DTC MICRO FILEAssembler

DTC MICRO FILE ONE-PASS ASSEMBLER

This document describes the implementation and operation of the one-pass assembler that executes on the DTC MICRO FILE System. It assembles a compatible sub-set of the Inteldefined 8080 microprocessor Assembly Language.

Refer to the INTEL 8080 Assembly Language Programming Manual for a complete definition of the instruction set.

TABLE OF CONTENTS

		PAGE NO.
I.	INTRODUCTION	1
II.	ASSEMBLER SPECIFICATIONS	2
	Labels Op-Codes Operands Comments One-Pass Characteristics Summary of Op-Codes/Functions Not Recognized	2 2 4 4 5
III.	ASSEMBLER USAGE	7
IV.	ERROR MESSAGES	9
V.	CONSOLE SESSION EXAMPLE	12

I. INTRODUCTION

This document describes the characteristics and usage conventions of a one-pass assembler that executes on the DTC MICRO FILE and assembles object code directly into memory, with an optional bias value.

An assembler program is a language processor. It simply translates statements of a 'source' program into their machine language counterparts.

Source programs are created as text files with the DTC Editor and then submitted to the assembly process with the command 'AS'. The console session example at the end of this document illustrates the use of the assembler

II. ASSEMBLER SPECIFICATIONS

The assembly language is a compatible sub-set of the language defined by Intel Corp. for its 8080 microprocessor. Any source program assembled by this assembler will also assemble on the Intel-provided resident or cross-assembler (note the ':' convention described below).

Labels

Labels are comprised of 1 to 5 alpha-numeric and special characters. A label must begin in column 1. The placement of a colon (:) after a label is optional. If a label is longer than 5 characters, only the first 5 are retained. Almost any special character may be used in a label but the user should restrict himself to only those special characters defined by Intel (@ and ?) if he wishes to maintain transportability.

Op-Codes

All of the Intel-defined instruction mnemonics are valid. An instruction mnemonic or pseudo-op may begin in any column except column 1. If a label is present, it must be separated from the label by one or more spaces.

The following pseudo-ops are valid:

(Label) ORG operand

Establishes the storage location to be occupied by subsequent instructions and data.

The label is optional and, if present, assumes the value of the operand field. The operand field must be a self-defining or previously-defined term. (label) EQU operand

Assigns a value to a label.

The label field must be present. The operand field must be a self-defining or previously-defined term.

(label) DS operand

Reserves a block of storage.

The label field is optional. The operand must be a self-defining or previously-defined term which is the number of bytes of storage to be reserved.

(label) DB operand, operand, . . ., operand

Defines 'bytes' of data.

The label field is optional. The operand(s) must be a self-defining or previously-defined term or it may be a character string bound by apostrophes. The value of a 'single-byte' operand must be between 0 and 255.

(label) DW operand

Defines 'words' of data

The label field is optional. The operand is any self-defining, previously-defined, or yet-to-be-defined symbol. Note that only one operand is permitted.

PON

Restores the output for subsequent statements to the listing option specified by the command statement (See section III). Label and operand fields are ignored.

POFF

Suppresses the listing output for subsequent statements until a PON statement is encountered. Label and operand fields are ignored.

END

This statement signals the end of the program. Neither a label field or an operand field is relevant.

Operands

Operands may take one of the following forms:

- 1. Reserved Symbols. Reserved symbols are A, B, C, D, E, H, L, M, SP, PSW.
- Decimal constants. Decimal constants begin and end with decimal numbers.
 e.g. 34, 192, 7429. The suffix 'D' is not permitted.
- 3. Hexadecimal constants. Hexadecimal constants begin with a numeric and end with the letter 'H' e.g. 47H, 0A42H, 0F34AH.
- 4. Character constants. Character constants are surrounded by single quote marks, e.g. 'K', 'T', '%'. A single apostrophe is represented within a string by two adjacent apostrophes.
- 5. Labels. Labels need not be defined before they are referenced if the referencing field is 16 bits. If the referencing field is 8 bits, they must be defined at the time of reference.

Operands may not contain expressions. That is, operands must be single terms and cannot contain an operator, e.g. DOG+3, 254/8 are not permitted.

Operands must be separated from the op-code field by one or more spaces.

Comments

Comments may appear in one of three places:

- 1. After an operand field.
- 2. After an op-code field, if the op-code does not require an operand.
- 3. As the only statement on a line. If this is the case, the comment must start in column 1.

Comments may be separated from the operand field (or op-code field if no operand is required) by one or more spaces. The semi-colon character denotes the beginning of the comment field.

