MYCPU80（TK80回路）操作説明書

〒463-0067 名古屋市守山区守山2－8－14
パレス守山305
有限会社中日電工
TEL052-791-6254 Fax052-791-1391
E-mail thisida@alles.or.jp
Homepage http://www.alles.or.jp/~thisida/

目次

1章 基礎知識 3
1. はじめに 3
2. コンピュータの命令 3
2章 基本操作 7
1. LED表示 7
2. キーボード 7
2.1 データキー 8
2.2 ファンクションキー 8
2.2.1 ADRSSET（ADDRESS SET、アドレス セット） 8
2.2.2 RDINC（RDREAD INCRIMENT、リード インクリメント） 8
2.2.3 RDDEC（READ DECRIMENT、リード デクリメント） 8
2.2.4 WRINC（WRITE INCRIMENT、ライト インクリメント） 8
2.2.5 RUN（ラン） 8
2.2.6 RET（RETURN、リターン） 8
2.3 RESET（リセット） 9
3. プログラムの入力 9
3.1 アドレス 9
3.2 キー入力とLED表示 9
3.3 サンプルプログラムの入力 10
3.4 プログラムの実行 12
3章 プログラムデバッグの仕方 13
1. はじめに 13
2. ステップ動作 13
3. ブレイク動作 14
4. レジスタの確認 15
4.1 MYCPU80（8080）のレジスタ 16
4.2 MYCPU80（8080）のフラグ 16
4.3 スタック 17
4.4 ブレイク、ステップ操作でのレジスタの値の設定、確認方法 18
5. プログラムの終わり方 19
4章 プログラムのSAVE、LOAD 21
1. はじめに 21
2. プログラムのSAVEの仕方 21
2.1 DOS／Vパソコン側の操作 21
2.2 MYCPU80側の操作 21
3. プログラムのLOADの仕方 22
3.1 MYCPU80側の操作 22
3.2 DOS／Vパソコン側の操作 22
5章 I／O制御 24
1. はじめに 24
2. I／Oインターフェース回路に対するデータ入出力の方法 24
2.1 I／Oアドレス 24
3. スピーカの使用方法 25
6章 応用プログラム 26
1. OHAYO（オハヨー） 26

1．1 プログラムの説明 26
1．2 プログラムリスト 26
2． 電子オルガンプログラム 28
2．1 プログラムリスト 28
2．2 各キーと音との対応 29
2．3 操作 29

7章 メモリマップ・I／Oマップ 31
1． メモリマップ 31
2． システムワークエリア 31
3． RSTジャンプテーブル 32
４． Ｉ／Ｏマップ ３３
8章 モニタサブルーチン 34
1． はじめに 34
2． LED表示 34
2．1 セグメント表示バッファとLED表示の関係 34
2．2 セグメントデータ変換ルーチン 34
2．3 アドレスレジスタ、データレジスタ表示ルーチン 35
3． キー入力 35
3．1 キー入力ルーチン① 35
3．2 キー入力ルーチン② 35
4． タイマー 36
4．1 タイマールーチン①(4．727ms) 36
4．2 タイマールーチン②(9．432ms) 36
4．3 タイマールーチン③(28．307ms) 36
9章 モニタプログラムリスト 37

2011．12．11 Rev． 1．1(8j)

1章　基礎知識

1．　はじめに

ここではMYCPU80回路を使うために、最低これだけは知っていなければならない基本的な事柄について、簡単に説明します。
　ここに書いてあることは、マイクロプロセッサやプログラムなどについて、ある程度の知識をお持ちの方ならすでに知っていることばかりのはずですから、読みとばしていただいても構いません。

2．　コンピュータの命令

　コンピュータはプログラムがなければ動きません。
　プログラムは命令を順番に書いて並べたものです。コンピュータはメモリに書かれたプログラムの命令をひとつずつ読みだして、実行します。
　この場合の命令とはマシン語の命令のことです。
　マシン語というのはコンピュータが直接理解できる命令のことです。
　これに対してBASICなどの命令は、コンピュータが直接理解することはできません。
　BASICのような言語はインタプリタとかコンパイラなどの翻訳プログラムによってマシン語に直してから実行します。
　それではそのマシン語とは、どんな命令なのでしょうか。
　以下簡単に説明をします。
　なお以下の説明の大部分はコンピュータ全体に共通する事柄ですが、マシン語コードは8080固有のものです。

●コンピュータと2進法

　具体的な命令について説明する前に、2進法について理解しておく必要があります。
　2進法とは0と1しか使わない計算方法のことです。
　私達は一般に10進法を使っています。
　10進法では１＋１＝2です。しかし2進法では１＋１＝10になってしまいます。
　そんなべらぼうな、と思うかもしれません。
　しかしちょっと考えると私達も普段べらぼうとも何とも思わないで、10進以外の計算をしている場合があります。
　下の計算をよく見て下さい。
　　59＋1＝100
　ちょっと見にはべらぼうにみえますが、これに少し細工をして、こう表現してみたらどうでしょうか。
　　59＋1＝1　00
　さらにこうすれば、なあんだ、そうか、わかりますね。
　　59＋1＝1:00
　そうです。分や秒の計算は10進表記をしていますが、59の次は60にならずに上の桁（単位）に繰り上がる、60進法なのです。
　さてそこでもう一度先程の計算を見てみます。
　2進法の１＋１＝10は、10（十）ではなくて、1，0（イチ、ゼロ）と考えて下さい。仮に ,（カンマ）をつけましたが、実際には10進と同じように10と表記します。
　2進法では0と1しか使わないので、１＋１＝2ですぐに桁上がりをする結果、10になるのです。
　同様にして10進の3は2進では11、4は100と表現します。以下5は101、6は110、7は111、8は1000、9は1001になります。
　このように10進の1桁を表記するのに2進では4桁も必要になります。
　しかし桁数は増えても、コンピュータにはこのほうが都合が良いのです。
　電気には＋と－の2通りしかありません。＋を1 、－を0と考えて、コンピュータは計算をするのです。

●16進法

　ところで10進の10以上の数は2進ではどう表現されるでしょうか。
　9＋1＝10の計算は、2進では1001＋1＝1010になります。
　この調子で計算して行くと、20は10100 、50は110010、100は1100100 となります。これだけ大きい数になると、2進←→10進の換算も厄介ですし、0と1がだらだら続いていて、見ているだけで疲れてしまいます。
　コンピュータはこれでも良いのでしょうが、これではプログラムを組むのが大変です。
　そこで上の2進数を見易くするため、4桁毎に区切って表示してみます。
　10進の20→0001 0100 、50→0011 0010 、100→0110 0100 となって少し見易くなりました。

さてこうして4桁毎に区切って見ると、4桁の2進数はなんとなくそれぞれ10進数に置き換えができそうな気がします。

そこで置き換えてみるとつぎのようになります。

0001 0100 → 14 、0011 0010 → 32 、0110 0100 → 64

これが16進法なのです。マシン語はこの16進法で表記します。

ところで上の例はたまたま良かったのですが、このままでは表現できない数が出てきます。たとえば、1010 1110 はどう表現すればよいのでしょうか。

1010は10、1110は14というので、1014と表現したいところですが、これは0001 0000 0001 0100 のことになってしまいます。

そこで図1-1 を見て下さい。1010～1111にはA～Fの文字を当てています。

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

（図1－1）

これならさきほどの1010 1110 も AE とすっきり表現できます。

そして10進←→16進の換算は比較的簡単に行うことができます。

10進→16進の場合は割れなくなるまでどんどん16で割って行きます。

例）10進の1234を16進に換算します。

```
16)1234
16)  77   余り2
      4  余り13(D)
```

上の計算により10進の1234は16進では4D2 になります。

また2進数で表現すると0100 1101 0010になります。

逆に16進→10進の場合は各桁ごとに16のn-1乗 を掛けて最後に合計します。ここで n は桁位置を示します。

例）16進の4D2 を10進に換算します。

```
4   D(13)   2
                └▶ 2 ×  16の0乗 =     2
          ┌──▶ 13×  16の1乗 =  208
     ┌──────▶ 4  ×  16の2乗 = 1024 (+
                                1234
```

このように2進数よりも16進数のほうが扱いやすいので、マシン語のプログラムは16進数で表記しますが、これはあくまで表現上の約束事であって、コンピュータの内部ではすべて2進数で処理されています。

たとえば上の例で使った16進数 4D2は、MYCPU80回路のキーから入力するときは、そのまま 4 D 2と入れますし、LED表示器にもそのまま 4D2と表示されますが、CPUの内部には010011010010というように区切なしの2進数で入っていきます（もう少し正確に言うと、8080は8ビットのCPUで、2進数の8桁を一区切りとして扱っています。したがって上の 4D2は頭に0 をつけて、04D2にして、00000100 11010010 という形で実際は扱われています）。

2進数と16進数とは4桁毎に区切って表現するかしないかの違いがあるだけで、数そのもののもつ値は同じです。

というわけで、プログラムを組むときは普通は16進だけで扱っていればよいのですが、もともとCPU内部では2進で処理しているために、16進で考えているとよく理解できない命令にぶつかることがあります。

そのような時には、図1-1 を見ながら2進数に置き換えて考えてみて下さい。

●マシン語コード

CPU内部には区切なしの2進数で入る、と説明しましたが、正しくは8桁ごとにまとめて処理される、ということはすでに説明しました。

4桁だったり8桁だったりややこしい話が続きますが、かんじんのところですから、もうしばらく我慢して下さい。

さきほどの16進数で4桁毎に区切って表現したのは、そのほうが（人間が）理解し易いという理由からで、言わば便宜的な表記に過ぎません。

しかしこれから説明する、8桁毎の処理、というのはハード上の制約からで、とにかく回路がそうなっているのです。

MYCPU80回路の回路図を見て下さい。CPUとメモリやI／Oポートをつなぐ線が沢山引かれています。このうちD0～D7の8本のラインがCPUとメモリやI／Oが命令コードやデータをやりとりする線で、データバスといいます。

この線が8本なので、8080は一度に8桁のデータを読んだり、書いたりします。

この1か0で示される2進数の桁のことを、ビットといいます。

8桁ですから8ビットです。

8ビットのパソコンとか16ビットのパソコンとかいうのは、ここからきています。

そしてこの8ビットを1バイトとよぶこともあります。

さてそこで、命令コードに戻ります。そのように一度に8桁(8ビット）のデータを扱うように作られたCPUなので、命令コードも8ビットが単位になります。

言い換えれば、1バイトが命令の単位になります。

ここに、00111100という2進数があります。16進で表せば3Cです。（10進数と間違えないように、普通16進数は、3CH というように、最後にH をつけて表しますが、これが命令コードである場合には、ただ3Cと表すだけでH はつけません）

これは数値として考えれば、10進に換算して60という値になります。

一方これをCPUが命令コードとして受け取った場合には「Aレジスタの中身を＋1せよ」という命令になります。

ある16進数(2進数）をCPUが命令として受け取るか、数値として受け取るかは、プログラムをルール通りに書きさえすれば、はっきりと区別されるので、誤ることはありません。

●命令の長さ

このような命令、データは決められた順序で予めメモリの中に書いておきます(これがプログラムです)。

CPUはメモリから1バイトずつ命令コードを読んで実行して行きます。

上で説明した命令コードは8ビット(1バイト）でした。つまりこの場合CPUはメモリから1回命令コードを読むだけで、ただちに実行します。

しかし命令の中には、一度では読めなくて2～3回読んで初めて実行できるものもあります。

一度で読めてしまう命令は1バイト長だといいます。したがって一度で読めない命令は2バイト長、3バイト長、ということになります。

たとえば3Eというコードは「Aレジスタに、数値を入れよ」という命令ですが、これだけでは実行できません。 どういう数値を入れるかを指定してやらなければなりません。3Eに続けて25と書いておくと、Aレジスタに25H が入ります。これはメモリには、図1-2 のように続けて書き込みます。

