Z80Asm.py - Z80 Cross Assembler for Windows and Linux (v220807)

Introduction

Z80Asm.py is a new assembler for 8080, Z80 and Z180 code, written in Python 3, so that it will run on both Windows and Linux (also any other operating system with a reasonable Python 3 implementation). It can generate machine code as either binary images or Intel Hex format files for writing to ROM (EPROM, EEPROM, or Flash).

One of the features of this program is that it supports multiple styles of coding, outlined below. It is also able to reformat source code from one style to another.

MA Style

This is the style of coding used by Martin Allcorn's BBC Basic assembler. The identifying features of this style are:

- Labels (call and jump destinations and data addresses) appear on a line of their own, prefixed by a stop ".".
- Labels are case insensitive.
- Opcodes start in column 1.

This mode has been tested by assembling the source of the CFX-II ROM and confirming that the resulting binary image is consistent with Martin's original.

The special macros needed to assemble the "System Diagnostic Cartridge" have not been implemented. It would not be difficult to add these but I would prefer to add a general macro capability.

Martin has now produced a second version of his assembler, written in ARM assembly language. The syntax is largely the same as the one in BASIC. The most important difference is the expression evaluator. The ARM version does not use the normal operator precedence, instead expressions are evaluated strictly left to right. The resulting values may be not what would normally be expected from the form of the expression.

The RISCOS file type of the source input to the two assemblers differ, 080 for the new ARM version and 281 for the original BASIC version. When transferring the files from RISCOS to Linux or Windows the file type is appended to the file name, preceded by a comma. Z80Asm.py uses this trailing file type (if present) to select which form of expression evaluator to use. This has been tested by assembling the source of the MFX ROM and confirming that this is consistent with Martin’s original.

M80 Style

This is not (yet) a complete emulation of the CP/M M80 assembler, but more a convenient designation of the style of coding. Macros have not been implemented. The identifying features of this style are:

- Labels generally appear on the same line as the instruction to which they refer, followed by a colon ":".
- Labels are case sensitive but only the first 6 characters are significant.
- Opcodes are generally indented.

This mode has been tested by assembling the CP/M and SDX ROMs from the source by Andy Key. For more details see the section on M80 / L80 emulation below.

ZASM Style

ZASM is my Z80 assembler that runs on CP/M. This is similar to M80 style:

- Labels are case insensitive and restricted to a maximum of 8 characters.
- Some pseudo-ops differ.

PASMO Style

PASMO is a Z80 cross assembler typically used in ZX Spectrum circles (http://pasmo.speccy.org/). Its main difference from M80 style is that labels are not case sensitive. Reformatting to PASMO style is not (yet) supported, but would not be very different to reformatting to M80 style.

Usage

```
usage: Z80Asm [-h] [-v] [-b [BINARY]] [-f FILL] [-x [HEX]] [-y [SYMBOL]]
              [-n] [-l [LIST]] [--list-force] [--list-cond] [-a] [-o [OUTPUT]]
              [-r {MA,M80,ZASM}] [--multi-inc] [-m] [-k [KEEP]] [-e]
              [-t {8080,Z80,Z180}] -s {MA,M80,PASMO,ZASM} [-p] [-u] [-c CSEG]
              [-d DSEG] [--debug] [-D DEFINE] [source ...]

Assemble Z80 code written in different styles

positional arguments:
  source                The Z80 source file(s)

options:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  -b [BINARY], --binary [BINARY]
                        Machine code in binary format
  -f FILL, --fill FILL  Fill byte for undefined addresses
  -x [HEX], --hex [HEX]
                        Machine code in Intel hex format
  -y [SYMBOL], --symbol [SYMBOL]
                        Save all symbol definitions in source format
  -n, --number-build    Append build number to assembled file names
  -l [LIST], --list [LIST]
                        List file
  --list-force          Ignore NOLIST directives
  --list-cond           List false conditional code
  -a, --address         Show load address as well as relocation
  -o [OUTPUT], --output [OUTPUT]
                        Reformatted source file
  -r {MA,M80,ZASM}, --reformat {MA,M80,ZASM}
                        Style for reformatted source (default M80)
  --multi-inc           Include files multiple times in reformatted source
  -m, --modeline        Emacs modeline in reformatted source
  -k [KEEP], --keep [KEEP]
                        Keep pass 1 list file
  -e, --echo            Echo source to screen
  -t {8080,Z80,Z180}, --cpu-type {8080,Z80,Z180}
                        The processor type
  -s {MA,M80,PASMO,ZASM}, --style {MA,M80,PASMO,ZASM}
                        The style of the Z80 source
  -p, --permissive      Ignore some syntax errors
  -u, --update          Allow updating (patching) of previous code
  -c CSEG, --cseg CSEG  Start address for code segment
  -d DSEG, --dseg DSEG  Start address for data segment
  --debug               Show assembler debug info
  -D DEFINE, --define DEFINE
                        Define an assembler equate
```

