WINDOWS CONSOLE Z80 MAC ASSEMBLER MANUAL (SEP 2022)

Contents

1. Overview
2. Operating Instructions
3. Interfacing To CMM2
4. Source File
5. Comments
6. Multiple Instructions on a Line
7. Mnemonics & Instructions
8. Assembler Directives
9. Labels
10. Constants
11. Macros
12. Maths Parser
13. Register Aliasing and Off-setting
14. Options
15. Created Text File Layout

Overview

This Windows based Z80 macro assembler is based on the Digital Research Macro Assembler Manual and was created to be used with the CMM2 Z80 Simulator project. Note that the assembler can be freely used but there is no implied warranty or fitness for purpose as it was only created as part of a fun project for the CMM2.
CP/M 2.2 was initially placed in the public domain in 2001 and further clarified in July 2022 so this assembler should not infringe on any licence rights.
Note the Digital Research Manual should be used as the main reference and this manual covers off on some additional clarification and differences. Note this assembler does differ in creating some of the output debugging files, in that it does not create a PRN or SYM file, but does create a text file that serves a similar purpose to these files. Any queries on this assembler should be directed to GerryL on The Back Shed (TBS).

Operating Instructions

1. The first step is to create the assembly source file in a text editor such as Notepad++. These source files should be saved as an **.asm** file.
2. If using a Macro Library file as part of the assembly code then that also would need to be created. The Macro Library file must be in the same folder as the source assembly file.
3. Drop the assembly source asm file either onto the Z80 Macro Assembler icon or open the Assembler by clicking on the Z80 Macro Assembler icon and dropping the asm source file on the screen of the opened assembler as directed.

4. The assembler will display the file line count as it assembles each line on the first pass. If a Macro Library file is included then the display will indicate its opening, the library file line count and then its closing.
5. The assembler will then undertake the second pass to resolve any unknown constants and then create a **.hex** file (Intel format) and a **.txt** file. The .hex file can be used directly with CMM2 Z80 Simulator or CP/M. Note that for running a hex file directly from the Simulator then it would normally be written to start at address 0 by using the ORG 0 statement, while a CP/M file normally starts at 100H.
6. On completion the assembler will indicate the number of error(s) detected, with the line number of each error, and the total lines and instructions assembled, the size of the code and the time in milliseconds to assemble. The generated .txt file will show the assembled instruction, their address, comments and errors (see the section Created Text File Layout**)**

Below is a list of items that users should be aware of when using the Macro Assembler. It covers off on general information, items not mentioned in the Digital Research Manual and any differences between this assembler and the Digital Research assembler. Refer to the Digital Research Manual where relevant for further clarification.

Interfacing To CMM2

An additional instruction with a mnemonic of MMB 0 has been added to the assembler that allows you to easily interface to the CMM2 Z80 Simulator. This uses the instruction code of DD 00 which is not used by the Z80 or Intel 8080/85. Refer to the document Z80SimCMM2.pdf that came as part of this project.

Source File

The source file type can be .asm or .z80. If using Notepad++ then .asm is the preferred as it will contextualise the text.

Comments

1. Once the comment start character is encountered (semicolon character,':') the rest of the line is treated as comment,. This is different from Digital Research assembler as that can embed comments in a multi-instruction line, but that can make reading a source file difficult when using a context sensitive editor like Notepad++. A whole line can be treated as a comment if the first character of a line is an Asterisk ('*').

Multiple Instructions on a Line

1. Multiple instructions can be placed on a line if they are separated by the exclamation character ('!'). Note that comments cannot be embedded in between instructions, see comment section above.

Mnemonics & Instructions

1. The assembler will accept Zilog Z80 mnemonics (only documented) and also Intel 8080/85 mnemonics. To resolve the few ambiguous instructions between CPU's (e.g. JP XXXX is jump positive in 8080/85 while it's an unconditional jump in Z80) you can force the assembler to interpret which CPU to cater for using the OPTIONS (see OPTIONS below). The default mode is Z80.
2. All instructions and assembly directives are converted to upper case. Comments and text in single quotes are left as is.
3. When using the instruction that has (IX+d) or (IY+d) the d value can be a LABEL or number (+ or -) BUT cannot be an equation or logical operator. The value is fixed at assembly time. Note that the value of the LABEL could be set using an equation or a SET directive which would allow it to be changed if used in an inline macro, but regardless it is fixed at assembly time.
4. In the RST Z80 instruction if the reset number is a LABEL value it does not need to be defined prior to using that instruction.