アドレス	データ
8000	3E
8001	25

（図1－2）

アドレス	データ
8002	C3
8003	07
8004	80

（図1－3）

3Eが命令コードで25が数値になります。またC3と言うコードは「次に示すアドレス(メモリ番地) に無条件にジャンプせよ」という命令ですが、このコードに続いて、ジャンプ先のアドレスを指定してやらなければなりません。 これは図1-3 のように全部で 3バイトになります。はじめのC3が命令コードでそのあとの07、80が数値です

●マシン語プログラムの表記法

図1-2 と図1-3 の命令をメモリに書き込む作業を考えます。普通はいきなりメモリに書き込んだりしないで、まずノートなどに下書きしてから、書き込みます（これをコーディングといいます)。

この場合に図1-2 や図1-3 のように1バイトずつ縦に並べて書いてしまうと、あとで見たときどれが命令コードで、どれが数値だか分からなくなってしまいます。

そこで下のように書きます。

8000　3E 25
8002　C3 07 80
こうするといつも一番前に命令コードがきて、その後ろに数値がならぶので、理解し易くなります。
なお、命令コードのことを「OPコード」、数値のことを「オペランド」ともいいます。

●ニーモニック

慣れてくると、上のようにマシン語コードでいきなりコーディングすることもできるようになります。
しかしはじめのうちは、そんなに簡単にはできません。
それにマシン語だけでは、あとから見た時、どんな命令だったのか分からなくなってしまいます。（それこそ暗号表を見ているようなものです）
じつは命令コード全てに、理解し易いように、英語名（省略形）がつけられているのです。この英語名のことをニーモニックといいます。
先程の例をこのニーモニックで書いてみると、下のようになります。
MVI A, 25
JMP ＄8007
MVIはMove Immediateの略で、JMPはJumpの略です。
8007の前に＄がついているのは、当社オリジナルの8080アセンブラのルールです（一般的なルールではありません）。
なんだい、少しも分かり易くないじゃないか、と思われたかもしれません。
でも少し慣れてくると、ニーモニックでプログラムを書いたり、読んだりすることが楽にできるようになります。
なにしろマシン語コードの場合には、00から始まってFFまで、256 個もあって、そのうち8080で命令として使われているコードだけでも100個以上あるのですから全部覚えるのは不可能です。
ニーモニックも結構沢山ありますが、同じ性質のものには同じニーモニックがつけてあるので、マシン語コードよりはずいぶんまとまりやすくなります。
たとえば、MOV A,B、MOV A,C、MOV A,D、MOV A,E、MOV A,H、MOV A,L、MOV A,M など全部MOVですが、マシン語コードはひとつひとつ異なっています。
それによく使う命令は数が限られているので、それさえ覚えてしまえば、あとはうんと楽になります。（参考までに代表的なものを下にあげておきます）
MOV、MVI 、ADD 、SUB 、ANA 、ORA、XRA 、CMP、INR 、DCR 、INX、DCX、PUSH、POP 、JMP、CALL、RET 、IN、OUT 、NOP
できるだけニーモニックに慣れるようにして下さい。

以上で、最低限必要と思われる事柄についての説明は終りです。

2章　基本操作

電源を入れる前に、MYCPU組立説明書43ページ～45ページ（13－14. ～13－17. ）の準備が完了していることを確認してください。

1.　LED表示

まず電源を入れて下さい。
図 2-1 のようにLEDには8個のゼロが表示されます。

（図2－1）

●アドレス表示、データ表示
この8桁のLEDは、キーボードからの入力データや、メモリ内容の表示などに使われます。
上位4桁はメモリなどのアドレスを主に表示します。アドレス表示部です。
下位4桁はデータ表示部です。キーボードからの入力データは、まずこのデータ表示部に表示されます。（図 2-2）

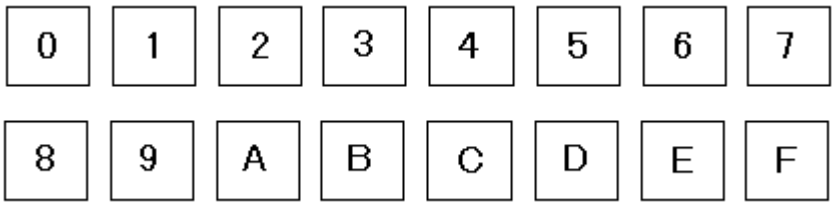
（図2－2）

アドレスは16進4桁ですから、4桁のアドレス表示部にそのまま表示されますが、データは16進2桁なので、メモリ内容の表示などの場合には、データ部の下2桁に表示が行われます（TK80回路のCPUは8ビットなので、データは16進2桁になる）。
その場合のデータ部の上2桁には、この表示前のデータ部の下2桁の表示がCOPYされるだけですから、普通は無視しても構いません。(レジスタ内容の表示やその他特別の場合には、データ表示部も4桁全部を使うことがあります）

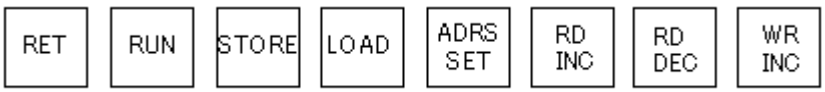
2.　キーボード

モニタプログラムに色々な指示を与えたり、メモリの中身を読んだり書いたりする場合に、それらの作業は全てキーボードからの入力によって行われます。
TK80回路のキーは、5×5配列のキー25個です。
この25個のキーは、その働きによって、次の3つのグループに分けられます。

①データキー



②ファンクションキー



③リセットキー

MYCPU基板のわずかな空きスペースを利用してパーツを配置したため、５×５キーも小型のタクトスイッチを使うしかありませんでした。
そのためキートップに文字が入れられません。
各キーの下側位置に小さな文字でマーキングしてあります。
キーとキーとの間も狭いため、ちょっと見にくいのですが、がまんしてください。

２．１　データキー

メモリアドレスを指定して、データを書き込んだり、プログラムを入力するときの、１６進数のキーです。

２．２　ファンクションキー

メモリにプログラムを書き込んだり、そのプログラムを実行させたりするのに、都合のよい機能がモニタプログラムに入れてあります。
このキーはそのうちの最も基本的な動作をさせるためのものです。
具体的な使い方については、「３．　プログラムの入力」の項で例をあげて説明します。ここでは各キーの役割を一通り簡単に説明します。

２．２．１　ADRSSET（ADDRESS　SET、アドレス　セット）

データキーを押してキーボードから１６進数を入力すると、その数はLEDのデータ表示部に表示されます。
ADRS　SETキーを押すと、LEDのデータ表示部にあった４桁の１６進数がアドレス表示部に移って表示されます。そしてデータ表示部には、そのアドレスのメモリの中身が表示されます。
メモリアドレスを指定するときに使うキーです。

２．２．２　RDINC（RDREAD　INCRIMENT、リード　インクリメント）

RDINCキーを押すと、LEDのアドレス表示部に表示されているアドレスが＋１進められて表示され、データ表示部にはその新しいメモリアドレスの中身が表示されます。
アドレス表示部に表示されているアドレスから順に（アドレスを＋１しながら）、データを読み出したいときに使います。

２．２．３　RDDEC（READ　DECRIMENT、リード　デクリメント）

上のRDINCと動作はよく似ていますが、アドレスが＋１されるのではなく、－１されます。
アドレス表示部に表示されているアドレスから順に（アドレスを－１しながら）、データを読み出したいときに使います。

２．２．４　WRINC（WRITE　INCRIMENT、ライト　インクリメント）

WRINCキーを押すと、LEDのアドレス表示部に表示されているメモリアドレスにデータ表示部の下２桁の内容が書き込まれます。そして書き込み後、アドレス表示部のアドレスは＋１進められて表示され、データ表示部にはその新しいメモリアドレスの中味が表示されます。
メモリアドレスにデータや命令コードを書き込みたいときに使います。

２．２．５　RUN（ラン）

RUNキーを押すと、LEDのアドレス表示部に表示されているメモリアドレスに書かれているプログラムが実行されます。
もう少し正確に表現すると、CPUはそのアドレスにプログラムが書いてあるものとして実行します（たとえでたらめのデータが並んでいても、命令コードと判断して実行してしまいます。その結果は勿論でたらめの動作になるのですが）。
このキーはプログラムを実行するときに使います。

２．２．６　RET（RETURN、リターン）

ブレイク動作で中断されたプログラムの実行を再開したいときに使います（ブレイクについては３章で説明します）。
このキーを押すと、強制的に一時停止させられていたプログラムが、その続きから再び実行されます。
モニタプログラムからユーザープログラムに戻るので「RETURN」です。

２．３　RESET（リセット）

このキーは、実行中のプログラムを強制的に打ち切るときに使います。
なお電源を入れた直後は、リセットキーが押されたのと同じ状態からスタートします。
このキーを押すとCPUは何を実行していても、あるいはどういう状態であっても、モニタプログラムの先頭（0000H番地）に戻って再スタートします。
LED表示はオール0になって、モニタプログラムのワークエリアはクリアされますが、ユーザープログラムは消えないで残ります。
いま実行中のプログラムを打ち切りたいときなどに使います。
プログラムミスなどによって、CPUが暴走してもとに戻らないときにも使います。

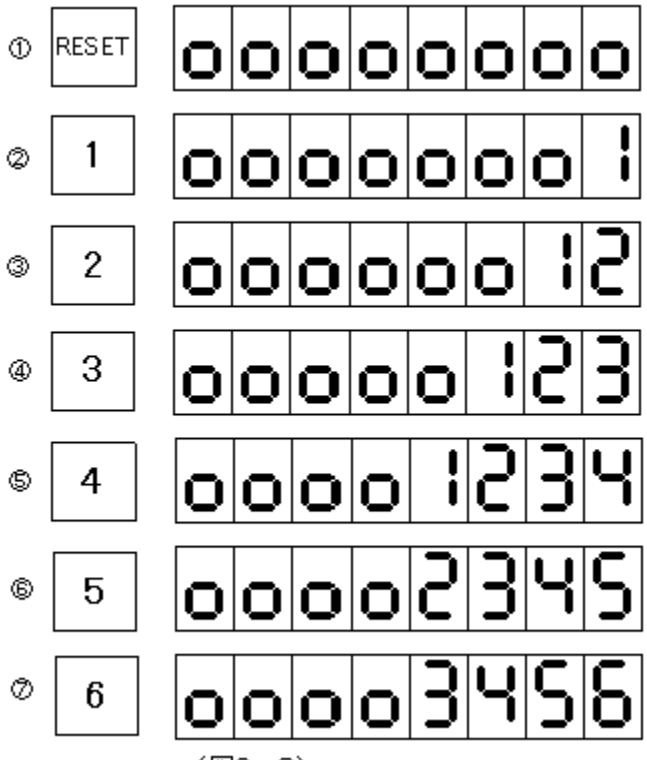
3. プログラムの入力

3．1 アドレス

プログラムはRAMに1バイトずつ書き込んで行きます。
RAM（ラム）は RANDOM ACCESS MEMORY の略称です。どこのアドレス（番地）からでも自由に読んだり書いたりできるメモリで、TK80回路ではでIC261の62256がそのRAMです。
TK80回路ではRAMのアドレスは8000～FFFFの32KB（キロ・バイト、1KB＝1024バイト）になっています。
このうちFFC8～FFFFはモニタプログラムのためのワークエリアなのでユーザーが使うことはできませんが残りの部分はどこにプログラムを書いても構いません。
しかしできれば8000から書きはじめるようにして下さい。
サブルーチンやPUSH命令を使うと、FFC7から前のRAMエリアがユーザープログラムのためのスタックとして使用され、若いアドレスに向かって消費されていきます。
ですから余りFFC7に近いアドレスにサブルーチンやPUSH命令を含むプログラムを書くとプログラムがスタックによって破壊されてしまいますから注意してください。