One-Pass Characteristics

The assembler uses RAM as a buffer for 'holding' the object code as source statements are being processed. This means you must typically use a bias parameter which causes the object code to be stored into RAM in a different location than the one in which it will ultimately be executed. Also, the assembler can accommodate a program only as large as it can 'hold' in RAM. Very large programs require a large RAM capacity.

Referencing of yet-to-be-defined symbols works as follows:

When such a reference is made, the assembler creates a 'link-chain' of references for that symbol. When the symbol is defined, the 'chain' is 'un-linked' and each reference to the symbol is resolved and filled in with the proper value. The result for the user, is that when an undefined symbol is referenced, a 'link-chain' address is placed in memory and printed on the assembly listing device. The fact that this is a link rather than the value itself, is denoted by the appearance of an asterisk(*) immediately after the address field in the object code listing. A further benefit of this feature is that it gives the user a 'cross-reference' listing for his referenced-but-not-defined symbols.

Summary of Op-codes/Functions Not Recognized

A summary of excluded functions, op-codes, and features is given in this section. This section assumes that the user is familiar with the Intelspecified 8080 assembler language.

1. The following pseudo-ops are not recognized:

SET MACRO ENDM IF ENDIF TITLE

- Expressions (e.g. A+3, DOG/94, SHR 8, etc.) are not permitted.
- 3. Multiple operands in the DW pseudo-op are not permitted.
- 4. Binary and octal constants are not allowed.
- 5. The suffix 'D' after a decimal constant is not allowed.
- 6. The operand 'self' (i.e. \$) is not allowed.

III. ASSEMBLER USAGE

AS1 FNAME (DN (HEXBIAS))

This command activates the Assembler program. The assembler reads source statements from the file FNAME T and assembles the equivalent machine instructions directly into RAM. The RAM is simply used as a large buffer to hold the assembled code, with the symbol table being stored in upper RAM, below the stack. The assembler resides in RAM between the addresses 2700H and 30FFH, when loaded.

WARNING -

You must be very careful to provide a HEXBIAS parameter such that the resultant storage address is above 30FFH. Otherwise, the assembler will be aborted with an error message: ASSEMBLER ABORT... RAM CONTENTION. This action is also taken if the program object data starts to overlay the assembler's symbol table. This means you must know, in advance, where the program being assembled is ORG'ed so you can provide the proper HEXBIAS value. You can acquire this information by PRINTing the text file through the first ORG statement.

Options

The third character of the command statement above, has the following meanings:

- 1 = Assemble the program without a program
 listing, except for detected errors.
- 2 = Assemble the program with a "short"
 program listing, except for detected
 errors.
- other = Assemble the program with a full program
 listing.

Statements which are flagged with error codes are always printed in full, regardless of the listing option selected.

The symbol table occupies 1664_{10} bytes on the minimum 8K ROM/8K RAM system (3900H to 3F80H). This space accommodates 208 symbols and leaves 2K bytes for program storage (3100H to 38FFH).

The amount of storage reserved for the symbol table increases by 512 bytes for each 4K bytes of additional RAM.

IV. ERROR MESSAGES

All assembly-detected error messages are listed in this section. Error messages are denoted by the occurrence of a single character in column 1 of the listing.

One type of error is not detected. That is an address error whose value exceeds 16 bits. All addresses/constants are evaluated module 65536. Hence an address of 65539 is generated as a value of 00003 and no error condition is flagged.

E - Expression error

Something is wrong with the construct of an operand. Typically this may be caused by an incorrect designation of a character constant.

Example:

MVI A, 'CR'

F - Format error

This message indicates that something is wrong with the format of a statement. This could be caused by a missing or invalid operand.

Example:

MOV A,

MVI ,'T'

I - Illegal characters

This error is caused when an invalid character appears in an operand field. This is typically a hex or decimal constant which does not contain a valid digit.

Example:

LXI H,74MH

LXI D,2B

L - Missing label

This message occurs when an instruction or pseudo-op that requires a label does not have one.

Example:

EQU 42

M - Multiply-defined symbol

This message indicates that the label on the statement has also appeared as the label on one or more preceding statements.

Q - Questionable syntax

This message is usually caused by a missing or misspelled op-code or mnemonic.

Example:

A 34

MVA M, A

R - Register error

This message specifies that something is wrong with the register designation in an instruction.

Example:

MOV M,M (both operands cannot reference memory)

T - Table overflow

This message indicates that the amount of space allocated for the symbol table is inadequate. The table must be enlarged (additional RAM) or the number of symbols in the program reduced before the assembly can be accommodated.

U - Undefined symbol

This means that a symbol is being referenced for an 8-bit field whose value has not yet been defined. All 8-bit value fields must be defined before they can be referenced. This is not true of 16-bit value fields.