Details

-b [BINARY], --binary [BINARY]

Save the machine code as a binary image file. Optionally followed by the name of the file. If no file name is specified, defaults to the root of the source file name with a ".bin" extension.

-f FILL, --fill FILL

If the source leaves gaps in the generated machine code, then these gaps have to be filled using padding bytes in the image. This option specifies the byte to use. If the switch is omitted, then a default of 0xFF is used.

-x [HEX], --hex [HEX]

Save the machine code as an Intel hex file. Optionally followed by the name of the file. If no file name is specified, defaults to the root of the source file name with a ".hex" extension. It is possible to save both binary and hex images.

-y [SYMBOL], --symbol [SYMBOL]

Save all the symbols (labels and equates) in the format of an assembler source file. Optionally followed by the name of the file. If no file name is specified, defaults to the root of the source file name with a ".sym" extension.

-n, --number-build

Append "__Bnnn", where nnn is the build number (see below) to the binary and/or hex image file names (before the extension).

-l [LIST], --list [LIST]

Save a list file showing the source code, generated machine code and any error messages. If no file name is specified, defaults to the root of the source file name with a ".lst" extension.

--list-force
: Ignore NOLIST commands, generate a listing anyway.

--list-cond
: List false conditional code by default. n be controlled by pseudo-ops.

-a, --address
: For relocated code (see below) show both the location in the binary image, and the addresses at which the code will be executed. If omitted only the execution addresses are shown.

-o [OUTPUT], --output [OUTPUT]
: Save a reformatted version of the input source code. If no file name is specified, defaults to the root of the source file name with an extension the lower case version of the reformatted style.

-r {MA,M80,ZASM}
: The coding style to use for reformatted source code. If not specified, defaults to "M80".

--multi-inc
: When generating reformated source files, if the original file includes the same file multiple times, include multiple copies in the generated output. This is likely to result in multiply defined labels.

-m, --modeline
: Adds an Emacs style modeline to the top of the list file and reformatted source file to specify the tab spacing (8 spaces) to use when displaying / editing the file. Many other editors understand Emacs modelines.

-k [KEEP], --keep [KEEP]
: Normally the list file generated during the first pass is overwritten by the second pass listing. For some errors it can be useful to compare the first and second pass listing. This option preserves the listing from the first pass. If no file name is specified, then it defaults to the root name of the source file, followed by "_p1.lst".

-e, --echo
: Echo source lines to the screen as they are processed in order to display progress. Otherwise only error messages and their corresponding source lines are displayed.

-t {8080,Z80,Z180}, --cpu-type {8080,Z80,Z180}
: Specifies the CPU to assemble for. Defaults to "Z80". Specifying "Z180" enables additional opcodes. For 8080, Intel style opcodes are used.

-s {MA,M80,ZASM,PASMO}, --style {MA,M80,ZASM,PASMO}
: Specifies the style of the source code. One of these options must be selected.

-p, --permissive

Allow a slightly looser coding style. Features enabled by this option include:

- Labels may be multiply defined, providing all definitions result in the same value. By default labels may only be defined in one location.
- Missing closing quotes are ignored in some cases.

-u, --update

Allow setting the load address back and updating previously output code. This is typically used to patch an included binary file.

-c CSEG, --cseg CSEG

Start address for code segment. Defaults to 0x0000.

-d DSEG, --dseg DSEG

Start address for data segment. Defaults to end of code segment.

--debug

Include information to assist debugging the expression evaluator in the list file. Probably only of interest to the author.