3．2 キー入力とLED表示

まず図のように順番にキーを押していって下さい。



（図2－3）

①必ずしも始めにリセットしなくてもよいのですが、慣れるまでのあいだは、このようにリセットしてから始めた方が確実です。
②キー入力されたデータ（16進数）はLEDの一番右に表示されます。
③④続いて入力していくと、先に表示されていたデータはキー入力の度に左に送られ、つねに入力データは右端に表示されます。

⑤このようにしてどんどん入力していくと、LEDのデータ表示部（下4桁）がいっぱいになってしまいます。
⑥⑦さらに続けて入力すると、先にデータ表示部の一番左に表示されていたデータはその右には行かないで、消えてしまいます。
　このように、データ入力はつねに今LEDのデータ表示部に表示されている、最後の4回分が有効で、それ以前の入力は無視されてしまいます。例えば図 2-3 ⑦ではあとから入力した3456が有効で、さきに入力した1と2は無視されます。
このキー入力とLED表示の関係をまず覚えておいて下さい。

3．3　サンプルプログラムの入力

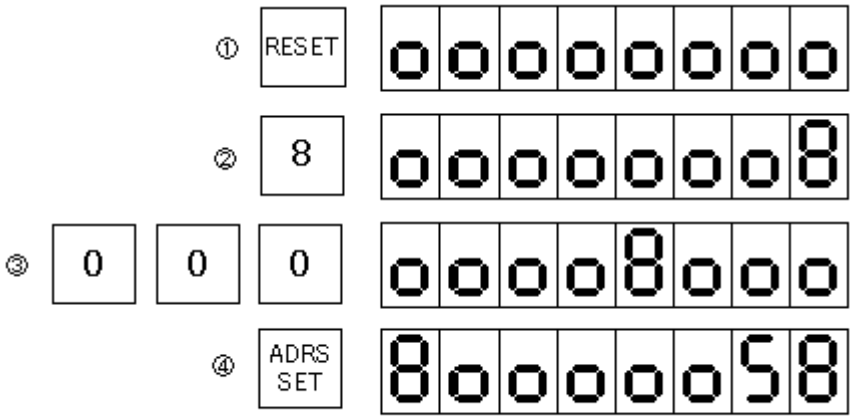
　アドレス8000から次のプログラムを入力してみます。

```
8000  3E00        MVI  A, 00
8002  3C          INR  A
8003  C30280      JMP  $8002
```
（リスト2－1）

　このプログラムははじめにAレジスタをゼロクリアしたあと、そのAレジスタの中味を＋1加算することを繰り返すものです。
　なおレジスタについては後程説明しますので、今はプログラムの入力方法と実行の仕方を覚えるため、以下の説明に従ってキー入力して下さい。

●アドレス（8000）をセットします（図2－4）

①RESETキーを押します。
②データキーの8を押して下さい。LEDの右端に8が表示されます。
③続いて0を3回押して下さい。LEDの下4桁に8000と表示されます。
　もし入れ間違えたら、気にしないでもう一度8から入れ直して下さい。（このときリセットする必要はありません）
　とにかくLEDの下4桁に8000が表示されるようにします。
④つぎにADRSSETキーを押します。するといままで下4桁（データ表示部）にあった8000が、上4桁（アドレス表示部）に移動します。
　そしてこのとき、データ表示部の下2桁が、8000番地のメモリの内容を表示しています。
　ここでは58が表示されていますが、これは例であってこのときどういう数が表示されるかは状況によります。
　TK80回路のRAMは電池でバックアップされているため、通常は前に書かれていたデータがそのまま読み出されます。



（図2－4）

●データ（プログラム）を入れます（図2－5）

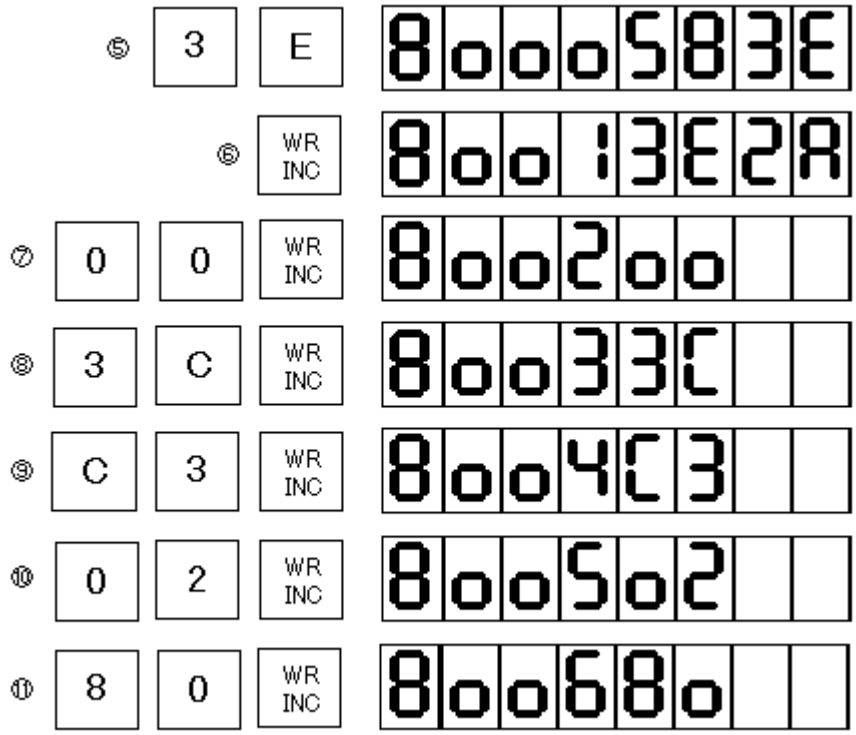
⑤　3　E　と押します。このとき入れ間違いをしたら、気にしないでもう一度　3　Eと入れ直します。データ入力の場合はアドレス入力とは違って、データ表示部の下2桁が有効になります。
⑥次にWRINCキーを押します。
　すると今入力したデータ3Eが左に移動してアドレスが＋1され、データ表示部の下2桁には新しいアドレス（8001番地）の内容が表示されます。

この図では2Aが表示されていますがこれも何が表示されても気にしないで下さい（⑦～⑪ではデータ部の下2桁を空白にしてあります。ここは何が表示されていても気にしないでください）。

以上のようにメモリにデータを書き込むにはデータ表示部の下2桁に、書き込みたいデータを表示させたあとWRINCキーを押します。

データが書き込まれると同時にアドレスが＋1されますから、つぎつきにデータを書き込んで行くことができます。

このようにしてこのあと8005番地までデータ（プログラム）を入力します。⑦～⑪



（図2－5）

●これであとは実行させれば良いのですが、念のため正しくメモリに書き込まれているか、チェックしてみます。（図 2-6）

①8000と入力します。

②続いてADRSSETキーを押すと、データ表示部の下2桁に8000番地の内容が表示されます。

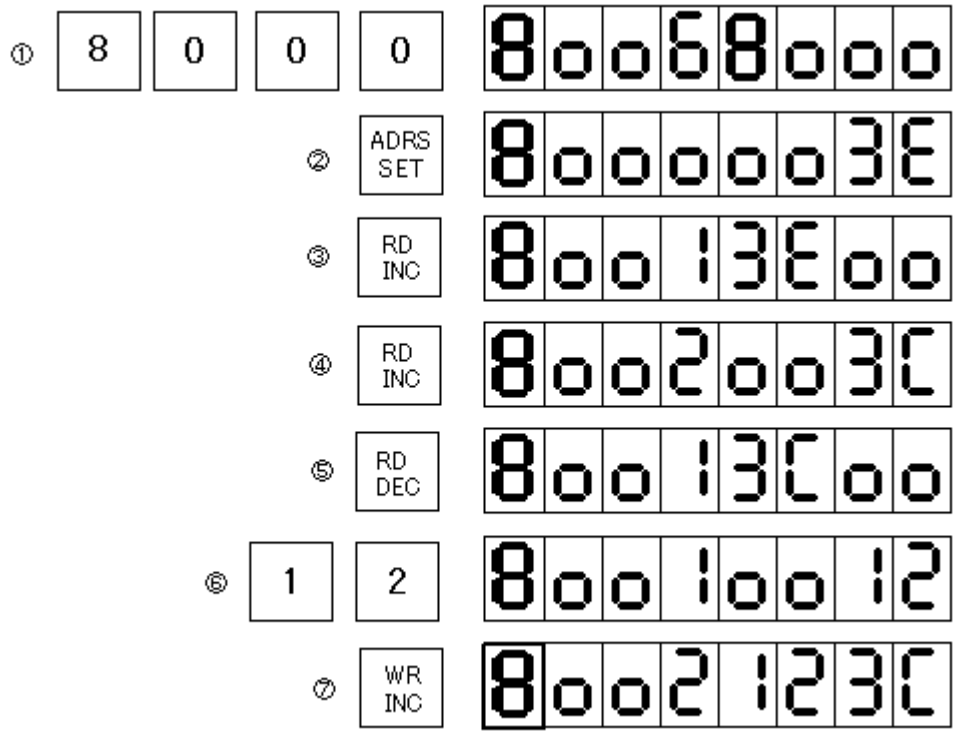
③ここでRDINCキーを押すと、アドレスの表示が8001になって、データ表示部の下2桁には8001番地の内容が表示されます。

④もう一度RDINCキーを押すとアドレス8002とそのメモリ内容が表示されます。

このようにRDINCキーは押す度にアドレスが＋1されて、順にそのメモリ内容を見ていくことができます。

⑤これに対してRDDECキーを押すと、アドレスが逆に－1されていきます。

⑥⑦チェックしているときにミスを発見したら、正しいデータをその場で入れ直してWRINCキーを押せば新しい内容に書き換えることができます（8001は00でよいのですがここでは練習のために12に書き換えています）。