Assembler Directives

1. A space must follow an assembler directive.
2. In addition to the standard assembler directives these additional directives are also recognised.
   - DEFB is the same as DB.
   - DEFW is the same as DW.
   - DEFS is the same as DS.
3. PAGE is not supported (that is not referring to the CMM2 PAGE).
4. If DS has a comma after the quantity of bytes followed by a value, then the memory will be filled with that value, e.g. LABONE: DS 20, 'A' will result in twenty bytes of hexadecimal 41 (letter A)

Labels

The assembler does conform to the Digital Research Manual Label naming conventions so the below items are for further clarification.

1. A label that defines an address must have a colon as the last character. An instruction referring to that label does not have a colon. Labels that define a constant do not have a colon. Address labels do not need to be defined prior to use as they will get resolved on the second pass.
2. A label must start with an alpha character, an '?' or a '@'. Note the '@' has a special meaning in a MACRO so should only be used there. The '?' is used in generating LOCAL labels. A label can have the dollar sign '$' and the underscore character '_' within the label. The '$' character is ignored in resolving labels and it was used mainly as a separator to assist in readability. Note that the dollar symbol is also used as the current assembly address in a constant.

Constants

1. As per Digital Research's Manual assembler constants can use the dollar character to assist in readability, e.g. 0101$1011 is treated as 01011011.
2. Constants will evaluate to a 16 bit value, however floating point numbers and mathematical functions can be used in creating the constant. The maths equation parser uses double floating point values to calculate the value then converts it back to a 16 bit number. For example, the following source code is acceptable,

   ```
   DEGREES EQU 45
   NUME EQU (1000* 3.141592654 * DEGREES/180.0)
   COSE EQU cos( 3.141592654 * DEGREES/180.0 )*1000
   SINE EQU sin(3.141592654 * 45.0/180.0)*1000
   TANE EQU tan(3.141592654 * 45.0/180.0) * 3e8H
   COTE EQU atan(3.141592654 * 45.0/180.0) * 1000
   SQRTE EQU sqrt(1600/4)
   SQRE EQU sqr(10+10)
   POWE EQU pow(10,2)
   ```

3. Constants can be expressed as decimal (1234 or 1234D), octal (4774O), hexadecimal (1234H) and binary (01110111B). Note any hexadecimal number starting with an alpha must be preceded by 0 (e.g. 0FFFFH)
4. The symbol '$' is the current program counter address and can be used in an equation. It can be placed directly next to the characters +,-,(,), but must be separated by a space from a label, logical and shift mnemonic.
5. It is best to use brackets in an equation to ensure precedence behaves as expected.
6. Logical operators (AND, NOT, SHL etc) must be separated from a defined label by a space or bracket but can be next to numbers. For example, 6 AND 5 can also be written as 6AND5, similarly, the shift operator can be used as SHR 3 or SHR3.
7. Constants used after the conditional assembly directives of IF must evaluate to a number and the IF must have an ENDIF directive for each IF.
8. The assembler directives of HIGH and LOW can be used with labels. These will convert the high or low byte of a 16 bit number into a byte. They precede the LABEL they act on, e.g HIGH 0FF9EH will create the byte FF.
9. Note that the assembly parser is not going to resolve complex instructions involving equations in brackets where brackets are part of the instruction mnemonic. Where the instruction involves brackets to indicate loading of data at an address ensure the value in the bracket is a number or a defined constant, for example:
   a. LD HL, TABLE_ONE + (15*10) will resolve correctly to the direct HL load code of 21XXXX, where XXXX is the calculated value.
   b. LD HL, (TABLE_ONE + (15*10)) will throw an error as it could be interpreted as a direct load of a constant value like above with additional use of outer brackets or a load of data at an address. To

overcome this and not generate any additional assembly instruction code use the following,

nOFFSET equ TABLE_ONE + (15*10)

