ASM65

Usage Instructions

For version 0.1

Duncan Munro

19-Jul-2022

Contents

[1. ASM65 Syntax 3](#_Toc109159433)

[2. Directives 3](#_Toc109159434)

[3. Instructions 4](#_Toc109159435)

[3.1. Labels 4](#_Toc109159436)

[3.2. Opcodes 5](#_Toc109159437)

[3.3. Operands 6](#_Toc109159438)

[3.4. Comments 7](#_Toc109159439)

[4. Expressions 7](#_Toc109159440)

[4.1. Literal values 7](#_Toc109159441)

[4.2. Symbols 7](#_Toc109159442)

[4.3. Operators and Expression Precedence 8](#_Toc109159443)

[4.4. Integer Functions 9](#_Toc109159444)

[4.5. String Functions 9](#_Toc109159445)

[5. Command Line Usage 11](#_Toc109159446)

[6. Index 12](#_Toc109159447)

# ASM65 Syntax

Main elements are:

* Directives
* Instructions

Instructions are further divided into

* Labels
* Opcodes
* Operands
* Comments

# Directives

All directives are prefixed by the ‘.’ Character. A full list follows:

|  |  |
| --- | --- |
| .BYTE expression, expression | Defines a byte value followed by a non-zero repeat count. For example, .BYTE 32,20 for 20 spaces |
| .DB expression\_list | Defines a series of bytes in memory. The expression list is a comma separated list of expressions |
| .DD expression\_list | Defines a series of doublewords (32 bit) in memory. These are stored in a little-endian form |
| .DEFINE symbol  .DEFINE symbol = constant\_expression | Defines a symbol to exist and to have an optional value. The expression must be a constant expression that exists are the time of definition. It cannot access labels which are yet to be defined, this is import for conditional assembly with the .IF and related directives |
| .DEFMACRO name | Defines a macro, concludes with a .ENDM directive |
| .DS string\_expression | Defines a string of ASCII characters in memory |
| .DSZ string\_expression | Defines a string of ASCII characters in memory and terminates them with a following zero byte |
| .DW expression\_list | Defines a series of words (16 bit) in memory. These are stored in a little-endian form |
| .ELSE | Marks the end of a .IF block and the start of a .ELSE block |
| .ENDIF | Marks the end of an .IF or .IF / .ELSE block |
| .ENDM | Ends a .DEFMACRO block |
| .ERROR string\_expression | Raises the error message listed in the string expression and halts the assembly |
| .IF constant\_expression | Evaluates the constant expression and if it’s zero the following lines are not assembled. Used in conjunction with the .ELSE and .ENDIF directives. .IF statements can be nested. The expression must be known on the first pass or the assembly will fail |
| .IFDEF symbol | Similar to .IF but activates the following code if the symbol exists |
| .IFNDEF symbol | Similar to .IF but activates the following code if the symbol does not exist |
| .INCLUDE filename | Includes the filename into the source file. The .INCLUDE directive can be nested |
| .LIST | Turns the list file on (the default) |
| .MACRO name [param[,param[,…]]] | Creates an instance of a macro expansion |
| .MESSAGE string\_expression | Includes the message in the string expression into the assembly |
| .NOLIST | Turns the assembly listing off |
| .ORG expression | Sets the assembly origin to the expression. A default value of $0200 is used if not specified |
| .SET symbol expression | Sets the value of the named symbol to the expression. If the symbol does not already exist, it is created. If the symbol does already exist, it is amended |
| .UNDEFINE symbol | Removes a symbol from the symbol table |
| .WARNING string\_expression | Issues the warning represented by the string expression |

# Instructions

Instructions take the form

[label] [opcode [operand]] [comment]

## Labels

The label takes the form of an alphabetic character or underscore followed by zero or more trailing characters. The trailing characters may be an alphabetic character, digit or underscore. Finally, this is suffixed by a colon ‘:’ to indicate a label. Examples are:

Start:

\_loop\_pos\_3:

KX0001:

Labels used within a macro can be local by prefixing with the @ character, for example:

|  |
| --- |
| LDX #$00  LDY #14  TXA  @loop: STA addr,X  INX  DEY  BNE @loop |

At expansion time the @ is replaced by a local prefix purely for that expansion. For nested macros, each expansion will have its own local environment, for example:

|  |
| --- |
| .DEFMACRO INNER addval  ADC #@0  BCS @SKIP  ROR A  @SKIP:  .ENDM  .DEFMACRO OUTER checknum  @LOOP: .MACRO INNER $2A // Generates @L0000@SKIP  .MACRO INNER $3C // Generates @L0001@SKIP  BCS @SKIP  SBC #@0  @SKIP: // Generates @L0002@SKIP  .ENDM |

As can be seen, all the SKIP labels are prefixed differently so they become local to the macro which defines them.