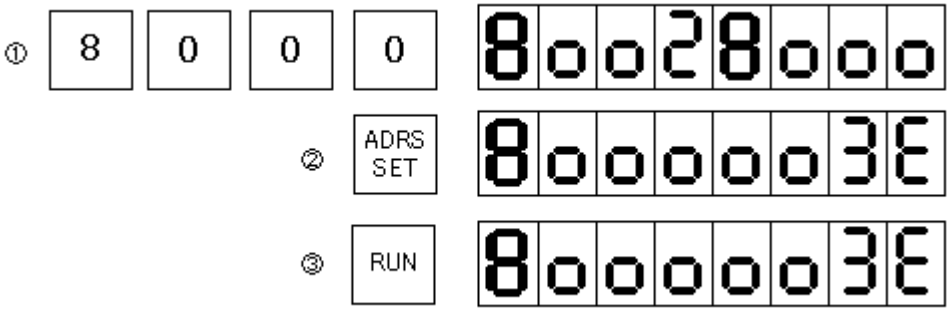


（図2－6）

3．4 プログラムの実行

プログラムを入力したときと同じようにしてプログラムの開始番地（この例では8000番地）をセットします（図 2-7①②）。

これで準備完了です。このあとRUNキーを押せば8000番地から書かれているプログラムが実行されます（図 2-7③）。



（図2－7）

しかしCPUはこのプログラムを非常に速いスピードで実行しているので、このままではどうなっているのか確かめることはできません。

TK80回路には、そのような場合にCPUの動作が確認できる便利な機能が備わっています。

その機能については次章で説明します。

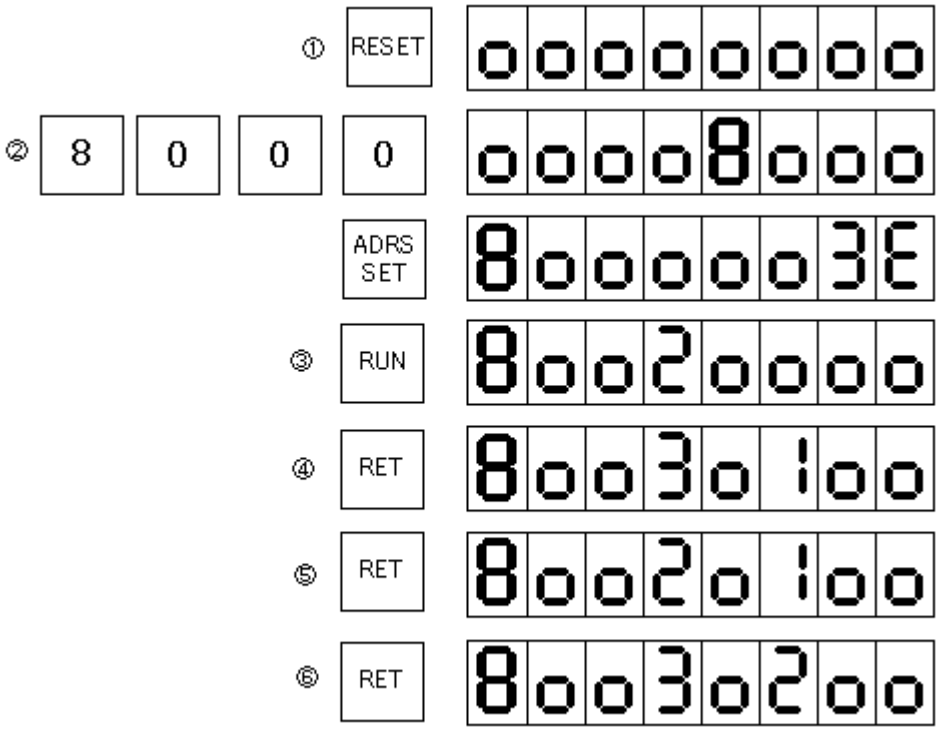
3章　プログラムデバッグの仕方

1.　はじめに

デバッグ(Debug) は虫取りと訳したりします。
自分で作ったプログラムは、考え方の間違いや色々不注意によるミスのため、中々一度では期待通りに動いてはくれないものです。
このようなミスをバグ(Bug, 虫) といいます。
毛の奥深くに、もぐりこんでいる虫を一匹ずつ、文字通り「シラミツブシ」にみつけ出す作業は、根気のいる仕事です。
特にマシン語のプログラムは、なにか手助けになるようなソフトがなければ、まずお手上げです。
このTK80回路のもっているステップ動作機能とブレイク動作機能は、そんなとき強力な助けになります。

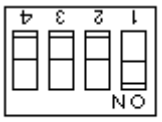
2.　ステップ動作

2章の終りのところで、CPUは非常に速いスピードでプログラムを実行するので確認できない、と書きました。
このステップ動作の機能を使うと、プログラムを1ステップずつ進めることができるので、その途中の状態を確認することができます。
2章で作ったプログラムを実行させてみます。念の為リスト 2-1 の通りにメモリに入っていることを確認しておいて下さい。
（2章の終わりのところで8001番地を12に直した人は、00に戻して下さい）
以下に操作方法を説明します。（図 3-1 参照）



（図３－１）

①まずリセットします。（必ずリセットして下さい）
②普通のプログラム実行のときと同じように、アドレスをセットします。
このあと（①のあとでもよい）RUNを押す前にディップスイッチDS3－1（TK80STEP）をON（下側）にします。



DS3

③RUNキーを押すと、CPUはあっという間に、8000番地の命令を実行して、次のステップのアドレス（8002番地）を表示して止まります。
④次からはRETキーを押します。するとさらに次のアドレス（8003）が表示されます。
⑤もう一度RETを押すとアドレスは8002に戻ります。

⑥このあとはRETを押す度に8002と8003が交互に表示されます。
　またこのときデータ表示部の上2桁はアドレスが8003になる度に、00、01、02と＋1ずつ増えて行きます。
　じつはこのときデータ表示部の上2桁にはAレジスタの内容が表示されているのです。（下2桁にはフラグ（F）レジスタの内容が表示される）

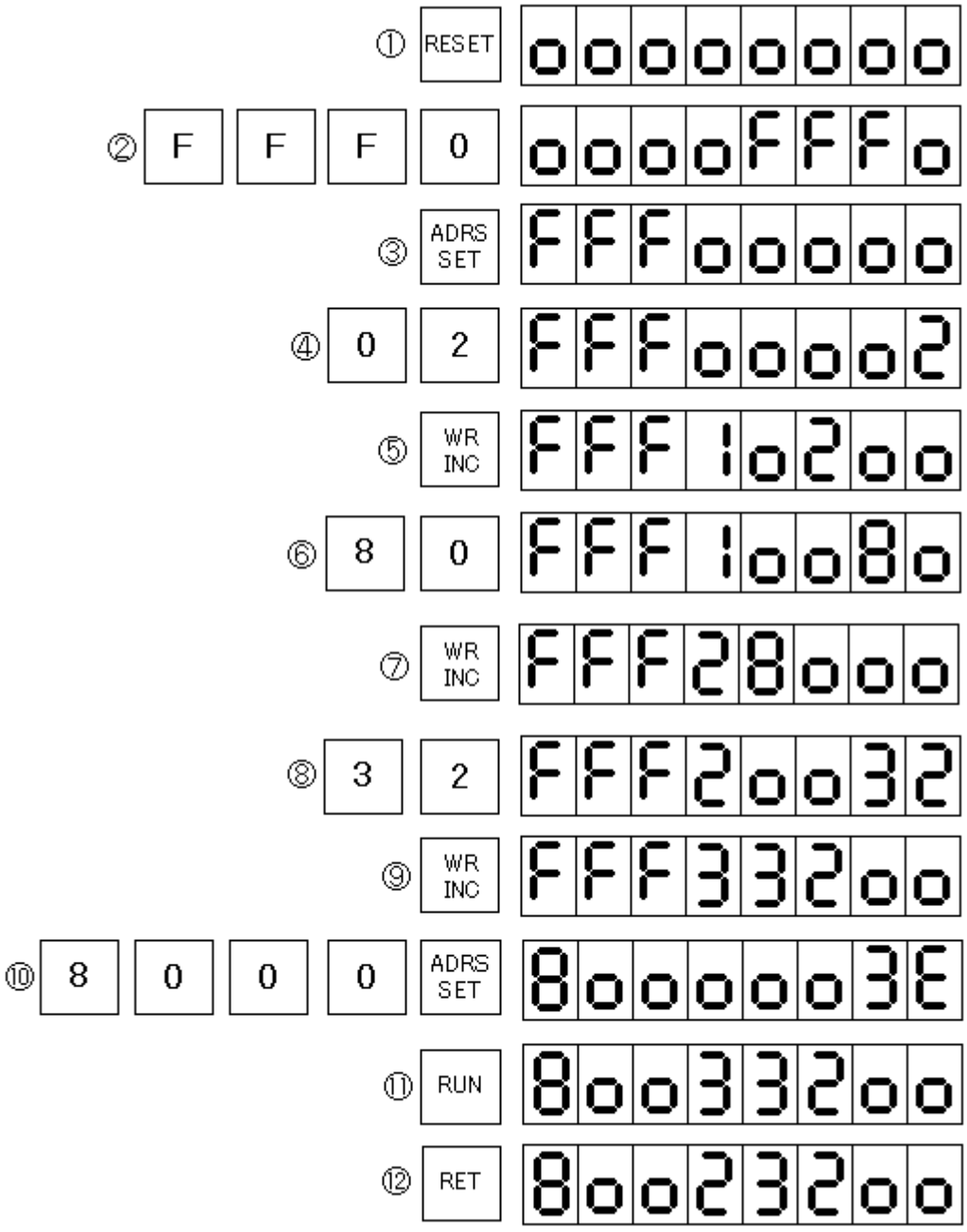
〔注意1〕ステップ動作は、モニタROM内のプログラムにも働きますが、ステップ動作そのものがモニタROM内のプログラムや機能を使っているために、期待した通りには動作してくれません。
〔注意2〕ステップ動作は割り込み（RST7）を利用しています。したがってもしユーザープログラムの中で、割り込み禁止命令（DI）を使うと、それ以後は割り込みが禁止されるため、ステップ動作ができなくなります。

3.　ブレイク動作

　上で説明したステップ動作は、作ったプログラムの動きをチェックするのにとても便利な機能ですが、さらに、あるところまでは普通に実行しておいて、指定したアドレスからはステップ動作になると便利な場合があります。これをブレイク動作と言います。
　このTK80回路のモニタはブレイクするアドレスを記憶するブレイクアドレスレジスタと、ブレイクするまでの繰り返し回数を記憶するブレイクカウンタをもっていますから、指定したアドレスでいきなりブレイクするのではなく、そのアドレスを指定回数繰り返し通過したのちにブレイクするという、きめ細かい処理が可能です。
　今回もステップ動作と同じく、2章で作ったプログラムをブレイクさせてみます。
　8002番地を50回実行したあとステップ動作に移る（ ブレイクする）ようにセットします。
　図 3-2 を参照しながら、以下の説明を読んで下さい。

①まずリセットします。（ブレイクの時は必ずしもリセットから始めなくてもよいのですが、この方が確実です）
②③ブレイクアドレスをセットします。
　［F］［F］［F］［0］［ADRSSET］の順にキーを押してください。
　アドレス表示部にFFF0と表示され、データ表示部は0000になります。
④FFF0にはプレイクアドレスの下位2桁が入るのですがリセット後は0000になっています。
　ここにブレイクしたいアドレスの下位2桁を入れます。今回は8002をセットしますから、その下位2桁の02を書き込みます。
　［0］［2］の順にキーを押してください。
⑤［WRINC］キーを押します。
　FFF0に02が書き込まれ、アドレス表示部にはFFF1が表示されます。
⑥FFF1にはプレイクアドレスの上位2桁が入るのですがリセット後は0000になっています。
　ここにブレイクしたいアドレスの上位2桁を入れます。今回は8002をセットしますから、その上位2桁の80を書き込みます。
　［8］［0］の順にキーを押してください。
⑦［WRINC］キーを押します。
　FFF1に80が書き込まれ、アドレス表示部にはFFF2が表示されます。
⑧FFF2にはプレイクするまでの繰り返し回数を入れます。
　FFF2はダウンカウンタの役目をしていて、ブレイクアドレスを実行するたびにダウンカウントされます。
　リセット後は0000になっています。
　今回は繰り返し回数を50回にします。10進の50は16進では32になります。
　そこで32と入力します。
　［3］［2］の順にキーを押してください。
⑨［WRINC］キーを押します。
　FFF2に32が書き込まれます。
　これでブレイクカウンタのセットができました。
［注記］ブレイクカウンタは16進2桁です。01～FFつまり1回から255回の繰り返し回数をセットすることができます。

⑩プログラムの開始アドレスをセットして下さい。8000です。
⑪ディップスイッチDS3－1（TK80STEP）がON（下側）になっているか確認して下さい。（OFFのままですと、ブレイクできません）
　RUNキーを押すと瞬間にプログラムが50回実行されて、ブレイクします。50回実行した証拠に、データ表示部の上2桁にはAレジスタの値、32（十進の50）が表示されています。
　これ以後はステップ操作と同じです。RETキーを押すと1ステップずつ進みます。
　なおブレイクカウンタはブレイク時点で0になります。ブレイクアドレスはRESETキーを押さない限りクリアされないでそのまま残ります。



(図３－２)

〔注意3〕ブレイクカウンタが0になっている時は、ブレイク動作ではなくてステップ動作になります。
〔注意4〕ブレイクアドレスは各命令の1バイト目でなければいけません。今回の例では8000、8002、8003は指定できるが、8001、8004、8005を指定してはいけません。
〔注意5〕ブレイクアドレスを設定した場合、ブレイクするまでの間のプログラム実行時間は、通常処理の場合の数十倍かかります。これはブレイクアドレス以外のプログラム部分でも1ステップ実行する毎に、ブレイク処理プログラムが実行されているためです。
〔注意6〕ブレイク動作もステップ動作の機能を利用しています。そのためステップ動作についての注意事項（注意1、注意2）はブレイク動作の場合でも同じように当てはまります。