LD HL, (nOFFSET). This will resolve correctly to 2AXXXX

Macros

1. **MACLIB**. The Macro Library file following this directive must be in the same folder as the assembly source code file.
2. **LOCAL** The Local pseudonym is supported in macros, however this assembler will automatically make any label in a macro unique and as such it is not required to use Local. For example, it is easier to understand the created text file where labels are named relevant to the program than the LOCAL auto generated label of ??0001 and ??0002 etc. Note the LOCAL statement cannot be ignored by using an IF statement as LOCAL is set up during the template line collection phase of pass one and not on the MACRO invocation. Comment LOCAL out if it is in the source code you want to use but want to prevent it being used.
3. **Parsed Parameter.** Parameters in a stored or in-line macro that are to be changed to the parsed parameter values in the actual source code must be preceded by the ampersand (&) character. This is different from Digital Research Manual as they specify only parameter change outs within single quotes need the preceding &. The reason I went down this path of requiring a '&' on ALL changeouts is that this way it forces the programmer to consciously define what parameters to change and in addition some Z80 code I have seen on-line use the & on all parameter change outs. The actual parsed parameters in the MACRO definition statement do not have the preceding &. Parsed parameter names in the macro definition statement must be just alpha characters. Also note as per Digital Research Manual a parameter that is to be changed out in single quotes must be in capitals
4. **Labels.**
    a. Macro address labels that start with an @ character have their address fixed to a single value for all invocations. This makes it easy to create subroutine code in a self-redefining MACRO that can then be used many times over without re-creating the subroutine section. This is a powerful tool to create a library of routines. Refer to the Digital Research Manual section on Redefinition Macros.
    b. LABEL values defined using the EQU and SET directives in a Macro Library file that are defined outside a macro are created on pass one at the time the Macro Library is read. Labels defined within the Macro are created at the time of invocation.
5. **Constants.** A LABEL value used in a MACRO must be defined prior to its invocation.

6. **REPT MACRO.** The parameter parsed (i.e. number of repeats) must be a number or a label that equates to a number. An equation cannot be parsed, however a LABEL that has its value defined by an equation is acceptable.
7. **IRCP MACRO.** The character list parsed to the MACRO starts from the very first character past the comma. If the first character is a space then that is included in the parameter list.
8. **ENDM**. Will reset any active conditional assembly directives of IF and ELSE to not active. ENDM should be on its own line.
9. **EXITM**. This will only be evaluated if the IF directive is true, it is not assessed on an ELSE directive.
10. **Nesting**. In-line macros can be nested in stored macros but you cannot nest in-line macros in an inline-macro. Stored macros can be nested in stored macros.
11. **NUL**. This directive must be separated by a white space either side of it and ONLY one NUL is recognised on a line. Like EXITM the NUL is only evaluated on an IF directive.
12. **% Character**. As per the Digital Research Manual this is used to force the parameter when used in single quotes to be a value, it must be placed directly next to the start of the parameter.
13. **Up Arrow**. This has special meaning for MACRO parameters. The ^ key (5E Hexi-decimal or 94 Decimal) is recognised by this assembler as an Up Arrow key in a macro parameter. Refer to Digital Research Manual section on Parameter Evaluation Conventions

Maths Parser

This assembler uses a modified version of Math Parser for C++ | Math Parsers
It supports the following:

1. Arithmetic operators: +-/*
2. Logical operations: NOT, XOR, OR, AND, MOD, SHR, SHL
3. Equality operators: GE, LE, NE, EQ
4. Mathematical functions: SQR, SIN, COS, ATAN, TAN, EXP, LN, LOG, SQRT, ABS and POW. These maths functions may be useful if defining pixel location on the CMM2 display, for example clock hand locations.

Note:

- Brackets should be used in an equation to ensure the correct precedence.
- All equations are evaluated as double floating-point values but are returned to the assembler as a 16 bit integer so if needed the user should ensure that to maintain the required accuracy it may be necessary to multiply by a constant for some values/constants in the equation.

