byte	OR	A = A + byte	IMM	AA	ii	2	2	-	-	#	#	-
ORA short	with A	A = A + [short]	DIR	BA	dd	2	3					
ORA long		A = A + [long]	EXT	CA	hi lo	3	4					
ORA long,x		A = A + [long+x]	IX2	DA	HI LO	3	5					
ORA short,x		A = A + [short+x]	IX1	EA	LO	2	4					
ORA ,x		A = A + [x]	IX	FA		1	3					
ROL short	ROtate	+--------- > ---------+	DIR	39	dd	2	5	-	-	#	#	#
ROLA	Left	_+-++---------------+_	INH(A)	49		1	3					
ROLX	through	+-++---------------+	INH(X)	59		1	3					
ROL short,x	carry	C 7 6 5 4 3 2 1 0	IX1	69	LO	2	6					
ROL ,x			IX	79		1	5					
ROR short	ROtate	+-------- < ----------+	DIR	36	dd	2	5	-	-	#	#	#
RORA	Right	_+---------------++-+-+_	INH(A)	46		1	3					
RORX	through	+---------------++-+	INH(X)	56		1	3					
ROR short,x	carry	7 6 5 4 3 2 1 0 C	IX1	66	LO	2	6					
ROR ,x			IX	76		1	5					
RSP	Reset Stack Pointer	SP = $00FF	INH	9C		1	2	-	-	-	-	-
RTI	ReTurn from Interrupt	SP = SP +1; CC = [SP] SP = SP +1; A = [SP] SP = SP +1; X = [SP] SP = SP +1; PCH = [SP] SP = SP +1; PCL = [SP]	INH	80		1	9	#	#	#	#	#
								LOADED FROM STACK				
RTS	ReTurn from Subroutine	SP = SP +1; PCH = [SP] SP = SP +1; PCL = [SP]	INH	81		1	6	-	-	-	-	-

SOURCE FORMS	OPERATION	BOOLEAN EXPRESSION	ADDRESSING MODES	MACHINE CODE		BYTES	CYCLES	CONDITION CODES				
				OPCODE	OPERAND			H	I	N	Z	C
SBC #byte	Subtract	A = A - byte - C	IMM	A2	ii	2	2	-	-	#	#	#
SBC short	with	A = A - [short] - C	DIR	B2	dd	2	3					
SBC long	Carry	A = A - [long] - C	EXT	C2	hi lo	3	4					
SBC long,x		A = A - [long+x] - C	IX2	D2	HI LO	3	5					
SBC short,x		A = A - [short+x] - C	IX1	E2	LO	2	4					
SBC ,x		A = A - [x] - C	IX	F2		1	3					
SEC	Set C bit	C bit = 1	INH	99		1	2	-	-	-	-	1
SEI	Set I bit	I bit = 1	INH	9B		1	2	-	1	-	-	-
STA short	STore	[short] = A	DIR	B7	dd	2	4	-	-	#	#	-
STA long	A	[long] = A	EXT	C7	hi lo	3	5					
STA long,x		[long+x] = A	IX2	D7	HI LO	3	6					
STA short,x		[short+x] = A	IX1	E7	LO	2	5					
STA ,x		[x] = A	IX	F7		1	4					
STOP	STOP clocks	I bit = 0 clocks stopped	INH	8E		1	2	-	0	-	-	-
STX short	STore	[short] = X	DIR	BF	dd	2	4	-	-	#	#	-
STX long	X	[long] = X	EXT	CF	hi lo	3	5					
STX long,x		[long+x] = X	IX2	DF	HI LO	3	6					
STX short,x		[short+x] = X	IX1	EF	LO	2	5					
STX ,x		[x] = X	IX	FF		1	4					
SUB #byte	Subtract	A = A - byte	IMM	A0	ii	2	2	-	-	#	#	#
SUB short		A = A - [short]	DIR	B0	dd	2	3					
SUB long		A = A - [long]	EXT	C0	hi lo	3	4					
SUB long,x		A = A - [long+x]	IX2	D0	HI LO	3	5					
SUB short,x		A = A - [short+x]	IX1	E0	LO	2	4					
SUB ,x		A = A - [x]	IX	F0		1	3					
SWI	SoftWare Interrupt	PC = PC + 1 [SP] = PCL; SP = SP -1 [SP] = PCH; SP = SP -1 [SP] = X; SP = SP - 1 [SP] = A; SP = SP - 1 [SP] = CC; SP = SP - 1 I bit = 1 PCH = swi_vector_hi PCL = swi_vect_low	INH	83		1	10	-	1	-	-	-
TAX	Transfer A to X	X = A	INH	97		1	2	-	-	-	-	-
TST short	TeST for	[short] - 0	DIR	3D	dd	2	4	-	-	#	#	0
TSTA	negative	A - 0	INH(A)	4D		1	3					
TSTX	or	X - 0	INH(X)	5D		1	3					
TST short,x	zero	[short+x] - 0	IX1	6D	LO	2	5					
TST ,x		[x] - 0	IX	7D		1	4					
TXA	Transfer X to A	A = X	INH	9F		1	2	-	-	-	-	-
WAIT	halt cpu clr I bit	I bit = 0 cpu clock stopped	INH	8F		1	2	-	0	-	-	-