4. レジスタの確認

以上ステップ動作とブレイク動作の基本的な操作について説明してきましたが、じつは両動作をさらに効果的にする機能が備わっています。
いままで説明したブレイク動作、ステップ動作では、ブレイクしたときの、またはステップごとのプログラムカウンタの値とAレジスタおよびフラグの状態を確認することができました。
じつはAレジスタだけではなくてそのほかのレジスタ、B、C、D、E、H、Lの各レジスタの値とスタックポインタの値についても確認することができるようになっているのです。

確認するだけではなくて、Aレジスタを含め、各レジスタの値を強制的に変更してから、ブレイクしたあるいはステップ動作中のプログラムの続きを実行させることもできます。
プログラムの続きからではなくても、レジスタに値を設定してからプログラムをスタートさせることもできます。
具体的な操作方法について説明する前に、まずMYCPU80（8080）のレジスタについて簡単に説明しておきます。

4．1 MYCPU80（8080）のレジスタ

MYCPU80（8080）は内部に下記のレジスタを持っていて、これらのレジスタはプログラムの中で色々な処理に利用されます（図3－3）。

← 8ビット→	← 8ビット→
A	F
B	C
D	E
H	L

← 16ビット →
SP
PC

（図3－3）

[A]
一般にアキュムレータ（加算器）と呼ばれているように、演算命令はこのレジスタを中心に行われる。
[F]
フラグレジスタ。命令の実行により現れる色々な状態を1ビットずつに記録して保持する。各ビットの意味は4．2で説明する。
[B][C]
共に 8 ビットのレジスタとして、独立して使うことが多いが、つないで 16 ビットのレジスタ[BC]として使うこともできる。その場合には[B]が上位8ビット、[C]が下位8ビットになる。
[D][E]
B、Cと同じ。
[H][L]
B、Cと同じだが、16ビットレジスタ[HL]はメモリ[M]を間接的に示す間接アドレッシングでのメモリアドレスを入れて使うことが多い。また[HL]は16ビットの加算命令（DAD）で加算器（アキュムレータ）としても使われる。
[SP]
スタックポインタ。現在のスタックのトップ・アドレスを示している。スタックについては、5．3で説明する。
[PC]
プログラムカウンタ。現在実行中のアドレスを管理している。（ 正しくは、次に実行する予定のアドレスを示している）

4．2 MYCPU80（8080）のフラグ

フラグは8ビットのフラグレジスタに、図3－4のように割りつけられています。

S	Z		H		P		C
7	6	5	4	3	2	1	0

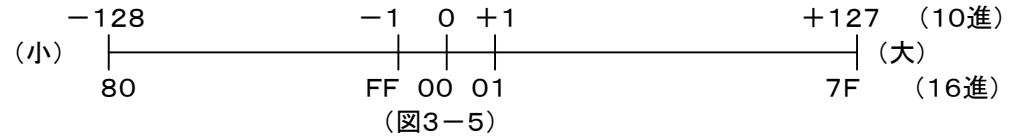
（ビット位置）

（図3－4）

各記号の意味は下の通りです。（ビット1、3、5は使用されない）
なお、フラグがセットされたときは、そのビットが1になり、リセットされたときは0になります。
C
キャリ・フラグ。計算の結果、上位桁へのキャリー、ボローが発生したときにセットされる。ローテイト命令でもセット、リセットされる。
P
パリティフラグ。論理、算術演算およびINR、DCR命令を実行した結果、1のビットが偶数個あるときにセットされ、奇数個のときはリセットされる。
H
ハーフ・キャリ・フラグ。算術演算でビット3からビット4へのキャリーや、ビット4からビット3へのボローがあったときセットされる。このフラグはCフラグとともに、BCD演算後のDAA命令で利用される。

Z
　ゼロ・フラグ。結果がゼロのときセットされる。
S
　サイン・フラグ。結果が負のときセット、正またはゼロのときリセットされる。

［参考］2進数の正数と負数
　8ビットの数00～FFは符号なしでは10進の0～255として扱われるが、符号付の数として扱ったときには、－128～＋127の数になり、これは16進では80～7Fになる。（ビット7が0のときはその数は正で、ビット7が1のときは負になる）
●符号付8ビットの数の大小



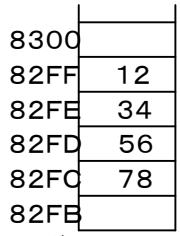
（図3－5）

4．3　スタック

　大きなプログラムになると、レジスタもたくさん必要で、とても4．1で説明した数では足りません。そこでレジスタの値をひとまずメモリのワークエリアにしまっておいて、そのレジスタを次の用途に使う、ということが簡単にできると便利になります。
　ところがレジスタの値をしまうときに、一々異なるメモリアドレスに割りつけていくのでは大変です。
　そんなときにこのスタックを使えば、一々メモリアドレスを指定しなくても簡単な操作でレジスタの値を保存することができます。
　スタックとは積み重ねるという意味です。
　ちょうど本などを積み重ねるように、メモリの中にレジスタの値を順番にしまうことができます。（PUSH命令を使います）
　取り出すときは、入れたときと逆の順番で取り出します。（POP命令を使います）
　そしてそのスタックの現在の位置を管理しているのがSP（スタックポインタ）です。
（例）SP＝8300、BC＝1234、DE＝5678のとき、

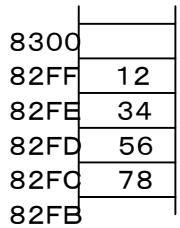
```
PUSH  B
PUSH  D
```

を実行すると、メモリ内容は下のようになります。



←命令実行前のSPの位置
←命令実行後のSPの位置（SP＝82FC。BC、DEは変化しない）

この後で POP H を実行すると、下のようになります。



←命令実行後のSPの位置（SP＝82FE。HL＝5678になる。BC、DEは変化しない）

（図3－6）

　こうなってしまった後で、POP　Dを実行してもDにはもとの値は戻りません（1234が入る）。
　PUSH、POPは常に順番を覚えておいて、間違わないように使う必要があります。
〔注意7〕スタックの操作はPUSH、POPだけではなくCALL、RET命令や割り込み処理でも使用されます（アドレスがスタックに入れられる）。
〔注意8〕スタックはメモリ上のどこにでも設定することができます（LXI　SP命令を使う）。
　しかし指定場所によってはプログラムやデータの入っている領域と重なってしまい、その結果プログラムやデータが壊されてしまうことがあるので、充分注意が必要です。
　マシン語プログラムでは、普通はその先頭部分でスタックポインタのセットが必要ですが、TK80回路はモニタプログラムによってリセット後はSP＝FFC7にセットされるのでユーザーがあらためてスタックポインタをセットする必要はありません。

4．4　ブレイク、ステップ操作でのレジスタの値の設定、確認方法

［ブレイク、ステップ動作時に各レジスタが格納されるメモリアドレス］

ブレイクしたとき、またはステップ動作時には、各レジスタは下記メモリアドレスに格納されます。
［RET］または［RUN］を押すと、下のメモリアドレスの値がそれぞれレジスタに入れられたあとでユーザープログラムにジャンプします。
［RET］キーを押したときにはFFE0、FFE1の値がPCに入りますが、［RUN］を押したときには、LEDのアドレス表示部に表示されている値がPCに入り、SPにはFFC7が入れられます。

FFEB	CPUレジスタセーブエリア　A
FFEA	CPUレジスタセーブエリア　F
FFE9	CPUレジスタセーブエリア　B
FFE8	CPUレジスタセーブエリア　C
FFE7	CPUレジスタセーブエリア　D
FFE6	CPUレジスタセーブエリア　E
FFE5	CPUレジスタセーブエリア　H
FFE4	CPUレジスタセーブエリア　L
FFE3	CPUレジスタセーブエリア　SP（H）
FFE2	CPUレジスタセーブエリア　SP（L）
FFE1	CPUレジスタセーブエリア　PC（H）
FFE0	CPUレジスタセーブエリア　PC（L）

（図3－7）

上のメモリアドレスはブレイク後やステップ動作のとき以外でも、その中身を確認したり、書き換えたりすることができます。

ここでは参考までに、HLに1234を、BにEFを書き込んでみます。
Hレジスタに12を、Lレジスタに34を書き込みますが、WRINCは書き込み後にアドレスがインクリメント（＋1）されることを考慮して、さきにLレジスタ（アドレスFFE4）から書くのが効率的です。

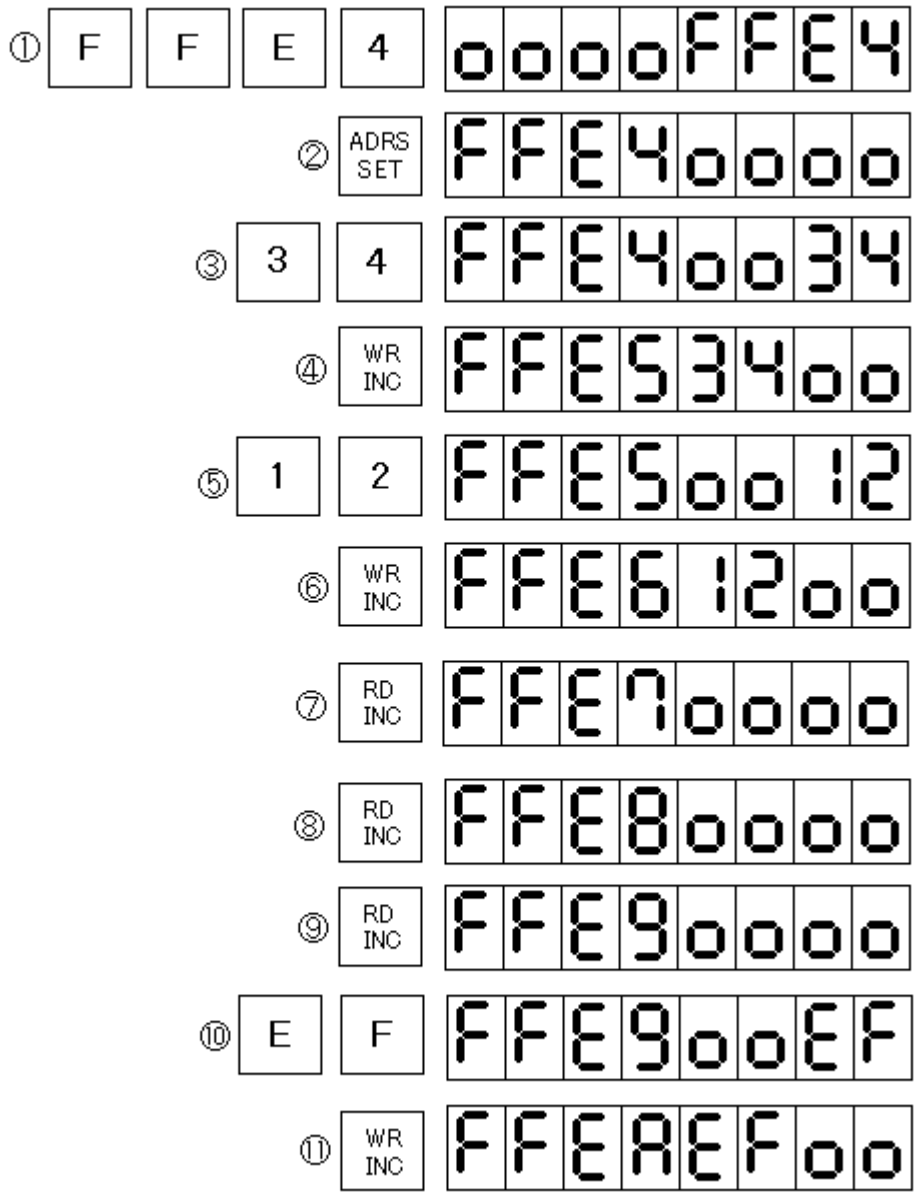
①②Lレジスタのセーブアドレスをセットします。
［F］［F］［E］［4］［ADRSSET］の順にキーを押してください。
アドレス表示部にFFE4と表示されます。ブレイク後やステップ動作のときはプログラムの実行によりそのときのLレジスタの値がデータ表示部の下位2桁に表示されます（リセットしてもFFE4の値はクリアされずに残ります）。
③このデータを書きかえることによって、Lレジスタに任意の値を与えてからユーザープログラムに戻るようにすることができます。
今回は操作例として34をLレジスタに与えることにします。
［3］［4］の順にキーを押してください。
④［WRINC］キーを押します。
FFE4に34が書き込まれ、アドレス表示部にはFFE5が表示されます。
⑤FFE5はHレジスタのセーブアドレスです。ブレイク後やステップ動作のときはプログラムの実行によりそのときのHレジスタの値がデータ表示部の下位2桁に表示されます（リセットしてもFFE5の値はクリアされずに残ります）。
Hレジスタの値を12にしてみます。
［1］［2］の順にキーを押してください。
⑥［WRINC］キーを押します。
FFE5に12が書き込まれ、アドレス表示部にはFFE6が表示されます。
⑦⑧⑨アドレスがFFE9になるまで［RDINC］キーを押します。
またはここで［F］［F］［E］［9］［ADRSSET］と操作することもできます。
⑩FFE9はBレジスタのセーブアドレスです。ブレイク後やステップ動作のときはプログラムの実行によりそのときのHレジスタの値がデータ表示部の下位2桁に表示されます（リセットしてもFFE9の値はクリアされずに残ります）。
Bレジスタの値をEFにしてみます。
［E］［F］の順にキーを押してください。
⑪［WRINC］キーを押します。
FFE9にEFが書き込まれます。