Examples of using maths functions is listed under Constants section above.

Register Aliasing and Off-setting

It is possible to substitute a label for a register in an instruction. For example,

```
MyVar EQU A
MyPtr EQU HL
```

```
LD MyVar, 55        ; load register A with 55
INC MyPtr           ; increment HL
```

The alias must be declared prior to use with an EQU directive. I personally think this actually creates confusing assembly code, especially if you're trying to debug it in STEP mode. It has been included in the assembler as Digital Research used register aliasing in one place in the CP/M source code. To use register aliasing, it must be enabled in the OPTIONS.

In addition, registers can be offset by a constant. In the instruction coding of the Z80 and 8080/85 registers have set values of:

B Reg = 0
C Reg = 1
D Reg = 2
E Reg = 3
H Reg = 4
L Reg = 5
A Reg = 7

Therefore, it is possible to write an instruction as,

```
LD B, C+1
```

This is equivalent to LD B, D.

Why you would want to write confusing code like this I have no idea but the Digital Research assembler will do it so I have included it. To use register off-setting it must be enabled in the OPTIONS.

Options

The OPTION directive can be used to force the assembler to process the source file in a pre-determined way or display additional information. Options can be combined and will remain in force until the next OPTIONS directive is encountered. To revert back to the default, use the directive OPTIONS with no parameters, which also means a missing option will set it to the default for that option. Note that the set of Assembly Parameters that are defined in Digital Research Manual, section 10, are not supported as most are redundant for this assembler's use.

Recognised options are:

1. **\8080** or **\8085** will force the assembler to treat all ambiguous instructions such as JP, CP, CPI and RLC as 8080/85 instructions. Normal default is Z80.
2. **\CONSOLE_TXT_ON.** Will duplicate the text file to the console screen. Note this will slow down the second pass significantly.
3. **\REG_SUB** Will allow register name substitution (aliases), see section on Register Aliasing and Off-setting.
4. **\REG_OFFSET** Will allow register offsetting, see section on Register Aliasing and Off-setting.
5. **\NOT_LIB_LIST**. Will prevent the MACRO definitions in a Macro Library being listed in the text file, however any invocations will be listed.

