6502 Macro Assembler Technical Documentation

Duncan Munro

January 2020

Contents

[1 High Level Design 3](#_Toc30769603)

[1.1 Software purpose 3](#_Toc30769604)

[1.2 General approach 3](#_Toc30769605)

[1.3 Files used 3](#_Toc30769606)

[1.4 Preprocessing pass 4](#_Toc30769607)

[2 Detail Design 5](#_Toc30769608)

[3 Technical Information 7](#_Toc30769609)

[4 File Formats 8](#_Toc30769610)

# High Level Design

This main section details the high level design of the assembler.

## Software purpose

The purpose of the software is to take a source file containing assembly language instructions for the 6502 processor and assemble this into an object file which can be executed in an appropriate processor environment.

## Key specifications

Some of the main functional features of the assembler are:

* Macro capability with nested macros
* Include file capability
* Conditional assembly using if / then / else
* Full expression evaluation
* Comprehensive error checking

## General approach

The assembly is conducted in three passes in order to deal with include files, macro expansions, variable definitions and code generation. The three passes are:

* 1 - Preprocessing
* 2 - Initial pass
* 3 - Final pass

At the end of the preprocessing stage, and interim file is created. The initial pass will resolve all variable address, preparing for the final pass which carries out the assembly and produces the output files.

## Files used

The following files are used and/or generated by the assembler:

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Contents | Ext | Attributes |
| Source | Source code to be assembled | .asm  .inc | Input  Human readable |
| Intermediate | Contains the preprocessed information from pass 1 | .asmpp | Temporary  Human readable |
| Debug | Records that can be used by an external debugging tool | .dbg | Permanent  Optional  Not readable |
| Object | Code records produced by the assembler in a binary form | .obj | Permanent  Not readable |
| Hex | Code records produced by the assembler in a text form | .hex | Permanent  Human readable |
| Listing | Listing in human readable form which can be reviewed or printed | .lst | Permanent  Optional  Human readable |
| Log | Details of progress, warnings and errors stemming from the assembly | .log | Permanent  Optional  Human readable |

## Preprocessing pass

The preprocessing pass will produce the intermediate file which can then be used for the subsequent passes. The purpose of this pass is to visit all include files referenced in the software, include them into the main file, define and expand and macros.

For this to be effective, the software must also process any conditional instructions at this point, these relying on expression evaluation.

A separate parser is used for this phase which contains many of the features of the main parser without the support for code generation.

The format of the intermediate file is described in section 4.1 of this document.

# Detail Design

# Technical Information

# File Formats

## Intermediate file

The intermediate file stores the results of the preprocessing, The lines contained within it could potentially be gathered from multiple source files so a list of files is summarised in the heading and referred to in short form later on.

|  |
| --- |
| **Contents** |
| File list |
| Source lines |

## Intermediate file list

The file list is a simple list of text lines followed by a line containing the string <end>.

Each text line has an ascending integer number starting at zero followed by a comma followed by the fully expanded filename, for example:

|  |
| --- |
| 0,C:\mydata\projects\spaceship\source\main\_unit.asm  1,C:\mydata\projects\spaceship\source\include\_file.inc  2,C:\mydata\projects\spaceship\source\another\_include\_file.inc  <end> |

## Intermediate source lines

The intermediate sources lines section is in the format:

*<filenum>,<linenum>,<sourceline>*

<filenum> The integer which corresponds to the file number in the intermediate file list described in section 4.2.

<linenum> The line number from the original source file which starts at 1 and increments sequentially as the file is read.

<sourceline> The optional source code line from the original file. This may be blank so the line may end after the second comma.

For example:

|  |
| --- |
| 0,1,;  0,2,; Sample file  0,3,;  0,4,.INCLUDE “extras.inc”  1,1,;  1,2,; Extras.inc file  1,3,;  : : : |

## Debug File Format

The debug file contains source lines and object code which can be used at a later stage by a debugging tool.

The general format is:

Line detail 1

Line detail 2

Line detail 3

: :

This is a binary format.

## Debug File Line Detail

Each debug file line contains the following information in a binary format

16 bit code address, or $FFFF if no code at this address

16 bit code output length indicator

Sequence of bytes output by this line

16 bit string length indicator

String as a sequence of bytes

This continues until the end of the file is reached

Index

Code generation, 3, 4

Conditional assembly, 3

Else, 3

Expressions

evaluation, 4

File format, 4

Hex, 4

If, 3

Include file, 3

Include files, 3, 4

Interim file, 3

Intermediate, 4, 8

Listing, 4

Log, 4

Macro, 1, 3

expansion, 4

Object, 4

Parser, 4

Preprocessing, 2, 3, 4

Source, 4, 8

Specifications, 3

Then, 3