-D symbol[=value], --define symbol[=value]

Defines the value of a symbol. Equivalent to an equate statement in the source code. Typically used to select conditional code sections. The value may be specified in decimal (the default), binary (prefix 0b), octal (prefix 0o) or hex (prefix 0x). If no value is specified it defaults to one (true).

A number of the above options are followed by an optional file name. If the next parameter following one of these options does not begin with a dash "-", it will be taken as the file name. This could result in the source file name being mistaken for the option file name. For this reason it is recommended that the options are specified in the order given above, with the style option last.

M80 / L80 Emulation

With the CP/M M80 assembler, code is generated in two stages:

- Firstly source code files were assembled to relocatable object files using M80.
- These object files were then linked together to form the final executable using L80.

As a consequence of this two stage process, there are two types of labels in the source code:

- Local labels, which are only relevant to the current source file (and any included files), and which may be reused (for different purposes) in other source files.
- Global labels, which are passed to L80, and must be unique across all the source files forming the final executable.

Z80Asm emulates this two stage process in a single operation. In order to do so, all of the sources that would be assembled in separate M80 steps are specified on the command line. These are then assembled to the final executable in one step. In order for this to work, Z80Asm still retains the distinction between global and local labels. Only global labels are visible across all of the source files listed on the command line. Each command line source has its own set of local labels. This is different to files specified by "INCLUDE" pseudo-ops, which are considered to be part of the source of the file which includes them, and therefore have access to the same local labels.

L80 was designed to generate CP/M COM files, and therefore generates the following memory layout by default:

0x0100	JP <start address>
0x0103	Data segment
?	Code segment

Z80Asm uses a different memory layout by default:

0x0000	Code segment
?	Data segment

Therefore to generate a CP/M COM file it is necessary to:

- Use the option "--cseg 0x100" to specify the start location.
- Explicitly code the jump to start in the source. If working with existing M80 source, this could be in an additional short source file, which is listed first on the command line.

M80 / L80 features that are not emulated include:

- M80 macros are not currently supported.
- There is no ability to search libraries of relocatable code to satisfy external references.

Reformatting

The option of reformatting code is mainly intended to improve readability for people who are used to a particular style of formatting. Some issues to be aware of when using reformatted code:

- Labels:
    - Z80Asm (in all modes), treats all characters in a label as significant. By default the characters must match in case (so loop:, Loop:, and LOOP: are different labels).
    - It is believed that Martin's assembler treats all characters as significant, but ignores case.
    - M80 only considers the first 6 characters of a label. This is why a new assembler was required, rather than just reformatting Martin's code to M80.
    - ZASM only considers the first 8 characters of a label. All labels are converted to upper case.

- Relocations:
    - Only M80 (and Z80Asm) have the concept of code and data segments, which makes it difficult to translate some M80 code into formats suitable for the other assemblers.
    - The ORG pseudo-op and other relocation pseudo-ops are interpreted differently by the different assemblers, and by Z80Asm in the different styles. Z80Asm attempts to translate these appropriately when reformatting, but the translation is not always reversible.
- Pseudo-ops:
    - Z80Asm attempts to recognise most of the pseudo-ops from MA, M80 and ZASM assemblers. When reformatting, there is often not a suitable equivalent pseudo-op to translate to.
    - A number of the M80 pseudo-ops start with a stop ".", and will therefore be regarded as a label by Martin's assembler.
- Include files:
    - All include files are merged into the reformatted source.
    - The INCLUDE statement is converted into a comment.

Relocated Code

For code that is initially loaded in ROM, but is then either paged to a different address, or copied into RAM there are two relevant addresses:

Location counter:
: Location of the code in ROM / image file.

Program counter:
: Location where the code is executed and address for calls and jumps.

The three different styles of coding differ in the pseudo-ops used to specify these addresses.

MA Style

OFFSET [address]
: Sets the program counter to the specified address, and remembers the offset between the program counter and location counter. If no address is specified, resets the program counter to the location counter, and sets the offset to zero.

ORG address
: Sets the location counter to the specified address and the program counter to the location counter plus the previously remembered offset.

BORG address

Sets the program counter to the specified address and the location counter to the location counter minus the previously remembered offset. The converse of ORG.

M80 Style (and PASMO Style)

ORG address

Sets both the location and program counters to the specified address.

