HandyAsm

Documentation for HandyAsm version 1.10
Date: 10-May-90

This document is proprietary and confidential.

HandyAsm is a 65c02 cross assembler, designed to run on an Amiga computer, and used primarily for the development of software for the Lynx hand-held game system. It is assumed that the reader of this document is familiar with the 65c02 instruction set, addressing modes and mnemonics. HandyAsm is a one-pass assembler (for speed) with expression resolution at the end.

Running the assembler & command line options

From the CLI prompt, type:

asm <file1> [<file2> ... [<filen>]] [<opt1> [<opt2> ...]]

where file1, file2, etc. specify source files to assemble. If no extension is provided in a file name, an extension of .src is assumed. If more than one file name is specified, the files are treated as one continuous source. The assembled output is written to a file in Epyx's BIN format. The default name for the assembler output is taken from the root of the first source file name with .bin appended.

Command line options are specified with either + or - as the first character, followed by the option letter, optionally followed by an argument string. A summary of the command line options follows:

+B<size> - Output buffer size (default 73728)
The output buffer must be large enough to hold the BIN output file and all unresolved target files. If assembly fails due to output buffer overflow, use this option to increase it.

+C - Generate a complete listing (used with +L or +X option)
This option requires that a BIN file exist from a previous successful assembly. The BIN file is used to resolve forward address references.

+D<symbol>[=<value>] - Define symbol
Define the specified symbol and set it equal to value or to -1 if value not specified.

+E - Save equated symbols (used with +S option)
Save all symbols to the symbol file. Normally symbols defined with .EQ , .SE or .= directives are not saved to the symbol file

+H<size> - Expression heap size (default 8000 entries)
If assembly fails due to expression heap overflow, use this option to increase it.

+I<path> - Include path
The path string specifies where to look for all files included with the .IN directive.

+L[<filename>] - Listing file
Create a listing using the filename if specified, or the root of the first source file name with .lst appended. Forward references in the listing file will not be resolved unless the +C option is used.

+O<filename> - Output file
Specify the name for the BIN format output file.

+R[<filename>] - Error file
Create an error file using either the specified name or the root of the first source file with .err appended. All warning and error messages will be written to the error file in addition to the screen and any listing file. If no errors occur during assembly, the error file will not be created.

+S[<filename>] - Symbol file
Create a symbol file using either the specified name or the root of the first source file with .sym appended. Symbols defined with .EQ , .SE or .= directives will not appear in the symbol file unless the +E option is used.

+W - Watch internal debugging
Enable internal debugging messages in the assembler.

+X[<filename>] - Listing file with cross reference
Create a listing file with a cross reference listing using the specified file name or the root of the first source file name with .lst appended. Forward references within the source listing will not be resolved unless the +C option is used.

Syntax

The syntax for HandyAsm is similar to that for most assemblers. The source files are ASCII text files, with one statement per line. Blank lines are ignored. An asterisk (*) in the first column marks the line as a comment. A semicolon (;) anywhere in the line (except in the middle of a string or character constant) marks the rest of the line as a comment. The label definition field starts in the first column. The opcode field starts with the first non-whitespace character following whitespace (space or tab) characters. The operand field follows the opcode field, and is separated by at least one whitespace character.

Opcodes, pseudo-ops, directives and macro references all start in the opcode field. Operands, macro arguments, and directive arguments all

start in the operand field. Conditional assembly and macro definition directives can start in either the opcode or label definition fields.

Opcode names, directive names, operators and special constructions (numeric constants, escape sequences, etc.) are not case sensitive. Label names, macro names, strings and character constants are case sensitive.

Opcode equivalents

The following opcode equivalents are allowed:

```
INA == INC A == INC
DEA == DEC A == DEC
ASL A == ASL
ROL A == ROL
LSR A == LSR
ROR A == ROR
BCS <expr> == BGE <expr>
BCC <expr> == BLT <expr>
```

Labels

Label names can be up to 80 characters in length. Characters within a label name can be upper or lower case letters (labels are case sensitive), digits, periods (.), underscores (_) and question marks (?). Label names must have at least one non-digit character, and cannot start with a period.

Labels are defined by starting them in the leftmost column of the line, optionally followed by a colon (:), and must be followed by at least one whitespace character (space or tab) or the end of the line. Labels can appear on the same lines as opcodes, pseudo-ops and directives. Labels are assigned the current value of PC unless an assignment directive appears on the same line.

If a label is defined with the .SE directive, it gets a temporary value, and can be reassigned to a different value later; otherwise the assignment is permanent and cannot be changed.

