

Z80/R800 macro assembler

ZMA

1.	はじめに	5
2.	インストール	5
3.	実行	5
4.	ニーモニツクの違い	5
5.	未定義命令	5
6.	R800 命令	5
7.	疑似命令	5
1.1.	DEFB 命令	6
1.1.1.	書式	6
1.1.2.	機能	6
1.2.	DEFW 命令	6
1.2.1.	書式	6
1.2.2.	機能	6
1.3.	DEFD 命令	6
1.3.1.	書式	6
1.3.2.	機能	6
1.4.	DEFS 命令	6
1.4.1.	書式	6
1.4.2.	機能	6
1.5.	ORG 命令	6
1.5.1.	書式	7
1.5.2.	機能	7
1.6.	INCLUDE 命令	7
1.6.1.	書式	7
1.6.2.	機能	7
1.7.	ADD_INCLUDE_PATH 命令	7
1.7.1.	書式	7
1.7.2.	機能	7
1.8.	MACRO 命令	8
1.8.1.	書式	8
1.8.2.	機能	8
1.9.	SCOPE 命令	10
1.9.1.	書式	10
1.9.2.	機能	10
1.10.	REPEAT 命令	10
1.10.1.	書式	10
1.10.2.	機能	10
1.11.	IF 命令	11
1.11.1.	書式	11
1.11.2.	機能	11

1.12.	MESSAGE 命令	12
1.12.1.	書式	12
1.12.2.	機能	12
1.13.	ERROR 命令	12
1.13.1.	書式	12
1.13.2.	機能	12
1.14.	ローカルラベル	12
1.14.1.	書式	12
1.14.2.	機能	12
1.15.	グローバルラベル	13
1.15.1.	書式	13
1.15.2.	機能	13
1.16.	ローカルラベル代入	13
1.16.1.	書式	13
1.16.2.	機能	13
1.17.	グローバルラベル代入	14
1.17.1.	書式	14
1.17.2.	機能	14
8.	式	14
9.	文字列式	15
文字列に対してもある程度の演算を行うことができる。		15
		15
		15
例えば、“abc” == “abc” は 1 になる。“abc” == “xyz” は 0 になる。		15
“ab” * 3 は、“ababab” になる。		15
“ab” + 3 は、“ab3” になる。		15
		15
式と文字列式は、特に区別が無く混在させることができるため、下記のような記述も可能。		15
“ab” * (3 + 2)		15
10.	コメント	15
; より右はコメントと見なし、無視される。		15
		16
11.	エラーメッセージ	16
		18
履歴		18
2019/06/26	t.hara v1.0 初版	18
2019/06/27	t.hara v1.0 文字列式の説明を追加	18
2019/06/29	t.hara v1.0 エラーメッセージの説明を追加	18
		18

1. はじめに

ZMA は、Z80/R800 用のコードを生成するクロスアセンブラです。ニーモニックで記述されたテキストファイル进行处理して機械語コードを生成します。また、いくつかの疑似命令を備えています。

MSX を主なターゲットとしていますが、Z80/R800 を搭載している別のシステム向けにも利用可能です。

2. インストール

ZMA は、インストーラーを用意していない。zma.exe と、include フォルダを好きな場所に配置し、zma.exe のある場所に実行パスを通せばインストール完了である。

3. 実行

zma.exe には 2 つの引数を指定する。1 つ目は入力ファイル名。2 つ目は出力ファイル名。

引数を指定せずに実行すると、簡単なヘルプが表示される。

4. ニーモニックの違い

ZMA では、構文解析の都合等によりザイログニーモニックを一部改変しています。違いを下記の表にまとめる。

Zilog ニーモニックと ZMA の違い

Zilog ニーモニック	ZMA	備考
LD A, (HL)	LD A, [HL]	アドレス参照は、数式の優先順位指定の() と区別するために [] を使う。
JP (HL)	JP HL	レジスタによる飛び先アドレス指定は () で囲まない。
LD A, (IX+0)	LD A, [IX]	[IX+0] を [IX] と記述できる。IY も同様。
SUB B	SUB A, B	演算命令の記述で A を記述する。