.PHASE address

Sets just the program counter to the specified absolute address.

.DEPHASE

Resets the program counter to the location counter.

ZASM Style

LOAD address

Sets the location counter to the specified address.

ORG [address]

Sets the program counter to the specified address. If no address is specified resets the program counter to the location counter. (Note: The original ZASM does not support omitting the ORG address).

Pseudo-ops

The following pseudo-ops are recognised by Z80Asm (in addition to the location ops given in the previous section). Most of these pseudo-ops are recognised for all styles, irrespective of which style they originated from.

IF expression
IFT expression

If the expression evaluates to non-zero, code to the corresponding ELSE or ENDIF is emitted. If zero, following code is not emitted. The following code must still be syntactically correct to enable reformatting.

IFF expression

If the expression evaluates to zero, code to the corresponding ELSE or ENDIF is emitted. If non-zero, following code is not emitted. The following code must still be syntactically correct to enable reformatting.

ELSE

Reverses the effect of the corresponding IFx pseudo-op. If code was being emitted previously then the following code up to the corresponding ENDIF will not be emitted. If code was not being emitted (and is not suppressed by an outer condition), then code up to the corresponding

ENDIF will be emitted. The following code must still be syntactically correct to enable reformatting.

ENDIF

Terminates a section of conditional code.

label[:] EQU expression

Assigns the value of the expression to a label. The colon after the label is optional (ZASM requires it). The absolute addresses of all labels are used in evaluating the expression, and the result (if an address) is absolute.

END

Marks the end of a source file.

LIST
.LIST

Turns on output to the list file (if enabled).

NOLIST
.XLIST

Turns off output to the list file.

.LFCOND

Turn on the listing of non-assembled conditional code.

.SFCOND

Turn off the listing of non-assembled conditional code.

.TFCOND

Toggles the listing of non-assembled conditional code. Reverses the effect of the --list-cond command option or the previous .TFCOND pseudo-op.

NAME name
NAMEX name

Specifies a name for the code. Ignored by Z80Asm.

TITLE title

Specifies a title for the code. Ignored by Z80Asm.

INCLUDE filename

Includes the assembler statements from the specified source file at this location. File name matching is case insensitive (even on Linux). For MA style any RISCOS file type (comma and number) at the end of the file name is ignored. For the other styles, if no extension is specified it is assumed to be the same as that for the original source file specified on the command line. If generating reformatted output, all the included files are in the single output file, multiple outputs are not generated. The include statement is converted to a comment in the reformatted output.

INSERT filename

Includes the binary contents of the specified file into the generated code at this location. File name matching is case insensitive (even on Linux). For MA style any RISCOS file type (comma and number) at the end of the file name is ignored.

.COMMENT delim

Text up to the next occurrence of the specified delimiter character is included in the list file as a comment, with no processing.

.PRINTF delim
.PRINTX delim

Text up to the next occurrence of the specified delimiter character is included in the list file and output to the terminal as a comment, with no processing.

DATE

Include the current date as a 10 byte string in the generated machine code.

BUILD

If not previously used, read the current build number from the file "build" in the current directory (as a 4 byte integer), increment it and save the new value. Include the build number as a 5 byte string in the generated machine code.

.8080

The following code is in Intel format for the 8080 chip.

.Z80

The following code is in Zilog format for the Z80 chip.

.Z180

The following code is in Zilog format for the Z180 chip.

ASEG

The locations referenced in the following section are absolute. This is the default if nothing else is specified.

CSEG

The locations referenced in the following section are relative to the start of the code segment.

DSEG

The locations referenced in the following section are relative to the start of the data segment.

EXT label [,label [...]]
EXTRN label [,label [...]]
ENTRY label [,label [...]]
PUBLIC label [,label [...]]

> Declares each of the labels listed to be global, having a common value across all the source files listed on the command line. A location label may also be made global by following it by two colons.

EQUD decimal_number

> Include the specified number as a 4 byte little-endian integer value in the generated machine code.

DB expression [,expression]...
DEFB expression [,expression]...

> Include the specified list of byte values in the generated machine code. If the expression is a string, then all the bytes of the string are output.

DW expression [,expression]...
DEFW expression [,expression]...