書き込んだ値を確認するには[RDDEC]キーを押して戻るか、[ADRSSET]キーを使って確認したいアドレスを指定します。必要ならば[RDINC]キーを使うこともできます。

上記例のような作業がブレイク後またはステップ後に行われたときは、このあとでRETキーを押すことによって、上の作業で最終的に書き換えられたデータを各レジスタにセットしたあと、ユーザープログラムに復帰します。

またブレイク動作やトレース動作以外の普通の処理でも、上記例のような操作で必要な値をセットしたあと、RUNすることにより、CPUレジスタに特定の初期値を持たせてユーザープログラムを開始させることができます。

実行途中のCPUレジスタの値を参照できるばかりではなく、必要ならば途中で各レジスタの値を変更することもできるため、非常にきめ細かなデバッグ作業が行えます。



（図３－８）

[注意9]プログラムカウンタ（PC）やスタックポインタ（SP）の値を不用意に変更すると、以後のユーザープログラムがまともに実行されなくなることがあります。

[注意10]上の操作例では、FFE4～FFEAのアドレスが指定されたときに、そのメモリの値がすべて00で表示されていますが、実際には、ここにはこのとき以前にブレイク操作かステップ操作を行ったときに入れられた各レジスタの値が表示されます。レジスタセーブエリア（FFE0～FFEB）はリセットしてもクリアされません。

5. プログラムの終わり方

いままでのところで説明に使ったサンプルプログラムは、終わりのないプログラム（これを無限ループといいます）でした。

しかし普通は、何かの処理をしたあとは、そこでストップするというプログラムが多いはずです。

マシン語のプログラムは必ず終わりをしめくくっておかなければいけません。（やりっぱなしにすると、暴走してしまいます）

終わり方には、次のような方法があります。プログラムを書く場合の参考にして下さい。

①HLT命令（コードは76）

最後にHLT（76）を書いておくと、そこで停止したままになります。このときZ80Aの18番ピン（HALT）はLレベルになります）

この状態から、通常のキー操作に戻るためにはリセットをする必要があります。

②RST 0命令（コードはC7）

最後にRST 0（C7）を書いておくと、モニタの0000番地に戻ります。つまりリセットが最後にかかったのと同じ状態になります。LED表示はオール0になり、システムワークエリアはクリアされます。

③RST 1命令（コードはCF）

最後にRST 1（CF）を書いておくと、モニタの0008番地に戻ります。

0008番地にはモニタのリスタートアドレス（0051）へのジャンプ命令が書いてあります。モニタが0051からリスタートするとLEDの表示はクリアされませんが、スタックポインタなどは初期セットされます。

④RST 7命令（コードはFF）

最後にRST 7（FF）を書いておくと、その次のアドレスをLEDのアドレス表示部に表示してブレイクします。このときレジスタの内容はレジスタセーブエリアに保存されますから、処理終了時点のレジスタの値を確認することができます。

なお、プログラムミスなどでCPUが暴走した結果、ROMの何も書かれていないアドレスやRAMのたまたまFFが書かれているアドレスにジャンプしてしまった場合にも、このRST 7が実行されます。

正規に終了してブレイクしたのか、暴走の結果なのかは、表示されたアドレスによって判断できます。

4章　プログラムのSAVE、LOAD

1.　はじめに

MYCPU80のRAMはボタン電池でバックアップをしていますから、RAMに書き込んだプログラムやデータは電源を切っても消えずにそのまま残っています。
しかし複数のプログラムをRAMに常駐させて保存するというのはあまり感心できる方法ではありません。プログラムが暴走したりすれば、プログラムもデータも一瞬で破壊されてしまいます。
MYCPU80は、パソコンとUSBケーブルで接続することによって、RAMにあるプログラムやデータをパソコンに送り、テキストファイルの形で保存することができます。
逆にパソコン上で8080アセンブラを使って作成したマシン語のプログラムをUSBケーブル接続でMYCPU80のRAMに送ることもできます。
USB接続によるパソコンとの間でのプログラムの送受信は、TK80モニタの機能によらなくても、RAMだけの場合でも先にブートプログラムを書いておくことによっても実行することができますが、TK80モニタの機能を利用する方がはるかに簡単で便利です。

ここではTK80モニタ機能を使った、プログラムのSAVE、LOADの方法を説明します。
なおDOS／Vパソコンには、USBシリアル変換ドライバのインストールが完了していて、USBケーブルでMYCPU80とDOS／Vパソコンが接続されており、RS232C送信および受信プログラムが起動できるように、DOS／Vパソコン上でDOSプロンプトが実行されているものとします。

2. プログラムのSAVEの仕方

上に書いたように、あらかじめDOSプロンプトを実行してデスクトップ画面にDOS窓が表示されている状態にしておいてください。
またDOS／VパソコンとMYCPU80とはUSBケーブルで接続しておいてください。
MYCPU80のRAMのプログラムをDOS／Vパソコンに送る場合には、先にDOS／Vパソコン側の受信プログラムを起動させておきます。

2. 1　DOS／Vパソコン側の操作

DOS窓が開いている状態で、キーボードから次のように入力します。
受信したプログラムを TEST. HTXという名前でSAVEする場合のキー入力例です。
R232. EXEで指定したファイル名がすでに存在すると、新しいファイルで上書きしてしまいますから注意してください。
大文字でも小文字でもかまいませんが、必ず半角で入力してください（入力モードは「直接入力」にしておくのが確実です）。

R232　TEST. HTX［Enter］　……［Enter］はEnterキーのことです
　　↑スペースを1文字空けてください

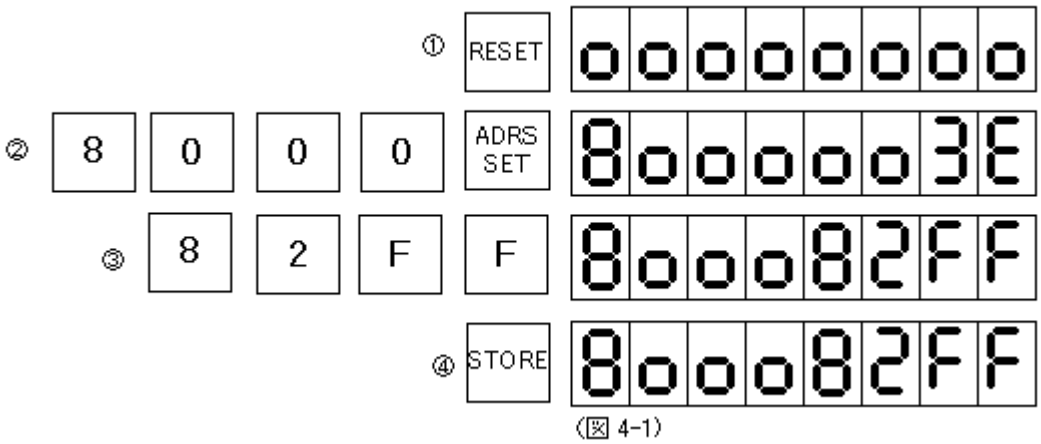
MYCPU80がUSBケーブルでパソコンに接続されていて、MYCPU80に電源が入っていると、次のように表示されます。

MYCPU80はCOM3に接続されました　（注）
送信を開始してください

（注）パソコンのハードウェア構成によって、COM3以外の場合もあります。

2. 2　MYCPU80側の操作

図 4-1 の通りに操作して下さい。ここでは例として8000～82FFの内容をSAVEすることにします。



（図 4-1）

①まずリセットして下さい。（リセットしなくてもよいのですが、慣れないうちは、こうしたほうが確実です）
②SAVE開始アドレス（この例では8000）をアドレス表示部にセットします。データ表示部には8000番地の内容が表示されます（ここでは3Eになっています）。
③続いてデータ表示部に、SAVE終了アドレスを入力します（[WRINC]キーは絶対に押さないように!!）。
④最後に[STORE]キーを押すと送信が開始されます。
　送信中や送信が終了しても7セグメントLEDの表示は変化しませんが、MYCPU80回路のHLレジスタのLEDが送信中のアドレスをカウントアップ表示します。
　送信が完了するとDOS／VパソコンのDOS窓に、

1546bytes　recieved
end

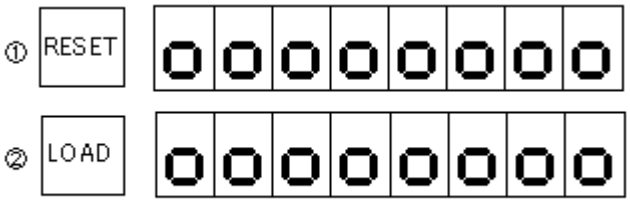
のように受信バイト数が表示されるので、受信が完了したことがわかります。

3.　プログラムのLOADの仕方

　さきほど説明したSAVEとは逆で、LOADの場合には先にMYCPU80の側を受信スタンバイにします。

3．1　MYCPU80側の操作

　LOADの操作は非常に簡単で、ただ[LOAD]キーを押すだけです（念のため先にリセットしてから[LOAD]を押したほうが確実です）。
　[LOAD]を押しても7セグメントの表示は変化しませんが、MYCPU80回路のレジスタLEDの表示が固定しますから受信スタンバイになったことがわかります（[LOAD]キーはすぐに離してください）。



（図4－2）

3．2　DOS／Vパソコン側の操作

デスクトップ画面にDOS窓が表示されている状態でキーボードから次のように入力します。
　TEST2．HTXという名前のファイルの内容をMYCPU80に送信する場合のキー入力例です。

　　W232　TEST2．HTX[Enter]またはw232　test2．htx[Enter]

　するとつぎの表示が出てただちに送信が開始されます。

　MYCPU80はCOM3に接続されました　（注）

プログラムを送り終わると、以下のように表示されます。

```
send  data  522bytes
end
```

（注）パソコンのハードウェア構成によって、COM3以外の場合もあります。

TK80回路の7セグメントLEDには、受信しメモリに格納された先頭のアドレスと終りのアドレスが表示されます。

（図4－3）

［注記1］

SAVE、LOAD終了後、アドレスが表示されている状態では普通のキー入力モードになっています。したがってこの状態ですぐにRUNキーを押せば、今SAVE（またはLOAD）したプログラムをただちに実行させることができます。またRDINCキーなど他のキーを使うこともできます（キー入力に先立ってリセットする必要はありません）。