5. 未定義命令

Z80 の未定義命令にも対応している。IXH, IXL, IYH, IYL を利用可能である。

6. R800命令

R800 の乗算命令に対応している。mulub, muluw を利用可能である。

7. 疑似命令

ZMA にはいくつかの疑似命令を利用可能になっている。本章では、各疑似命令について個別に説明する。

1.1. DEFB命令

1.1.1. 書式

DEFB 式

1.1.2. 機能

式の評価結果を 8bit 値と見なし、下位 8bit を記述箇所に配置する。

式は、カンマで区切って複数記述できる。

複数記述した場合は、左から順に 8bit 値として配置される。

1.2. DEFW命令

1.2.1. 書式

DEFW 式

1.2.2. 機能

式の評価結果を 16bit 値と見なし、下位 16bit を記述箇所に配置する。

式は、カンマで区切って複数記述できる。

複数記述した場合は、左から順に 16bit 値として配置される。

バイトオーダーはリトルエンディアンであり、下位 8bit、上位 8bit の順で配置される。

1.3. DEFD命令

1.3.1. 書式

DEFD 式

1.3.2. 機能

式の評価結果を 32bit 値と見なし、下位 32bit を記述箇所に配置する。

式は、カンマで区切って複数記述できる。

複数記述した場合は、左から順に 32bit 値として配置される。

バイトオーダーはリトルエンディアンである。

1.4. DEFS命令

1.4.1. 書式

DEFS 文字列式

1.4.2. 機能

文字列式の評価結果を配置する。

1.5. ORG命令

1.5.1. 書式

ORG 式

1.5.2. 機能

この記述箇所を、式の評価結果の値のアドレスであると見なすようになる。

例えば、下記の 2 つのプログラムは、出力されるコードが異なる。

ORG	0
JR	0
ORG	50
JR	0

ともに 2 バイトのコードを生成するが、上の記述の場合 0 番地に配置されているつもりで 0 番地へ相対ジャンプするコードを生成するため、生成コードは 0x18 0xFE となる。下の記述の場合 50 番地に配置されているつもりで 0 番地へ総体ジャンプするコードを生成するため、生成コードは 0x18 0xCC となる。

生成ファイルの先頭から何バイト目であるかは独立して、各行が何番地に配置されているかを認識しており、ORG はこれを変更するための疑似命令となる。

これは、例えば ROM 上のコードを、DRAM へコピーして使う場合など「元々格納されているアドレス」と「実際に実行されるアドレス」が異なる場合にも対応できるようにするための仕組みである。

1.6. INCLUDE 命令

1.6.1. 書式

INCLUDE 文字列式

1.6.2. 機能

文字列式の評価結果による文字列をファイル名と認識し、そのファイルを読み出して記述箇所に挿入する。

1.7. ADD_INCLUDE_PATH 命令

1.7.1. 書式

ADD_INCLUDE_PATH 文字列式

1.7.2. 機能

文字列式の評価結果による文字列をディレクトリ名と認識し、INCLUDE 命令によるファイル名指定の前につけるパス名リストにそのディレクトリ名を追加する。

デフォルトでは、カレントディレクトリと、ZMA.exe が格納されているパスの中にある include ディレクトリがリストに含まれている。これ以外の場所から INCLUDE したい場合に使用する命令である。

ZMA.exe はオブジェクトファイル生成とリンクの過程を省略するため、INCLUDE が各種ライブラリのリンクの役割を兼ねる。ライブラリのソースの中に特定のパスを記述したくないため、リンクする側のソースに `ADD_INCLUDE_PATH` を記述して「ライブラリのソース置き場」を指定することで、使い回すライブラリの中にパスの記述をしなくて済む。

1.8. MACRO命令

1.8.1. 書式

```
マクロ名 MACRO 引数リスト
    文
ENDM
```

1.8.2. 機能

文に記載の内容を「マクロ名」の記述で表記できるようにする。

文は複数行にわたる記述も可能である。

```
HOGE MACRO
    LD    D, H
    LD    E, L
ENDM
```

このような記述の後に HOGE と記述すると、LD D, H/LD E, L の 2 行に置き換えられる。
マクロには引数を持たせることができる。

```
MOGE MACRO n
    LD    A, n
ENDM
```

n は引数であるが、MOGE 100 のように記述すると LD A, 100 と展開されるようになる。

当然 MOGE 10+20 のような式も記述可能であるが、単純置換になるため下記のような記述は注意が必要である。

```
FOO MACRO m
    LD    A, 10*m
ENDM
```



```
FOO 1+2
```

FOO 1+2 は、LD A 10*1+2 に展開されるため、最終的に LD A, 12 になる。下記のように記述するのが安全である。

```
FOO MACRO m
    LD    A, 10*(m)
ENDM
```

```
FOO 1+2
```

引数を複数持たせることもできる。

```
BAR MACRO n, m
    LD    A, n
    LD    B, m
ENDM
```

引数の名前に @ をつけると、その引数は文字列に変換されるようになる。

```
LDM MACRO @D, @S
    IF D == "BC"
        IF S == "DE"
            LD B, D
            LD C, E
        ELSEIF S == "HL"
            LD B, H
            LD C, L
        ELSE
            ERROR "LDM macro isn't support the source "+S+"."
        ENDIF
    ENDIF
ENDM

LDM BC, DE
LDM BC, HL
```