## Opcodes

Opcodes can be one of the following:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ADC  AND  ASL  BCC  BCS  BEQ  BIT  BMI | BNE  BPL  BRK  BVC  BVS  CLC  CLD  CLI | CLV  CMP  CPX  CPY  DEC  DEX  DEY  EOR | INC  INX  INY  JMP  JSR  LDA  LDX  LDY | LSR  NOP  ORA  PHA  PHP  PLA  PLP  ROL | ROR  RTI  RTS  SBC  SEC  SED  SEI  STA | STX  STY  TAX  TAY  TSX  TXA  TXS  TYA |

## Operands

There are 13 different type of operands described for 6502 instructions. These are:

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Name** | **Example** | **Notes** |
| 1 | Implied | NOP  CLC | No operand actually exists, hence the term “implied” |
| 2 | Accumulator | ROL A  ASL A | The operand is the accumulator |
| 3 | Relative | BPL got\_one  BEQ -5 | Addr 🡨 PC + signed(expression8). A relative operator where the 8 bit signed expression yields a value in the range -128..+127 from the instruction following the current instruction |
| 4 | Literal | LDA #$20  AND #BITMASK | Value 🡨 expression8 |
| 5 | Absolute Zero Page | STA $82 | Addr 🡨 expression8 |
| 6 | Zero page indexed X | ORA $F0,X | Addr 🡨 expression8+X |
| 7 | Zero page indexed Y | DEC $40,Y | Addr 🡨 expression8+Y |
| 8 | Zero page indirect X | LDA [$A0,X] | Addr 🡨 (expression8+X). The effective memory address is the 16 bit address in the zero page memory pointed to by $A0+X. example, if X is 4 the 16 bit address from $A4 (low) and $A5 (high) is used |
| 9 | Zero page indirect Y | STA [$A0],Y | Addr 🡨 (expression8)+Y. The effective memory address is the 16 bit address in the zero page memory pointed to by $A0 to which Y is added after looking up the address |
| 10 | Absolute | EOR $2106  JSR get\_key | Addr 🡨 expression16. Uses the 16 bit address represented by the expression |
| 11 | Absolute indexed X | LDA $2000,X | Addr 🡨 expression16+X. Uses the 16 bit address represented by the expression + X |
| 12 | Absolute indexed Y | XOR $033A,Y | Addr 🡨 expression16+Y.Uses the 16 bit address represented by the expression + Y |
| 13 | Indirect | JMP [$FFFE] | Addr 🡨 (expression16). Uses the address stored at the expression |

In the above, expression8 indicates an 8 bit expression and expression16 indicates a 16 bit expression. Values in [brackets] indicate the 16 bit address stored at the address in memory pointed to by the expression.

## Comments

These will appear as a whole line, or after an instruction or directive. Comments are prefixed by ‘;’ or ‘//’. For example:

LDA #$20 ; Load ASCII space character

Loop: ; Come back round for next character

CALL key\_get ; Wait for a key

BMI Loop ; Sorry, it wasn’t valid

// We now have a valid key !

# Expressions

Expressions can be integer expressions or string expressions

Expressions are formed from literal values, symbols, operators and functions. Examples are:

A > B

1 << bit\_5

2 + 3 \* 4

LOW(address)

15 \* (1 + 2)

LEFT(title,3)

POS(“-“,title)

IIF(condition,a,b)

## Literal values

Literal values can be:

1. Decimal numbers – for example 123 or 0
2. Hexadecimal numbers, prefixed by $ for example $2F
3. Binary numbers, prefixed by % for example %01101001
4. String values enclosed in double quotes, for example “MyString”

## Symbols

Symbols are constant values or variables used within the assembly. They can be associated with:

* A null value
* An integer value
* A string value

A null value is produced by the .DEFINE directive where a symbol is declared but has not specific value associated with it. It can only be used with .IFDEF or .IFNDEF directives.

The following special symbols have been defined:

ORG – the current assembly origin

## Operators and Expression Precedence

Expressions are evaluated using the following precedence:

|  |  |
| --- | --- |
| **Precedence** | **Element** |
| 1 | ( expression ) |
| 2 | String to integer functions |
| 3 | Symbols  Special symbols  Numeric functions  + unary plus  - unary minus  ~ Not operator |
| 4 | \* multiplication  / division  & bitwise and  ^ bitwise xor  << shift left  >> shift right |
| 5 | + addition  - subtraction  | bitwise or |
| 6 | == comparison operators  !=  <  >  <=  >= |
| 7 | && boolean and  ^^ boolean xor |
| 8 | || boolean or |
| 9 | ! boolean not |
| 10 | = assignment operator |

## Integer Functions

These are functions returning an integer value. They may be dealing with strings.

|  |  |
| --- | --- |
| Function | Detail |
| ASC(string) | ASCII value of the first character in a string. Produces an error if the string is empty |
| HIGH(expression) | Returns the high byte of an expression |
| IIF(condition,trueexpr,falseexpr) | Immediate If, the expression returned depends if the condition is true or false |
| LOW(expression) | Returns the low byte of an expression |
| POS(substr,string) | Returns the position of a substring within another string in the range 1..n or zero if the substring is not found |
| VALUE(string) | Returns a value from a string for example “123” or “%011011”. Binary and hex strings are allowed |