Local labels are very similar to normal labels, but only have scope between two normal labels. As soon as a normal label is defined (except with the .EQ and .SE directives), local labels are resolved and cleared out for the next set of local labels. Local label names consist of a period (.) followed by one or two hex digits. The hex digits are treated as a number so .5 is the same as .05 and .fe is the same as .FE .

*Note: Macros and labels with the same name are not allowed since label names and macro names have entries in the same symbol table.

*Note: The label names A and a are legal, but certain opcodes will use the accumulator addressing mode if either appear alone in the operand field.

Numeric expression syntax

A constant can be either a numeric or character constant. Numeric constants are represented by a string of digits and are assumed to be decimal unless preceded by a base prefix character: dollar sign ($) for hex, at symbol (@) for octal, and percent sign (%) for binary. Character constants are a single character or escape sequence surrounded by either single quotes (') or double quotes ("). If a character constant uses single quotes, its value is the ASCII value of the character. If the constant uses double quotes, its value is determined by taking the ASCII value as an index to look up in the CharSet table (see the section at the end on strings and the .CSET directive).

Numeric expressions are evaluated left to right. Labels are case sensitive and are referred to by name. The asterisk (*) is used to refer to either the value of PC at the beginning of the current instruction or in the case of a directive, the current value of PC.

Expression operators have the following precedence: unary operators < (low byte), > (high byte), ~ (bitwise not) and - (unary minus) followed by * (multiply), / (divide), & (bitwise and), and ^ (bitwise xor) followed by + (add), - (subtract) and | (bitwise or). Precedence can be forced by grouping expressions with curly braces ('{' and '}').

A BNF description of expression syntax follows:

```
<expr> := <label> | <constant> | '*'
<expr> := <expr><op><expr>
<expr> := <uop><expr>
<expr> := '{'<expr>'}'

<op> := '+' | '-' | '*' | '/' | '&' | '|' | '^'

<uop> := '<' | '>' | '~' | '-'
```

Pseudo-ops and Directives

Pseudo-ops and directives all start with a period (.) and appear in the opcode field of the source line. A summary of the pseudo-ops and directives supported by HandyAsm follows:

.65c02
: No effect. Included for compatibility with a previous assembler.

.AL [<cexpr>]
.ALIGN [<cexpr>]
: Align PC. Align the PC to an even multiple of cexpr or to a multiple of 256 if cexpr not specified.

.AS <string>
.ASCII <string>

.65c02
No effect. Included for compatibility with a previous assembler.

.AL [<cexpr>]
.ALIGN [<cexpr>]
Align PC. Align the PC to an even multiple of cexpr or to a multiple of 256 if cexpr not specified.

.AS <string>
.ASCII <string>
ASCII string. Take the characters of the string specified and copy them as bytes to the output file. The characters will translated by any active .CSET directive. The string can be delimited by any character except space, tab or newline.

.AT <string>
ASCII terminated string. Same as the .AS directive except the high bit is set on the last character of the string.

.BS <cexpr>
.DS <cexpr>
Block skip or define storage. Add cexpr to PC and write a new origin header to the BIN file.

.BY <expr>,<expr>,<expr>, ...
.BYTE <expr>,<expr>,<expr>, ...
Byte data. Evaluate each expr and write as a data byte to the BIN file.

.CH <filename>
.CHAIN <filename>
Chain to file. Stop reading the current source file and start reading the specified chain file as the current source file. Does not return control to the current file (see the .INCLUDE directive).

.CL [ON|OFF]
.CLIST [ON|OFF]
Conditional list of false branches. Enable/disable listing of false lines in conditional assembly. If neither ON nor OFF specified, then toggle current state. Conditional list defaults to OFF at the beginning of assembly.

.CS [<filename>]
.CSET [<filename>]
Character set substition/translation. Build a character set translation table (the CharSet table) as specified by the file. If no file is specified, cancel current translation. See the section at the end on strings and the .CSET directive.

.DA <expr>,<expr>,<expr>, ...

.DATA <expr>,<expr>,<expr>, ...
.WO <expr>,<expr>,<expr>, ...
.WORD <expr>,<expr>,<expr>, ...
Word data. Evaluate each expr and write as a data word (two bytes, low byte first) to the BIN file.

.DB <expr>,<expr>,<expr>, ...
.DBYTE <expr>,<expr>,<expr>, ...
Double byte data. Evaluate each expr and write as double byte data (two bytes, high byte first) to the BIN file.