An OPTION example could be,

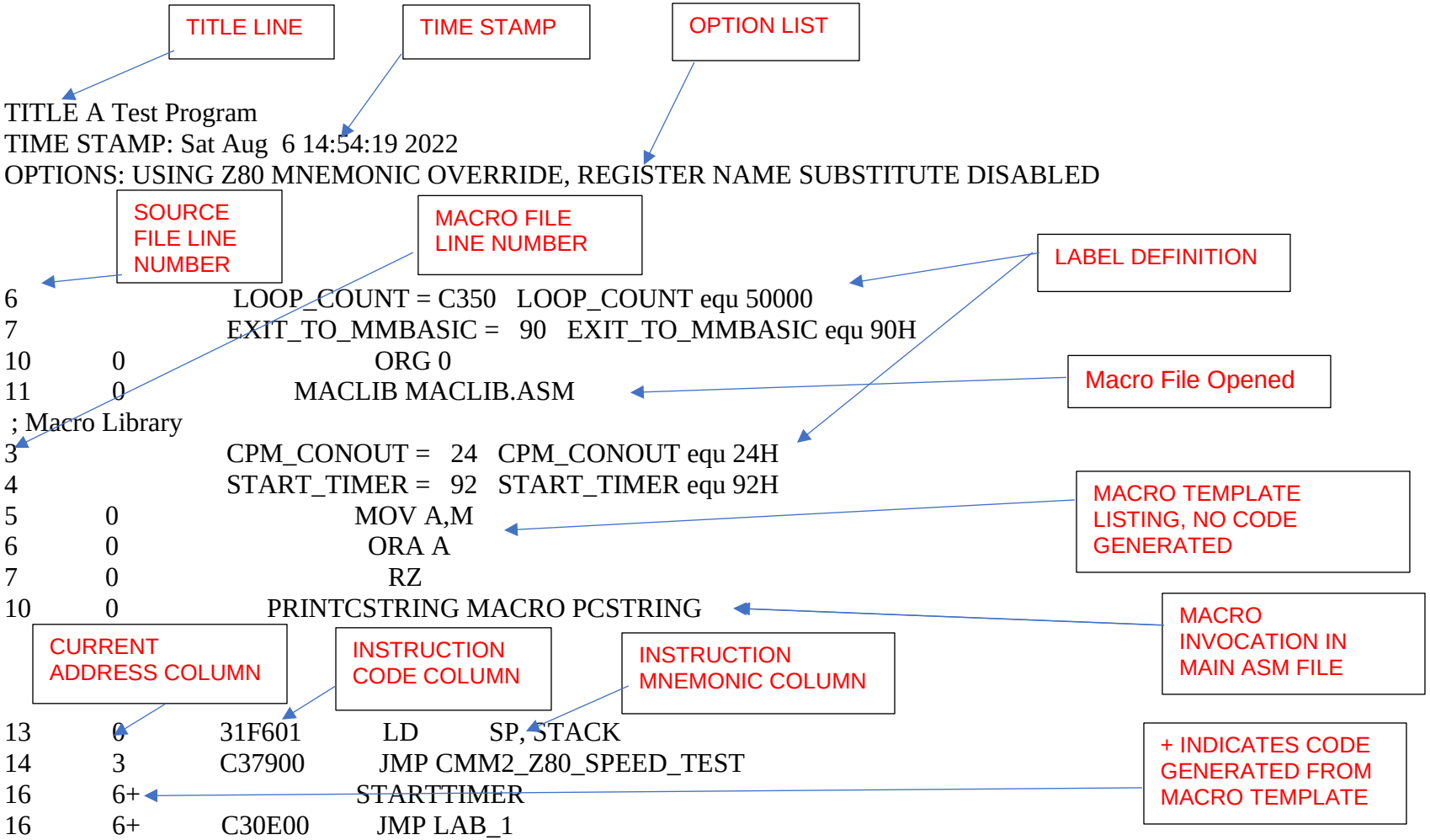
OPTIONS \8080\REG_SUB\NOT_LIB_LIST. It is not necessary to have spaces between options, spaces would be ignored but it is essential to have the same characters in an option.

To reset to the defaults during any file assembly just use the directive OPTIONS. Note on starting the assembler all OPTIONS default.

Note that the generated Text File will list if Z80 or 8080/85 is the override and if Register Substitution or Register Offset are active or disabled.

Created Text File Layout

1. In addition to the details listed below the Text File also list at the bottom of the file all the Labels in alphabetical order with their values, line number where defined and what type of Label (e.g. EQU or Function address etc).
2. Page 9 below shows the layout of a Text File.
3. The first line of the Text File is the Title (if used), regardless of where the TITLE statement is in the source.
4. The second line is the time stamp of the assembly.
5. The third line is the last options set (normally there is only one OPTION directive used).
6. The current address details are written with MSB to the left and LSB to the right while the instruction details are listed how they appear in memory, the lowest address byte is on the left.




```
TITLE A Test Program
TIME STAMP: Sat Aug  6 14:54:19 2022
OPTIONS: USING Z80 MNEMONIC OVERRIDE, REGISTER NAME SUBSTITUTE DISABLED
6                       LOOP_COUNT = C350   LOOP_COUNT equ 50000
7                       EXIT_TO_MMBASIC =   90   EXIT_TO_MMBASIC equ 90H
10          0                       ORG 0
11          0                   MACLIB MACLIB.ASM
 ; Macro Library
3                       CPM_CONOUT =   24   CPM_CONOUT equ 24H
4                       START_TIMER =   92   START_TIMER equ 92H
5           0                       MOV A,M
6           0                       ORA A
7           0                       RZ
10          0                   PRINTCSTRING MACRO PCSTRING
13          0           31F601          LD          SP,STACK
14          3           C37900          JMP CMM2_Z80_SPEED_TEST
16          6+                      STARTTIMER
16          6+          C30E00          JMP LAB_1
```