なおSAVE、LOADの説明では、「プログラム」のSAVE、LOADということで説明してきましたが、プログラムでも単なるデータでも、扱いは全く同じです。

［注記2］

第6章　応用プログラム　のOHAYO. HTXとSOUND. HTXはCDROMに入っていて、MYCPU80フォルダにCOPYされているはずですから、

```
W232  OHAYO. HTX[Enter]
W232  SOUND. HTX[Enter]
```

とキー入力することによってMYCPU80に送信し、ただちに実行させることができます。

5章　I／O制御

1．　はじめに

　CPUはメモリとの間でデータやプログラムを書いたり読んだりします。
　この取扱説明書もいままで説明した部分は、全てメモリに対して読んだり書いたりする作業が基本になっていました。
　それに対してこの章では、外部に対して働きかける方法について説明します。

●リレーやスイッチはCPUと直結できない

　CPUはメモリに対しては直接読んだり書いたりすることができます。ハード的に説明するならばCPUとメモリとはアドレスバスやデータバスを直接つなぐことができます。
　ところが例えばスイッチやリレーから信号をCPUに送ったり、逆にCPUからのデータでLEDを光らせたり、リレーをON、OFFさせたりすることは、直接CPUとの間で行うわけにはいきません。
　勿論TK80回路に使われているICでは、リレーを直接駆動させることは電気的に考えて無理があります。普通はトランジスタが必要です。
　ここで直接制御できないと言ったのは、そういう電気的な問題ではなくて、回路そのものが直接つなぐことができないのです。

●アドレスバスとデータバス

　メモリとCPUとの間でデータをやりとりするには、データバス（回路図で D0～D7 と表示されているライン）を通じて行います。8080にはメモリは最大64KBも接続できますが、データバスはたった8本しかありません。つまり一度に8ビット＝1バイトのデータしか読んだり書いたりできません。
　そこでアドレスが必要になってきます。CPUはデータの読み書きをする場合に、その対象になっているアドレスをまず出力し、それによってメモリの特定部分のみを選択するのです。アドレス信号はアドレスバス（A0～A15 ）を通してメモリに与えられます。そしてメモリICは、アドレス信号（およびその他の制御信号）が与えられると、メモリIC自体の働きで、該当するアドレスの記憶場所（メモリセルなどと言います）だけがデータバスにつながるようになっています。
　リレーやスイッチなどには、いま説明したアドレスによる選択機構はついていません。
　これらの外部回路、外部装置部品とのデータのやりとりも、メモリと同じようにデータバスを通じて行われ、その選択はやはりアドレスバスに出力されるアドレス信号によって行われます。（ただしメモリの場合と違って、アドレスバスの下位8ビット（A0～A7）のみが使用されます）
　またCPUからの出力データは瞬間的に出されるだけなので、それを保持するラッチ回路も必要です。

●I／Oインターフェース回路

　そこで、前述のスイッチやリレーなどとCPUの間にアドレスやデータの受け渡しをする、特別な回路が必要になります。
　そのような回路は、入力と出力を別々にして単純な機能にするならば、普通のTTL回路でも作ることができます。
　TK80回路には汎用のロジックICを使った出力回路と入力回路があって、ユーザーが利用することができるようになっています。

　なお今までのところで説明してきたメモリのアドレスとは異なり、I／Oアドレスは16進2桁しかありません。（メモリアドレスは16進4桁）
　命令もメモリに対するもの（代表的なものはMOV命令）とは区別されており、IN、OUT命令を使います。

2．　I／Oインターフェース回路に対するデータ入出力の方法

2．1　I／Oアドレス

　ＴＫ８０回路の、ユーザーが入出力に利用できるＩ／Ｏアドレスは９８～９Ｂです。
　Ｉ／Ｏアドレスの下位２ビットはデコードしてないため、９８～９Ｂのどのアドレスを指定しても同じ回路がアクティブになります。
　入力も出力も同じ９８～９Ｂに対して行いますが、入力はＩＯＲＤ、出力はＩＯＷＲ信号でコントロールしていますから、入力と出力がぶつかることはありません。

　アドレス　９８～９Ｂ　８ビット出力ポート
　　ビット０～ビット３　ＰＩＣ１６Ｆ８８コントロール（ユーザーは使えません）

ビット４　スピーカー出力
ビット５～７　ユーザー用出力

アドレス　98～9B　8ビット入力ポート
ビット0～2　PIC16F88コントロール（ユーザーは使えません）
ビット3～7　ユーザー用入力

（1）データの出力
Aレジスタのデータが OUT命令によって、指定したI／Oアドレスの出力回路から出力されます。
命令は下のようにコーディングします。

D398　OUT 98　……Aレジスタの内容がI／Oアドレス98の出力回路から出力される

〔注記〕I／Oアドレス98～9Bの出力回路から外部に出力されるデータはラッチされています。
したがって新たに別のデータをそのポートから出力するまではもとのデータの出力が維持されます。
I／Oアドレス98～9Bの出力回路のビット0～3はシステム専用です。出力データの下位4ビットは0000または0100にしてください。

（2）データの入力
IN 命令によって、指定したI／Oアドレスの入力回路からのデータが A レジスタに入ります。
命令は下のようにコーディングします。

DB98　IN 98　……I／Oアドレス98の入力回路から入力されたデータがAレジスタに入る。

〔注意〕入力データはラッチされません。
I／Oアドレス98～9Bの入力回路のビット0～2はシステム専用です。入力データの下位3ビットはユーザーにとって意味はありません。

3．　スピーカの使用方法

I／Oアドレス98～9Bの出力回路のビット4はスピーカ出力回路につながっています。
したがって　I／Oアドレス98～9Bの出力回路のビット4から、任意の周波数のパルスを出力することにより、その周波数に相当する高さの音を出すことができます。
具体的な使い方については、6章　応用プログラム　2．電子オルガンプログラム　を参照してください。

6章　応用プログラム

1．　OHAYO（オハヨー）

このプログラムは中日電工のオリジナルではありません。雑誌だったのか、どういう本に載っていたのかも思い出せません。もちろんプログラムリストなどありません。
確かこんな動作をしてたよねぇ、という感じで作ったのがこのプログラムです。
簡単なプログラムですが、なかなか味があって面白いと思います。

プログラムを入力して、[8][0][0][0][ADRSSET][RUN]とするとコンピュータが起きあがってのそりのそりと歩き始めます（LEDに表示されるのは足跡のみ）。
目をパチパチさせて、それからゆっくりと「おはよー」と声をかけます（言葉は話せませんからLEDにそれもローマ字で表示します）。
このプログラムはそれだけです。

1．1 プログラムの説明

LEDに0～F以外のパターンを表示するとか、表示を全部クリアして（0を表示するのではなくて）ブランクにしたいときなどは、LED表示アドレス（＄FFF8～＄FFFF）に直接書き込みます。各アドレスの8ビットのデータのうち1のビットに対応するLEDのセグメントが点灯し0のビットに対応するセグメントは消灯します。以下のプログラムの中でLED1～LED8に対してデータを書き込んでいるところは全部この目的で使われています。
;EYE close／openやサブルーチンCLRがその例です。;ASIATO dispと;ohayo dispでは表示するデータを8バイト分用意しておいて、全部の表示を順次置換えています。
サブルーチンCLRは＄FFF8～＄FFFFに00を書き込むことでLEDを全消灯しています。

1．2 プログラムリスト

```
2009/10/1  14:16  OHAYO.TXT
END=8088
              ; OHAYO for MYCPU80
              ; 09/10/1
              ;
                   ORG $8000
                   LED1=$FFF8
                   LED4=$FFFB
                   MONRST=$0051
                   D1=$02DD
              ;
8000 CD5880        CALL CLR
              ;ASIATO disp
8003 217980        LXI H,ASIDT
8006 11F8FF        LXI D,LED1
8009 0608          MVI B,08
800B CD6580   ASIDP2:CALL TM1S
800E 7E            MOV A,M
800F 12            STAX D
8010 23            INX H
8011 13            INX D
8012 05            DCR B
8013 C20B80        JNZ ASIDP2
8016 CD6580        CALL TM1S
8019 CD5880        CALL CLR
801C CD6580        CALL TM1S
              ;EYE close/open
801F 0603          MVI B,03
8021 211C1C        LXI H,$1C1C
8024 113F3F        LXI D,$3F3F
8027 22FBFF   EYE:SHLD LED4
```

```
802A CD6B80        CALL TM025
802D EB            XCHG
802E 22FBFF        SHLD LED4
8031 EB            XCHG
8032 CD6880        CALL TM05
8035 05            DCR B
8036 C22780        JNZ EYE
8039 CD5880        CALL CLR
                ;ohayo disp
803C 218180        LXI H,OHAYODT
803F 11F8FF        LXI D,LED1
8042 0608          MVI B,08
8044 CD6880    OHYDP2:CALL TM05
8047 7E            MOV A,M
8048 12            STAX D
8049 23            INX H
804A 13            INX D
804B 05            DCR B
804C C24480        JNZ OHYDP2
804F CD6580        CALL TM1S
8052 CD6580        CALL TM1S
8055 C35100        JMP MONRST
                ;LED clear
8058 21F8FF    CLR:LXI H,LED1
805B 010008        LXI B,$0800
805E 71        CLR2:MOV M,C
805F 23            INX H
8060 05            DCR B
8061 C25E80        JNZ CLR2
8064 C9            RET
                ;1sec timer/0.5sec timer
8065 CD6880    TM1S:CALL TM05
8068 CD6B80    TM05:CALL TM025
806B D5        TM025:PUSH D
806C C5            PUSH B
806D 0635          MVI B,35;=53
806F CDDD02    TM025_2:CALL D1;4.727MS
8072 05            DCR B
8073 C26F80        JNZ TM025_2
8076 C1            POP B
8077 D1            POP D
8078 C9            RET
                ;asiato data
8079 43        ASIDT:DB 43
807A 4C            DB 4C
807B 43            DB 43
807C 4C            DB 4C
807D 43            DB 43
807E 4C            DB 4C
807F 43            DB 43
8080 4C            DB 4C
                ;"ohayo---"
8081 3F        OHAYODT:DB 3F;O
8082 76            DB 76;H
8083 77            DB 77;A
8084 6E            DB 6E;y
8085 3F            DB 3F;O
8086 40            DB 40;-
```

```
8087 40            DB,40;-
8088 40            DB 40;-
                 ;
ASIDP2       =800B  ASIDT        =8079  CLR          =8058
CLR2         =805E  D1           =02DD  EYE          =8027
LED1         =FFF8  LED4         =FFFB  MONRST       =0051
OHAYODT      =8081  OHYDP2       =8044  TM025        =806B
TM025_2      =806F  TM05         =8068  TM1S         =8065
```

2. 電子オルガンプログラム

TK80回路のキーボードを利用して、各キーに対応する高さの音を発生させるプログラムです。
ここでは音の高さが周波数によって決まることを利用し、それぞれの音の高さに対応する周波数のパルスを発生させています。