.EC <string>
.ECHO <string>
.ME <string>
.MESSAGE <string>
Echo message to screen. Copies the string (with source file name and source line number) to standard output (the screen). The string can be delimited by any character except space, tab or newline. The characters in the string are not translated by the .CSET directive.

.EJ [<cexpr>[,<cexpr>]]
.EJECT [<cexpr>[,<cexpr>]]
.PA [<cexpr>[,<cexpr>]]
.PAGE [<cexpr>[,<cexpr>]]
Page eject or set page size. If no arguments specified, output a form-feed and a new page header to the listing file. If one argument specified, set the page length. If two arguments specified, set the page length and width.

.EN
.END
End of source file. Stop assembling current source file. If the current file was included by another file, return to the previous file.

.EQ <expr>
.EQU <expr>
.EQUATE <expr>
Define equate. Set the label on the source line to the value specified by expr. The source line must have a label. Labels defined with .EQ do not disturb the current local labels.

.HE <hex string>
.HEX <hex string>
.HS <hex string>
Hex string. Take the hex string (no delimiters) two characters at a time and output as bytes to the BIN file.

.IN <filename>
.INCLUDE <filename>
Include source file. Temporarily suspend reading of

the current source file, and read the specified file. Included files can be nested.

.LI [ON|OFF]
.LIST [ON|OFF]
Enable/disable output to listing file. If neither ON nor OFF specified, toggle the current state. Listing defaults to ON.

.ML [ON|OFF]
.MLIST [ON|OFF]
Macro listing. Enable/disable listing of macro expansions. If neither ON nor OFF specified, toggle the current state. Macro listing defaults to ON.

.NS <string>
Negative ASCII. Same as the .AS directive except the high bit is set on every character in the string.

.OR <cexpr>
.ORG <cexpr>
.ORIGIN <cexpr>
Set origin. Assign PC to the value of cexpr.

.PC <filename>
Load core file. Load in and merge the specified file as raw data with the current output.

.PD <filename>
Load data file. Load in and merge the specified BIN format file with the current output.

.RS <string>
Reverse string. Same as the .AS directive except the characters of the string are output in reverse order.

.RU [<expr>]
.RUN [<expr>]
Run address. Set the RUN address of the BIN file to expr if specified or to the current value of PC.

.SE <expr>
.SET <expr>
.= <expr>
Set temporary value. Same as the .EQ directive except the label is marked as temporary and can be redefined to another value later.

.ST <string>
.STITLE <string>
.SU <string>
.SUBTITLE <string>
Subtitle. The specified string becomes the new subtitle printed at the top of every listing page.

.SY <filename>
.SYMBOL <filename>
Include symbol file. Loads in the specified symbol file and adds the label definitions to the current symbol table.

.TA <cexpr>
.TARGET <cexpr>
Target address. Writes a new origin header to the target file without disturbing PC. WARNING: the .DS , .BS , .AL or .ALIGN directives may cause new origin headers to written out with the current PC.

.TF <filename>
Target file. Stop writing to the current BIN file and direct output to the specified file.

.TI <string>
.TITLE <string>
Title. The specified string becomes the new title printed at the top of every listing page.

.VE <cexpr>[,<cexpr>]
.VERSION <cexpr>[,<cexpr>]
Version number. Set the version number and optionally the revision number to be printed at the top of every listing page.

.ZS <string>
Zero terminated string. Same as the .AS directive except a byte of zero is output after the string.

A cexpr (constant expression) is a numeric expression that must be resolved immediately (it cannot have forward references).

Filenames can include full path names, but cannot contain any imbedded whitespace characters.

Conditional assembly and macros

Macro and conditional assembly directives can either start in the first column, or can start in the in the opcode field. Conditional assembly directives and macro definitions and macro references can be nested.

#IF <ifexpr>
Evaluate the comparison expression, and if true continue assembling code, otherwise skip to the matching #ELSE or #ENDIF statement.

#IFDEF <label> [<lop> <label> [...]]
Check the specified label(s) and if defined continue assembling code, otherwise skip to the

matching #ELSE or #ENDIF statement.

#IFNDEF <label> [<lop> <label> [...]]
Check the specified label(s) and if not defined continue assembling code, otherwise skip to the matching #ELSE or #ENDIF statement.

#ELSE
If skipped to from a false #IF , #IFDEF or #IFNDEF continue assembling code, otherwise skip to the matching #ENDIF statement.

#ENDIF
Terminate a conditional #IF , #IFDEF or #IFNDEF construct.

#REP <cexpr>
#REPEAT <cexpr>
Repeatedly assemble the code following until the matching #ENDM statement cexpr times.

