UNIMON for SAKI80

このモニタープログラムは、asano氏（https://electrelic.com/electrelic/node/1317）
が公開されているUniversal MonitorをベースにAki.H氏による大幅な拡張がなされてEMUZ80用に公開されているEMUZ80_Monitor Rev.B04を Super AKI-80 で動作するよう再ビルドしたものです。

unimonは多数のCPU上で動くように設計されており、unimonのZ80版をベースにミニアセンブラの追加やデバッグ機能を強化しています。
また、メニュー表示(#コマンド)によるROM内プログラムの実行やRAMへのロード(Lコマンド)など効率的なコマンドの強化が行われています。

<コマンド>

- **?** コマンドヘルプ

  コマンドヘルプで以下のヘルプが表示されます。

```
? :Command Help
#L|<num> :Launch program
A[<address>]: Mini Assemble mode
B[1|2[,<adr>]]:Set or List Break Point
BC[1|2]:Clear Break Point
D[<adr>]:Dump Memory
DI[<adr>][,s<steps>|<adr>]:Disassemble
G[<adr>][,<stop adr>]:Go and Stop
L[G|<offset>]:Load HexFile (and GO)
P[I|S]:Save HexFile(I:Intel,S:Motorola)
R[<reg>]:Set or Dump register
S[<adr>]:Set Memory
T[<adr>][,<steps>|-1]: Trace command
TM[I|S]:Trace Option for CALL
TP[ON|OFF]:Trace Print Mode
```

- **#L** コマンド
  SAKI80MONITORは、32KBのROMを内蔵した形になっているので、GRANT's BASIC版のZ80 BASIC Ver 4.7bCとTiny Basic及びMicro Pascal
  を組み込んであります。

  登録ソフトの表示

```
1. GBASIC Start
2. GBASIC Restart
3. Palo Alto Tiny BASIC Start
4. Micro Pascal Start
```

- **#番号**

選択したソフトを実行します。

- **G(Go)** コマンド

  G[addr][,stop addr]
  ストップアドレスの指定を追加

  ストップアドレスでモニターに制御が戻ります。
  制御が戻った時に、レジスタを表示します。

  ROM内のアドレスでストップさせることはできません。

- **L(Load)** コマンド

  HEXファイルのロード
  コマンドの実行後、エディタ等でHEXファイルを開きコピー&ペーストすればロードさせることができます。
  HEXファイルの1ライン入力毎に「. 」を表示します。

- **LG(Load and Go)** コマンド

  HEXファイルのロード後、ロードアドレスの先頭にジャンプします。

- **R(Register)** コマンド

  F,F'レジスタを数値ではなく、ビジュアル化し直感的にフラグの状態が分かるよう
  にしました。

デバッグコマンド(Debug)

- **B(Break)** コマンド

  B[1 | 2 [,break point address]]

  ブレイクポイント設置コマンド。2か所設定できます。
  プログラムを実行すると、ブレークポイントでモニターに制御が戻ります。
  戻った時に、レジスタの表示を行います。
  break point addressの指定が無い時は、現在のブレークポイントの状態を
  表示します。
  ROM内にブレークポイントを設定することはできません。

```
例
]B1,C000
]B2,C001

]B
BP(1):C000
BP(2):C001

]B1
BP(1):C000
```

- **BC[1 | 2]**

  ブレークポイントの解除
  パラメータ指定なしの時は、ブレークポイント2か所とも解除

- **T(Trace)** コマンド

  T[address][,step数]

  指定されたアドレスから指定ステップ実行する。ステップ数（10進数）を省略すると、1ステップトレースする。アドレスを省略すると、現在のPCアドレスからトレースする。ROM内をトレースすることはできません。

  > 注）RST命令は、monitorで予約されているため、RST命令の次の命令までトレースできません。（RST命令実行し、リターン後にトレース続行となる）

```
例
]T C000, 10
```

> C000H番地から10ステップ実行し、モニターに戻ります。
> モニターに戻るとレジスタ表示し、PCのアドレスを逆アセンブルし表示します。（TP ON 時）
> step数に-1を指定すると、無限トレースモードとなりCTRL+Cキーで中断するまでトレースを実行し続けます。

```
例
]T -1
```

> 現在のPCのアドレスからCTRL+Cが押されるまでトレースを続ける。

```
TP [ON | OFF]
```

> トレース実行後のレジスタ表示のON,OFFを制御します。パラメータ省略時は、表示モードを表示します。初期値は表示モード（TP ON）

```
TM [I | S]
```