V - Value not in range

This error message indicates that the value of a symbol is incorrect for the op-code or that an 8-bit field has a value greater than 255.

Example:

RST 94

MVI A,929

V. CONSOLE SESSION EXAMPLE:

Creating a program:

The following console session illustrates the usage of the Editor, Assembler, and the SAVE command to create an executable program. The program is functionally very simple, it types a message at the console and then returns to the Monitor.

First, I must create a text file containing the source statements of my program. This is done with the Text Editor. I decide upon a name for my file and which disk I wish to store it on. (I will name it SAMP and store it on disk #1). Now I'm ready to invoke the Text Editor:

DTC MICRO FILE

```
*EDIT SAMP D1
NEW FILE:
-I
      ORG
            28ØØH
TOUT
                   ; SYSTEM MSG OUTPUT ROUTINE
      EOU 824H
      LXI H, MSG ; MESSAGE ORIGIN
      CALL TOUT
                   ; OUTPUT THE MESSAGE
      RET
                    : RETURN TO THE MONITOR
            24, 'THIS IS A SAMPLE MESSAGE'
MSG:
      DB
PEND:
      DS
                ; END OF PROGRAM
       END
-E
```

I now have a text file stored on disk #1. I can list it with the PRINT command:

```
*PR SAMP D1
       ORG 2800H
TOUT
       EOU
            824H
                    ; SYSTEM MSG OUTPUT ROUTINE
       LXI
            H,MSG
                    ; MESSAGE ORIGIN
       CALL TOUT
                    ; OUTPUT THE MESSAGE
       RET
                    ; RETURN TO THE MONITOR
MSG:
       DB
            24, THIS IS A SAMPLE MESSAGE'
PEND:
       DS
                    ; END OF PROGRAM
       END
```

I am now ready to assemble the program. I know that my program origin is 2800H and that the assembler's object program buffer begins at 3100H. Therefore, I invoke the assembler with a bias value of 0900H (3100H-2800H).

*ASM SAMP D1 0900 2800 ORG 28ØØH Ø824 TOUT EQU 824H ; SYSTEM MSG OUTPUT ROUTINE 2800 210000 * LXI H,MSG ; MESSAGE ORIGIN 2803 CD2408 CALL TOUT ; OUTPUT THE MESSAGE 2806 C9 RET ; RETURN TO THE MONITOR 2807 18544849 MSG: DB 24, 'THIS IS A SAMPLE MESSAGE' 2820 0000 PEND: DS ; END OF PROGRAM

00 ERRORS

At this point, the assembled object program resides in RAM beginning at 3100H. I will store it as an executable program on disk #0 and give it the same name 'SAMP':

END

*SAVE SAMP 3100 3120 2800

Now I can execute the program...

*RUN SAMP
THIS IS A SAMPLE MESSAGE

*

This concludes the console session.

Refer to the Programmer's Guide for more information about system routines.

III. Assembler Usage

AS1 HEXBIAS PN FN1 FT1 DN1 (FN2 FT2 DN2 . . .)

This command activates the Assembler program. The assembler reads source statements from the file(s) specified in the command line. Source files are designated by name, type and disk no. (FN FT DN). Several source files may be specified which will result in a logical 'cancatenation' at assembly time. This feature allows several small programs to be maintained seperately for ease of editing and/or as subroutines and brought together at assembly time as a complete program. The port number that is to receive the printed output is designated by the parameter 'PN' and can be any legitimate port designation. The assembler assembles the equivalent machine

instructions directly into RAM. The RAW is simply used as a large buffer to hold the assembled code, with the symbol table being stored in upper RAM, below the stack. The assembler resides in RAM between the addresses $\frac{2700}{34FF}$, when loaded.

40 DEE

WARNING - ESS 3500H

You must be very careful to provide a HEXBIAS parameter such that the resultant storace address is above COFFE. Otherwise, the assembler will be aborted with an error message: ASSEMBLER ABORT... RAM CONTENTION. This action is also taken if the program object data starts to overlay the assembler's symbol table. This means you must know, in advance, where the program being assembled is ORG'ed so you can provide the proper MEXBIAS value. You can acquire this information by PRINTing the text file through the first ORG statement.

Options

The third character of the command statement above, has the following meanings:

- 1 = Assemble the program without a program
 listing, except for detected errors.
- 2 = Assemble the program with a "short" (cbject or ly) program listing, except for detected errors.
- other = Assemble the program with a full program listing.

Statements which are flagged with error codes are always printed in full, regardless of the listing option selected.

The symbol table occupies 14,592 bytes between the addressed 8600H and EEFFH. This accomodates 1824 symbols. The remainder of RAM (3500H to B5FFH) is available for program storage.