#MAC <label>
#MACRO <label>
Start a macro definition bound to label and continue until the matching #ENDM is found.

#ENDM
End a macro definition or a #REP construct.

The #IF statement requires a comparison expression. Two or more comparisons can be evaluated by linking them with .AND. (& or &&), .OR. (| or ||) or .XOR. (^ or ^^) logical operators. The truth of an individual comparison can be inverted with the .NOT. (!) logical operator. The allowed comparison operators are: > (greater than), < (less than), = or == (equal to), != or <> (not equal to), <= (less than or equal to) and >= (greater than or equal to). Comparisons can be grouped with parentheses, but grouping within expressions still requires curly braces.

A BNF description of #IF comparison expression syntax follows:

```
<ifexpr> := <cexpr><cmpop><cexpr>
<ifexpr> := <ulop><ifexpr>
<ifexpr> := '('<ifexpr>')'
<ifexpr> := '('<ifexpr>')'<lop>'('<ifexpr>')'
<ifexpr> := <ifexpr> <lop> <ifexpr>

<cmpop> := '>' | '<' | '=' | '==' | '!=' | '<>' | '<=' | '>='

<ulop> := '!' | '.NOT.'

<lop> := '&' | '&&' | '.AND.' | '|' | '||' | '.OR.'
<lop> := '^' | '^^' | '.XOR.'
```

#IFDEF and #IFNDEF check for the existence or non-existence of a label definition in the symbol table. Logical operators (separated from the

abel names by spaces) can be used to check more than one label in the same atement. The logical operators allowed for #IFDEF and #IFNDEF are .AND. nd .OR. (& or && and | or ||).

The syntax for a macro invocation is

<macro name> [<arg>[,<arg>[...]]]

with the macro name in the opcode field, and a maximum of 16 arguments to pass to the macro. Commas and spaces can be passed in a macro argument by surrounding the argument with double quotes ("). A double quote can be passed in the macro argument by using two double quotes (""").

Argument expansion constructs start with a question mark (?). Argument expansions can appear anywhere within the line, and are performed with a literal string replacement.

Within the macro, ?<n> (where <n> is a single hex digit) is replaced by the (n+1)'th argument (?0 expands to the first argument, ?1 the second, ?f the 16th).

The construct ?%d<n> expands to a string of digits which is the decimal value of argument n+1, and ?%x<n> expands to a string of digits which is the hex value of argument n+1. Arguments which are expanded by ?% constructs must consist of single label names (expressions are not allowed).

The construct ?? is replaced by a string which is unique to the irrent invocation of the macro (the macro name, followed by a period, ollowed by a string of hex digits representing the number of different macro invocations before the current one).

The construct ?# is replaced by a two digit decimal string which is the count of the arguments passed to the macro.

To avoid expansion of question mark constructs, the question mark can be preceded with a backslash (\).

*Note: Macros and labels with the same name are not allowed since label names and macro names have entries in the same symbol table.

*Note: The assembler will not complain if a macro is given a name that is the same as an opcode or that starts with a period. However, macros with these names will not be able to be invoked.

Strings and the .CSET directive

Strings and character constants are translated one character at a time into byte values. Escape codes of two or more characters are provided to allow entry of characters which are difficult to get into a source file. Escape sequences all start with a backslash (\) character. The available scape sequences are:

```
\b       - backspace (CTRL-H or $08)
\e       - escape ($1b)
\f       - form feed (CTRL-L or $0c)
\n       - newline (CTRL-J or $0a)
\r       - return (CTRL-M or $0d)
\t       - tab (CTRL-I or $09)
\v       - vertical tab (CTRL-K or $0b)
\<nnn>   - where <nnn> is a string of 1 to 3 decimal digits,
           convert the decimal string to a 1 byte number
\x<nn>   - where <nn> is a string of 1 or 2 hex digits,
           convert the hex string to a 1 byte number
\$<nn>   - where <nn> is a string of 1 or 2 hex digits,
           convert the hex string to a 1 byte number
\\       - backslash
```

Each byte in strings (and character constants surrounded by double quotes) is translated through the CharSet table loaded by the last .CSET directive. If no .CSET directive has been encountered, or the CharSet table has been cleared, then the characters are passed through with no translation.

The file specified by the .CSET directive is a 256 line text file. One hexadecimal number (with no prefix characters) is supplied in each line in the file starting in the first column. Comments can follow the number. Each number specifies the translation value for a character (i.e. the first line gives the value that instances of character $00 are replaced by, the next line gives the value for character $01, etc.).