このように、あたかもそのような命令があるかのような記述が可能となる。

文字列に変換する際、小文字は大文字に変換される。単語間は半角スペース 1 つで区切られる。例えば、a+B + 3 と書かれていても、“A + B + 3”のように変換される。+の前後に半角ス

ペースが1つ入る。

マクロ名として、実際に存在する命令と同じ名前をつけると、その命令が使えなくなるので注意すること。

MACRO ~ ENDM, REPEAT ~ ENDR, IF ~ ENDIF の中でマクロを宣言することはできない。

1.9. SCOPE命令

1.9.1. 書式

```
SCOPE   スコープ名
        文
ENDSCOPE
```

1.9.2. 機能

SCOPE から ENDScope までの間を、スコープ名で示したスコープに指定する。

この中で宣言されたローカルラベルは、この中でのみ有効となる。

ネストも可能である。

1.10. REPEAT命令

1.10.1. 書式

```
REPEAT ループ変数名, 式
        文
ENDR
```

1.10.2. 機能

式の評価結果で示される回数だけ、文を繰り返し出力する。

その際、ループ変数は 0 から順に 1 ずつインクリメントされ、式-1 まで変化する。

文の中の式でループ変数を利用することも可能である。

```
REPEAT I, 2
    REPEAT J, 3
        DEFB I*5 + J
    ENDR
ENDR
```

これにより、0, 1, 2, 5, 6, 7 の 6byte が生成される。

ループ内は、専用のスコープを形成するが、式はそのスコープの外側の扱いである。

ネストも可能である。

```
REPEAT I, 3
    REPEAT I, I+1
```

```
    DEFB I
ENDR
ENDR
```

これにより、0, 0, 1, 0, 1, 2 の 6byte が生成される。

内側の REPEAT の式である I+1 に登場する I は、外側の REPEAT のループ変数である。

DEFB I の I は、内側の REPEAT のループ変数である。

スコープが形成されているため、同じ名前でもループ回数は問題なく展開されるが、一番内側の式の中では、一番内側のループ変数しか使えなくなるため、式の中で使うループ変数名はユニークにしておかねばならない。

1.11. IF命令

1.11.1. 書式

```
IF 式
    文
ENDIF
```

```
IF 式 1
    文 1
ELSEIF 式 2
    文 2
ENDIF
```

```
IF 式
    文 1
ELSE
    文 2
ENDIF
```

1.11.2. 機能

式の評価結果が 0 以外の場合に文を展開する。

主に MACRO の中で、引数の値に応じて生成されるコードを変更したい場合に利用する命令であるが、MACRO の外でも利用可能である。

ELSEIF を含む記法では、式 1 が 0 以外の場合に文 1 を展開し、式 1 が 0 かつ式 2 が 0 以外の場合に文 2 を展開する。

ELSE を含む記法では、式が 0 以外の場合に文 1 を、0 の場合に文 2 を展開する。

ELSEIF は複数使うことができ、ELSE を ELSEIF と組み合わせることもできる。

```

IF 11 == 100
    ld a, 100
elseif 11 == 200
    ld a, 100
    ld b, 2
elseif 11 == 300
    ld a, 100
    ld b, 3
else
    ld b, 2
endif

```

1.12. MESSAGE命令

1.12.1. 書式

MESSAGE 文字列式

1.12.2. 機能

ログに文字列式の評価結果を出力する。

1.13. ERROR命令

1.13.1. 書式

ERROR 文字列式

1.13.2. 機能

ログに文字列式の評価結果を出力し、エラーが発生したことにする。

1.14. ローカルラベル

1.14.1. 書式

ラベル名:

1.14.2. 機能

ラベル名に、記述位置のコードアドレスを紐つける。

ラベル記述位置のスコープに影響される。

スコープ内の場合、そのスコープ内からしか参照できない。

また、別のスコープの中に同じ名前のラベルがあっても問題ない。

```

        ORG    100
LABEL1:
        SCOPE  HOGE

```

```
ORG 200
LABEL1:
MESSAGE "LABEL1 = " + LABEL1
ENDSCOPE
MESSAGE "LABEL1 = " + LABEL1
```

これをアセンブルすると、下記のようなメッセージが表示される

```
LABEL1 = 200
LABEL1 = 100
```

1.15. グローバルラベル

1.15.1. 書式

ラベル名::