> Include the specified list of values in the generated machine code as 2 byte little-endian word values.

DC string [,string]...
DEFC string [,string]...

> Include each of the listed strings in the generated machine code, with the msb of the last byte of each string set.

DZ string [,string]...
DEFZ string [,string]...

> Include each of the listed strings in the generated machine code, with each string terminated by a zero byte.

(M80 style only) DS value

> Reserve the specified number of bytes of space in the machine code for variables. In a binary image the space is padded with the fill byte.

(MA and ZASM styles) DS string, [,string]...
DEFS string, [,string]...

> Include each of the listed strings in the generated machine code.

BYTE expression

> Reserve the specified number of bytes of space in the machine code for variables. In a binary image the space is padded with the fill byte.

WORD expression

Reserve the specified number of 2 byte words of space in the machine code for variables. In a binary image the space is padded with the fill byte.

ZERO expression

Insert the specified number of zero bytes into the machine code.

EVAL {FULL | SIMPLE}

Select either the normal (full) expression evaluator or the simple left to right evaluator. This is typically used in source files generated by the reformatter. It is not practical to re-organise expressions from one format to the other, so setting this option ensures that they are evaluated in the same way as the original source.

LABCASE {YES | NO | iexpression}

Select case sensitivity of labels. If an expression is used, then a non-zero value sets case sensitive, while zero selects insensitive. This is typically used in source files generated by the reformatter. When converting source files that are case sensitive it is not known in advance whether there will be any case collisions, so the converted file has to assume that the resulting labels are case sensitive.

Expressions

As far as practical Z80Asm allows expressions from any style to be used in any style of source code, only restricting items to a particular style where otherwise there would be ambiguity.

Numeric Constants

In the following, the number of digits shown (as lower case letters) is illustrative only, and the number used (up to the maximum allowed by the data type) is optional.

%bbbbbbbb
bbbbbbbbB

Binary constant.

nnnnnnQ

Octal constant.

dddd
ddddD

Decimal constant.

&hhhh (MA style only)
#hhhh
dhhhH (the leading digit must be 0-9, so for example 0F123H)
0xhhhh

Hex constant.

"c"
+ASC"c" (MA style only)
    The ASCII value of a single character.

False is represented by zero, and true by all bits set.

String Constants

Strings may be enclosed in single or double quotes. The opening and closing quotes must match. A quote of the opposite style may be included. To include a quote of the enclosing style, double it. In ZASM style, bytes in a string may be specified in hex with the escape sequence \xx. A backslash is generated by doubling it.

Operators

The following operators are supported in expressions:

Operator	Rank	Description
Brackets	10	Sub-expression.
+	9	Unitary plus.
-	9	Unitary minus.
~	9	Bitwise not.
LOG2	9	Highest set bit
LOW	8	Low byte of value.
HIGH	8	High byte of value.
*	7	Multiply.
/	7	Divide.
MOD	7	Modulus.
SHL, <<	7	Shift left.
SHR, >>	7	Shift right.
+	6	Binary addition.
-	6	Binary subtraction.
LT, <	5	Less than.
LE, <=	5	Less than or equal.
EQ, ==, =	5	Equal.
NE, !=	5	Not equal.
GE, >=	5	Greater than or equal.
GT, >	5	Greater than.
NOT	4	Bitwise not.
AND, &	3	Bitwise and.
OR, !	2	Bitwise or.
XOR, ^	2	Bitwise xor.
End of expression	1	End of expression

Examples

To assemble the CFX-II ROM from Martin's code:

```
Z80Asm.py -b -l -p -s MA CFX-II-Main,281
```

You may find that there are a couple of typo's in the files from Dave's site.

To assemble the SDX ROM from the code on Andy's site:

```
Z80Asm.py -b -l -c 0x2000 -f 0x00 -s M80 SDXMAIN.MAC SDXDISC.MAC SDXFDSC.MAC
SDXBAS.MAC SDXFMT.MAC SDXSTAT.MAC SDXSD.MAC SDXRAM.MAC
```

The resulting file is less than 8KB long, free space in the ROM. Andy's makefile has another source file which adds padding to the end, and then uses the Linux routine "dd" to skip the space from the default COM start at 0x0100 and the start of ROM at 0x2000, and then copies exactly 8KB.