> CALL命令のトレースモードの制御を行います。[I | S]省略時は、現在のトレースモードを表示します。（初期値）はステップインモード（TM I）

```
]TM I
```

> ステップインモード。CALLが実行された先のアドレスをトレースします。

```
]TM S
```

> スキップモード。CALL命令を実行し、リターン後にトレース続行となります。この機能を使うと、call先がROMでも実行できます。
> （ROM内CALL実行し、リターン後にトレース続行）

> ブレークポイントを指定したアドレスをトレースするときに、ブレークポイントが優先されトレースできませんでした。使い勝手が良くない為、トレースを優先し、ブレークポイントを指定しているアドレスを実行できます。

アセンブルコマンド（Minimal Assemble）

- **A[<address>]**

  指定したアドレスから、アセンブルモードになります

```
    .<CR>      アセンブルモードから、モニターに戻ります。
```

＜Minimal Assemblerの仕様＞

ニモニックは、Z80アセンブラの仕様に従います。

＜数字の扱い＞

- 数字は10進数、16進数を扱うものとする。
- A～Fで始まる数字には、前に0を付ける。
- 16進数は末尾にHを付ける。
- 数字は16ビット長で扱われる。0～65535
- 数字の冗長入力は7文字以上はエラーとなる。（最大0FFFFHの6文字）
- 有効ビット長が8ビットの場合、16ビット長の下位8ビットが有効で上位8ビットは無視される

＜疑似命令＞

1. **ORG <address>**

   指定されたアドレスへロケーションカウンタを進めます。

2. **DB [1バイト数値 | "文字列"]**

   1バイトの数値入力、または、"文字列"を入力します。
   DBはサブセットです。「,」で区切っての複数入力はサポートしていません。

3. **DW**

   2バイトの数値入力。
   DWはサブセットです。「,」で区切っての複数入力はサポートしていません。

計算式はサポートしていません。

- リラティブ数値の入力

  JR、DJNZで指定するリラティブ数値は、目的のアドレスを入力することで、自動的に計算されます。
  指定アドレスが範囲外の場合、エラーとなります。

逆アセンブルコマンド（Disassemble）

- **DI[<adr>][,s<steps>|<adr>]**

  指定したアドレスから、指定したアドレスもしくは指定したステップ数（10進）を逆アセンブルし、表示します。
  アドレス、ステップ指定を省略した場合、直前に逆アセンブル終了したアドレスから10ステップ表示します。表示中に何かキーを入力すると、表示を終了します。

```
例
DI C000,S30       C000H番地から30ステップ逆アセンブルします。

DI C000, C010     C000H番地からC010番地まで逆アセンブルします。

DI                直前に終了したアドレスから10ステップ逆アセンブルします。

DI C000           C000H番地から10ステップ逆アセンブルします。
```

API機能拡張

- **RST 08H**

  Aレジスタのキャラクタコードをシリアル出力

- **RST 10H**

  Aレジスタへシリアル入力

- **RST 18H**

  シリアルステータスチェック

  Zフラグ

  1：入力なし

  0：入力あり

- **RST 30H**

  Cレジスタに機能コードを入れて RST 30H を実行することで各種機能を呼び出せます。

```
        DW  API00       ; 00: CSTART
        DW  API01       ; 01: WSTART
        DW  CONOUT      ; 02: CONOUT
        DW  STROUT      ; 03: STROUT
        DW  CONIN       ; 04: CONIN
        DW  CONST       ; 05: CONST
        DW  API06       ; 06: PSPEC
        DW  HEXOUT4     ; 07: CONOUT HEX4bytes: input HL
        DW  HEXOUT2     ; 08: CONOUT HEX2bytes: input A
        DW  HEXOUT1     ; 09: CONOUT HEX1byte : input A
        DW  CLR_CRT     ; 10: Clear screen (ESC+[2)
        DW  GETLIN0     ; 11: GET a line (input HL : input buffer address)
        DW  SKIPSP      ; 12: SKIP Space
        DW  CRLF        ; 13: CONOUT CRLF
        DW  UPPER       ; 14: Lower to UPPER
        DW  RDHEX       ; 15: get hex number from chr buffer
                    ; input  HL : hex string buffer
                    ; output DE : hex number
                    ;      CF=1 : error, C, A = hex counts(1-4)
        DW  DEC_STR     ; 16: get decimal strings
                    ; input HL : return strings buffer addr.
                    ;       DE : 16bit binary
        DW  DIV16_8     ; 17: division 16bit / 8bit
        DW  MUL_8       ; 18: multiply 8bit * 8bit
```