1.15.2. 機能

ラベル名に、記述位置のコードアドレスを紐つける。

スコープには影響されず、コード上のどこからでも参照可能である。

サブスコープ内に同名のローカルラベルが存在する場合、ローカルラベルが優先される。

```
ORG 100
LABEL1::
MESSAGE "LABEL1 = " + LABEL1
```

これをアセンブルすると、下記のようなメッセージが表示される

1.16. ローカルラベル代入

1.16.1. 書式

ラベル名 = 式

1.16.2. 機能

ラベル名に、式の評価結果を紐つける。

記述の順序は無関係であり、1つのラベルには1つの値しか紐つけられない。

式は文字列式でもかまわない。

```
LABEL1 = LABEL2
LABEL2 = " hoge "
```

1.17. グローバルラベル代入

1.17.1. 書式

ラベル名 := 式

1.17.2. 機能

使い方はローカルラベル代入と同様である。

スコープ内で使用しても、グローバルなラベルに対する紐つけになる点のみ異なる。

8. 式

ZMA では、様々な場所に式を記述できる。

下記の演算子を利用できる。

演算子と優先順位

優先順位	演算子	名前	意味
1	+	正值	なにもしない。
1	-	負値	符号を反転する。
1	!	否定	0は 1に。0以外は 0 になる。
1	~	ビット反転	ビットごとに 0は1に、1は0になる。
1	()	優先記述子	()の内側を先に演算する。
2	*	乗算	掛ける
2	/	除算	割る
2	%	剰余	割ったあまり
3	+	加算	足す
3	-	減算	引く
4	<<	左ビットシフト	左ビットシフト
4	>>	右ビットシフト	符号付き右ビットシフト
5	<	小なり	左項が小さければ1, 小さくなければ0
5	>	大なり	左項が大きければ1, 大きくなければ0
5	<=	小なりまたは一致	左項が小さいか一致していれば1, 大きければ0
5	>=	大なりまたは一致	左項が大きいか一致していれば1, 小さければ0
6	==	一致	一致していれば1, 不一致なら0
6	!=	不一致	不一致していれば1, 一致なら0
7	&	ビットごとの論理積	ビットごとの論理積
8	^	ビットごとの排他的論理和	ビットごとの排他的論理和
9		ビットごとの論理和	ビットごとの論理和
10	&&	論理積	左右の項が両方とも 0以外 なら 1, 一方でも

			0 なら 0
		論理和	左右の項が一方でも 0 以外なら 1, 両方 0 なら 0

式の中では、下記の定数を利用できる。

利用可能な定数一覧

定数	意味
\$	記述行の開始コードアドレス
CODE_ADDRESS	\$ と同じ
FILE_ADDRESS	記述行のファイル内アドレス(先頭から何バイト目か)
ラベル名	ラベルの値

9. 文字列式

文字列に対してもある程度の演算を行うことができる。

演算子と優先順位(文字列式)

優先順位	演算子	名前	意味
2	*	乗算	反復。左項または右項は数値でなければならない。
3	+	加算	連結。左項または右項が数値なら 10 進数文字列にして連結。
6	==	一致	一致していれば 1, 不一致なら 0
6	!=	不一致	不一致していれば 1, 一致なら 0

例えば、“abc” == “abc” は 1 になる。“abc” == “xyz” は 0 になる。

“ab” * 3 は、“ababab” になる。

“ab” + 3 は、“ab3” になる。

式と文字列式は、特に区別が無く混在させることができるため、下記のような記述も可能。

“ab” * (3 + 2)