## String Functions

These are functions returning a string value.

|  |  |
| --- | --- |
| Function | Detail |
| BUILD () | Current build string, for example 138 |
| CHR (value) | Returns the character corresponding to an ASCII value, for example CHR(65) will return “A” |
| DATE () | Returns todays date in the form YYYY-MM-DD, for example 2020-04-26 |
| HEX(value[,digits]) | Returns a value as a hex string without the preceding $ character. The correct number of digits will be used, for example 826 will convert to “33A” but it is possible to force the number of digits. As an example of that, HEX(826,4) will yield “033A”. If value cannot be represented in the number of digits, an error is produced |
| LEFT (string,count) | Returns the leftmost count characters from a string. If count is greater than the length of the string, then the whole string will be returned. If count is < 1 then an error is produced |
| LOWER (string) | Returns a string in lowercase form |
| MID (string,start,count) | Returns the mid part of a string from start character for count characters. If start < 1 or count < 1 then an error is produced. If start+count is greater than the length of the string, the rightmost part from start characters is returned |
| RIGHT (string,count) | Returns the rightmost count characters from a string. If count is greater than the length of the string, then the whole string will be returned. If count is < 1 then an error is produced |
| STRING(value) | Returns the value as a string representation |
| TIME () | Returns the current time in the form HH:MM:SS, for example 08:05:33 |
| UPPER (string) | Returns a string in uppercase form |
| VERSION () | Current version string, for example 1.0.3.22 |

# Command Line Usage

From the program startup when invoking asm65 with no parameters:

6502 Macro Assembler V0.1

Copyright (C)2020-2022 Duncan Munro

Usage: asm65 filename <options>

Options:

-b <bn> --debug=<bn> Set the debug filename to <bn>

-d <id> --define=<id> Define one or more symbols

-e <en> --errorlog=<en> Set error log to <en>

-h --help Display this message

-I <id> --include=<id> Set the include directory to <id>

-l <ln> --listing=<ln> Set the listing name to <ln>

-m <mn> --map=<mn> Set the map filename to <mn>

-o <on> --object=<on> Set the object name to <on>

-t <n> --tab=<n> Tab size for input file (default 4)

-v <n> --verbose=<n> Verbose output while assembling

-V --version Display version and other status info

-x <hn> --hex=<hn> Set the hex filename to <hn>

<bn> / <en>/<ln>/<mn>/<on>/<hn> default to the filename with ext changed to

.log/.hex/.lst/.obj/.hex respectively. Not specifying <en>, <ln>, <mn>

or <hn> will stop that output.

verbose <n> options:

0 Normal output levels (the default)

1 Verbose output

2 "War and Peace", lots more output

3 Debug level output

The include file directory and define list <id> can contain names or

symbols delimited by ; for example:

--define=DEBUG;ALLOW\_SPACES

--include=source/tables;source/help;/users/me/includes

An example of the above would be:

asm65 myfile --listing=myfile --map=myfile --object=newprog --verbose=1

# Index

.BYTE, 3

.DEFINE, 8

.DEFMACRO, 4, 5

.ELSE, 4

.ENDIF, 4

.ENDM, 3, 5

.IF, 3, 4

.IFDEF, 8

.IFNDEF, 8

.INCLUDE, 4

.MACRO, 5

Absolute, 6

Accumulator, 6

Boolean, 8

Comment, 4, 7

Comments, 3

Comparison operators, 8

Directive

.BYTE, 3

.DB, 3

.DD, 3

.DEFINE, 3

.DEFMACRO, 3

.DS, 3

.DSZ, 3

.DW, 3

.ELSE, 4

.ENDIF, 4

.ENDM, 4

.ERROR, 4

.IF, 4

.IFDEF, 4

.IFNDEF, 4

.INCLUDE, 4

.LIST, 4

.MACRO, 4

.MESSAGE, 4

.NOLIST, 4

.ORG, 4

.SET, 4

.UNDEFINE, 4

.WARNING, 4

Directives, 3

Expression, 3, 4, 6, 7, 8, 9

Expression16, 6, 7

Expression8, 6, 7

Functions, 7, 8, 9

ASC, 9

BUILD, 9

CHR, 9

DATE, 9

HEX, 9

HIGH, 9

IIF, 7, 9

LEFT, 7, 9

LOW, 7, 9

LOWER, 9

MID, 9

POS, 7, 9

RIGHT, 10

STRING, 10

TIME, 10

UPPER, 10

VALUE, 9

VERSION, 10

Implied, 6

Indexed, 6

Indirect, 6

Instruction, 6

Instructions, 3

Label, 4, 5

Labels, 3

Literal, 6, 7

Local prefix, 5

Macro, 3, 4, 5

Null, 8

Opcode, 4, 5

Opcodes, 3

Operand, 4, 6

Operands, 3, 6

ORG, 8

Precedence, 8

Relative, 6

String, 7, 8, 9

Symbol, 7, 8

Unary minus, 8

Unary plus, 8

Zero page, 6