10. コメント

; より右はコメントと見なし、無視される。

11. エラーメッセージ

エラーメッセージ	意味
(' are not closed.	式の中で (に対応する) が存在しない。
Block processing is not close.	ブロック処理(IF,REPEAT,MACRO)が閉じられていない。
Cannot evaluate the expression(N)	defb/w/d で N番目の式が評価できない。
Cannot open include file 'FILE_NAME'.	INCLUDE に指定されたファイル 'FILE_NAME' が開けない。
Description of numerical value '0XABCDEF' is abnormal.	数値の記述が異常。 数値の記述と、ラベルの記述が繋がっていると1つの単語と誤認されてこのエラーになるケースがある。
Divided by zero.	除算演算子 / および % において、右側の項が 0 であり、値が確定しない。
ENDIF is not need parameters.	ENDIFに不正なパラメータ指定がある。
ENDR is not need parameters.	ENDRにパラメータの記述がある。
ENDSCOPE command has not parameter.	引数を持たない ENDSCOPE に引数が指定されている。
ENDSCOPE in wrong position.	ENDSCOPEの記述位置がおかしい。SCOPEしてないのにENDSCOPEしていたり、 MACROやIFの中にENDSCOPEだけ書いていたりすると発生する。
Illegal command	解釈不能な命令がある。
Illegal condition.	IF/ELSEIFの条件式に、値が確定しない式の記述がある。
Illegal ENDIF.	ENDIFが不正な位置に存在する。
Illegal ENDR.	不正な位置にENDRの記述がある。
Illegal expression.	数式の記述が異常。
Illegal operand	オペランドの値を確定できない。
Illegal parameter in ADD_INCLUDE_PATH.	ADD_INCLUDE_PATH命令のパラメータを評価できない。
Illegal parameter in MESSAGE.	MESSAGE命令のパラメータを評価できない。
INCLUDE command has only one parameter.	INCLUDE に2つ以上のパラメータが指定されている。
Invalid expression.	式の記述に異常があり、評価できない。
Invalid operator '-'	二項演算子 - の右側に値が記述されていない。あるいは、左右の項が評価不能な値である。

Invalid operator '!'.	単項演算子 ! に対する値が記述されていない。
Invalid operator '!='	二項演算子 != の右側に値が記述されていない。あるいは、左右の項が評価不能な値である。
Invalid operator '%'	二項演算子 % の右側に値が記述されていない。あるいは、左右の項が評価不能な値である。
Invalid operator '*'	二項演算子 * の右側に値が記述されていない。あるいは、左右の項が評価不能な値である。
Invalid operator '-'.	単項演算子 - に対する値が記述されていない。
Invalid operator '/'	二項演算子 / の右側に値が記述されていない。あるいは、左右の項が評価不能な値である。
Invalid operator '~'.	単項演算子 ~ に対する値が記述されていない。
Invalid operator '+'	二項演算子 + の右側に値が記述されていない。あるいは、左右の項が評価不能な値である。
Invalid operator '+.'	単項演算子 + に対する値が記述されていない。
Invalid operator '<'	二項演算子 < の右側に値が記述されていない。あるいは、左右の項が評価不能な値である。
Invalid operator '<<'	二項演算子 << の右側に値が記述されていない。あるいは、左右の項が評価不能な値である。
Invalid operator '<='	二項演算子 <= の右側に値が記述されていない。あるいは、左右の項が評価不能な値である。
Invalid operator '=='	二項演算子 == の右側に値が記述されていない。あるいは、左右の項が評価不能な値である。
Invalid operator '>'	二項演算子 > の右側に値が記述されていない。あるいは、左右の項が評価不能な値である。
Invalid operator '>='	二項演算子 >= の右側に値が記述されていない。あるいは、左右の項が評価不能な値である。
Invalid operator '>>'	二項演算子 >> の右側に値が記述されていない。あるいは、左右の項が評価不能な値である。
Label name cannot be string.	ラベル名として文字列が指定されている。
Message not found in MESSAGE.	MESSAGE命令にメッセージの指定が無い。
Mismatched number of arguments.	マクロに対する引数指定が少なすぎる。
Multiple arguments of the same name 'XXXX' exist.	マクロ宣言において、引数名 XXXX が複数指定されている。
Must be set include file name.	INCLUDEにファイル名指定が無い。
Must be set scope name.	SCOPEにスコープ名の指定が無い。
Offset value is out of range (N).	[IX+n], [IY+n] の n が、-128~127 の範囲を超えている。N は実際の n の値。
Path not found in ADD_INCLUDE_	ADD_INCLUDE_PATH命令にパスの指定が無い。

PATH.	
SCOPE command has only one parameter.	SCOPE に2つ以上のパラメータが指定されている。
Scope is not closed.	SCOPEに対応するENDSCOPEが存在しない。
Scope of ENDR does not exist.	REPEAT と ENDR の対応関係がとれない。MACROの中にENDRだけ記述されている場合など。
There are declarations of the same label 'XXXX' in multiple places.	ラベル XXXX が複数箇所で宣言されている。
There is an ELSE description at an incorrect position.	IF無しのELSEがある。
There is an ELSEIF description at an incorrect position.	IF無しのELSEIFがある。
There is an extra account that cannot be interpreted.	解釈不能な余計な記述がある。例えば LD A, B AA A のような場合。余計な記述 AAA がある。
Too many arguments for XXXX.	マクロXXXX に対する引数指定が多すぎる。
User error	ERROR命令で引数がない。

履歴

2019/06/26 t.hara v1.0 初版

2019/06/27 t.hara v1.0 文字列式の説明を追加

2019/06/29 t.hara v1.0 エラーメッセージの説明を追加