2.1 プログラムリスト

```
2009/10/9  17:11  sound.txt
END=804D
               ;;; sound 09.10.1 10.9
               ;;;
                 ORG $8000
               ;
                 KEY=$0247
               ;
8000 CD4702    SND:CALL KEY
8003 3C          INR A
8004 CA0080      JZ SND
8007 3D          DCR A
8008 CD0E80      CALL SNDSB
800B C30080      JMP SND
               ;
800E F5        SNDSB:PUSH PSW
800F E5          PUSH H
8010 D5          PUSH D
8011 213680      LXI H,SNDTBL
8014 85          ADD L
8015 6F          MOV L,A
8016 1E1A        MVI E,1A
8018 56        SNDS1:MOV D,M
8019 3E14        MVI A,14
801B D398        OUT 98
801D 00        SNDS2:NOP
801E 00          NOP
801F 15          DCR D
8020 C21D80      JNZ SNDS2
8023 56          MOV D,M
8024 3E04        MVI A,04
8026 D398        OUT 98
8028 00        SNDS3:NOP
8029 00          NOP
802A 15          DCR D
802B C22880      JNZ SNDS3
802E 1D          DCR E
802F C21880      JNZ SNDS1
8032 D1          POP D
8033 E1          POP H
8034 F1          POP PSW
```

```
8035 C9          RET
                 ;
                 ; SOUND TABLE
8036 7F         SNDTBL:DB 7F;so4
8037 77          DB 77;so#4
8038 71          DB 71;ra4
8039 6A          DB 6A;ra#4
803A 5F          DB 5F;do5
803B 59          DB 59;do#5
803C 54          DB 54;re5
803D 4F          DB 4F;re#5
803E 47          DB 47;fa5
803F 43          DB 43;fa#5
8040 3F          DB 3F;so5
8041 3B          DB 3B;so#5
8042 35          DB 35;ra#5
8043 32          DB 32;si5
8044 2F          DB 2F;do6
8045 2C          DB 2C;do#6
8046 25          DB 25;mi6
8047 27          DB 27;re#6
8048 2A          DB 2A;re6
8049 4B          DB 4B;mi5
804A 38          DB 38;ra5
804B 64          DB 64;si4
804C 23          DB 23;fa6
804D 21          DB 21;fa#6
                 ;END
KEY          =0247  SND          =8000  SNDS1        =8018
SNDS2        =801D  SNDS3        =8028  SNDSB        =800E
SNDTBL       =8036
```

２．１ 各キーと音との対応

RET レ＃ （６）	RUN ミ （６）	STORE ファ （６）	LOAD ファ＃ （６）	RESET
C ラ＃ （５）	D シ （５）	E ド （６）	F ド＃ （６）	ADRSSET レ （６）
8 ファ （５）	9 ファ＃ （５）	A ソ （５）	B ソ＃ （５）	RD INC ラ （５）
4 ド （５）	5 ド＃ （５）	6 レ （５）	7 レ＃ （５）	RD DEC ミ （５）
0 ソ （４）	1 ソ＃ （４）	2 ラ （４）	3 ラ＃ （４）	WR INC シ （４）

（図６－１）

[注記]各音の表示の下の(４)～(６)はオクターブを示しています。

２．３ 操作

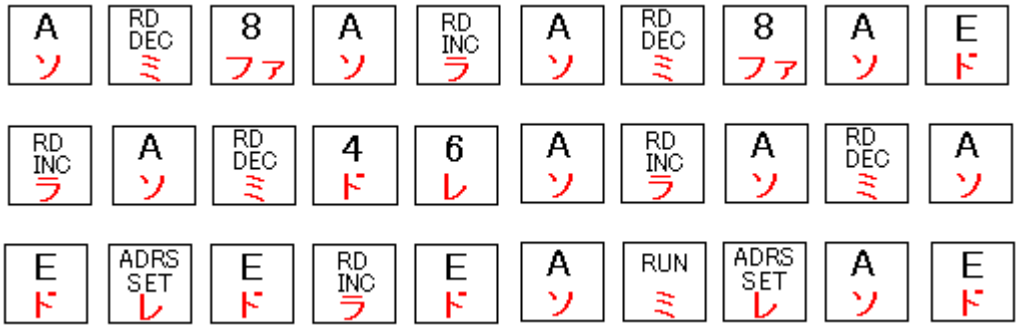
プログラムを入力後、８０００番地からRUNさせると、それ以後はキーを押すとその間中キーに対応する高さの音がスピーカから出力されます。

なおモニタサブルーチン０２４７は、キーの状態を一回だけスキャンしてチェックしどのキーも押されていなければAレジスタにFFHを入れてリターンします。

キーが押されたときはそのキーコード（00H～17H）をAレジスタに入れてリターンします。

［キー操作例］

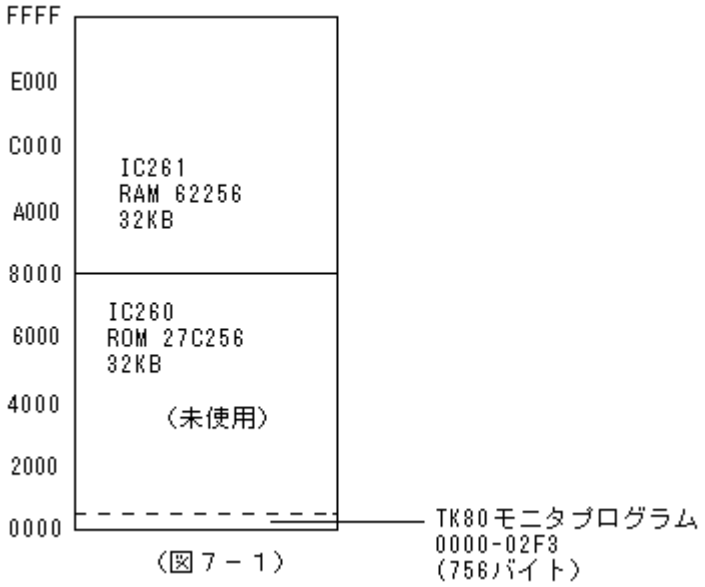
次のようにキーを押していってみてください（さて何の曲でしょう？）。



7章　メモリマップ・I／Oマップ

1．メモリマップ

TK80回路はモニタROM（32KB）とRAM（32KB）で構成されています。
モニタROMは32KB（キロバイト）の27C256ですが、ほとんどは使われていません。実際にモニタプログラムに使用されているのは、たったの756バイト（0．756KB）です。



（図７－１）

2．システムワークエリア

IC261のRAMのうちFFC7～FFFFはモニタプログラムのワークエリアとして使用されます。この範囲はモニタプログラムの専用エリアなので、ユーザーがプログラムなどを書くことはできません。
参考までにそのワークエリアのメモリマップを示します。
なお、これらはモニタプログラムがそれぞれの目的のために管理するエリアなので、通常はユーザーが意識する必要はありません。しかしより高度なプログラムを書く場合などには、知っておくと便利なこともあります。

FFFF	LED 表示用セグメントデータバッファ　No.8
FFFE	LED 表示用セグメントデータバッファ　No.7
FFFD	LED 表示用セグメントデータバッファ　No.6
FFFC	LED 表示用セグメントデータバッファ　No.5
FFFB	LED 表示用セグメントデータバッファ　No.4
FFFA	LED 表示用セグメントデータバッファ　No.3
FFF9	LED 表示用セグメントデータバッファ　No.2
FFF8	LED 表示用セグメントデータバッファ　No.1
FFF7	LED 表示用データレジスタ　No.4
FFF6	LED 表示用データレジスタ　No.3
FFF5	LED 表示用データレジスタ　No.2
FFF4	LED 表示用データレジスタ　No.1
FFF3	キー入力フラグ
FFF2	ブレイクカウンタ
FFF1	ブレイクアドレス（H）
FFF0	ブレイクアドレス（L）
FFEF	アドレスレジスタ（H）
FFEE	アドレスレジスタ（L）
FFED	データレジスタ（H）
FFEC	データレジスタ（L）
FFEB	CPUレジスタセーブエリア　A
FFEA	CPUレジスタセーブエリア　F
FFE9	CPUレジスタセーブエリア　B
FFE8	CPUレジスタセーブエリア　C

FFE7	CPUレジスタセーブエリア　D
FFE6	CPUレジスタセーブエリア　E
FFE5	CPUレジスタセーブエリア　H
FFE4	CPUレジスタセーブエリア　L
FFE3	CPUレジスタセーブエリア　SP(H)
FFE2	CPUレジスタセーブエリア　SP(L)
FFE1	CPUレジスタセーブエリア　PC(H)
FFE0	CPUレジスタセーブエリア　PC(L)
FFDF	RST6ジャンプ先アドレス(H)
FFDE	RST6ジャンプ先アドレス(L)
FFDD	RST6ジャンプコード(C3)
FFDC	RST5ジャンプ先アドレス(H)
FFDB	RST5ジャンプ先アドレス(L)
FFDA	RST5ジャンプコード(C3)
FFD9	RST4ジャンプ先アドレス(H)
FFD8	RST4ジャンプ先アドレス(L)
FFD7	RST4ジャンプコード(C3)
FFD6	RST3ジャンプ先アドレス(H)
FFD5	RST3ジャンプ先アドレス(L)
FFD4	RST3ジャンプコード(C3)
FFD3	RST2ジャンプ先アドレス(H)
FFD2	RST2ジャンプ先アドレス(L)
FFD1	RST2ジャンプコード(C3)
FFD0 FFC7	モニタ用スタックエリア
FFC6	ユーザー用スタックエリア

3.　RSTジャンプテーブル

システムワークエリアの中に、RST6～RST2ジャンプテーブルがあります。
これはユーザーがプログラム中でRST命令を使ったり、あるいは割り込み処理を行ったときに、ユーザー領域にジャンプさせるためのものです。
割込みには通常はRST7を使うのですが、TK80回路ではRST7をステップ動作に使用しているためにユーザーが使うことはできません。
TK80回路でユーザーに開放されているのはRST6～RST2です。
RST命令のエントリアドレスは0000～0038の間のアドレスで8バイトごとに置かれています。
TK80回路ではそのアドレスはTK80モニタROMの領域なので、そこにユーザーが任意のジャンプ命令などを自由に書き込むことはできません。
そのための対策として、ROMに置かれている本来のRST命令のエントリアドレスには、上のメモリマップにあるRAMのアドレスへのジャンプ命令が書かれています。
たとえばアドレス0010はRST2のエントリアドレスですが、TK80モニタROMの0010には下のように書かれています。

0010　C3D1FF　JMP　RST2

ユーザープログラムの中でRST2命令が実行されると(あるいは割込みによってRST2が実行されると)、RAMのFFD1番地にジャンプします。
ユーザープログラムの先頭で、FFD1～FFD3にユーザーが希望するRST2の処理ルーチンへのジャンプ命令を書き込むようにしておくことによって、ユーザーがRST命令を利用することができるようになります。
たとえば8200にジャンプさせたい場合には、FFD1にC3を、FFD2に00を、FFD3に82を書き込んでからRUNさせます。あるいはユーザープログラムの先頭に次の命令を書いておきます)

```
3EC3    MVI A,C3
32D1FF  STA $FFD1
210082  MVI H,$8200
```

22D2FF SHLD $FFD2

４．　Ｉ／Ｏマップ

　ＴＫ８０回路ではＩ／Ｏアドレスとして９４～９Ｆを使っています。
　なぜそのような半端なアドレスを使っているかといいますと、デコード回路が簡単になるという理由からだけです。
　さらにデコード回路を簡略化するために、アドレスラインの下位２ビットＡ１、Ａ０はデコードされていません。
　そのためＩ／Ｏアドレスの９４～９７、９８～９Ｂ、９Ｃ～９Ｆは同じセレクト回路をアクティブにします。
　また入力と出力の別々の回路に同じＩ／Ｏアドレスを割り付けています。これはシステムが占有するＩ／Ｏアドレスを少なくするという目的からです。

　ＴＫ８０回路のＩ／Ｏマップです。
Ａ０～ＦＦ　未使用
９Ｃ～９Ｆ　５×５キーマトリクスラインセレクト（出力）
　　５×５キーマトリクスデータ（入力）
９８～９Ｂ　８ビット出力ポート
　　ビット０～ビット３　ＰＩＣ１６Ｆ８８コントロール
　　ビット４　スピーカー出力
　　ビット５～７　ユーザー用出力
　　８ビット入力ポート
　　ビット０～２　ＰＩＣ１６Ｆ８８コントロール
　　ビット３～７　ユーザー用入力
９４～９７　ＰＩＣ１６Ｆ８８とのデータ送受信（入力および出力）
００～９３　未使用

8章　モニタサブルーチン

1.　はじめに

TK80回路のモニタプログラムには、幾つかのサブルーチンが含まれており、この中にはユーザーが利用すると便利なものもあります。

ここではそのようなサブルーチンをリストアップして、簡単な説明を加えました。具体的なプログラム内容については、9章のモニタプログラムリストを参照して下さい。

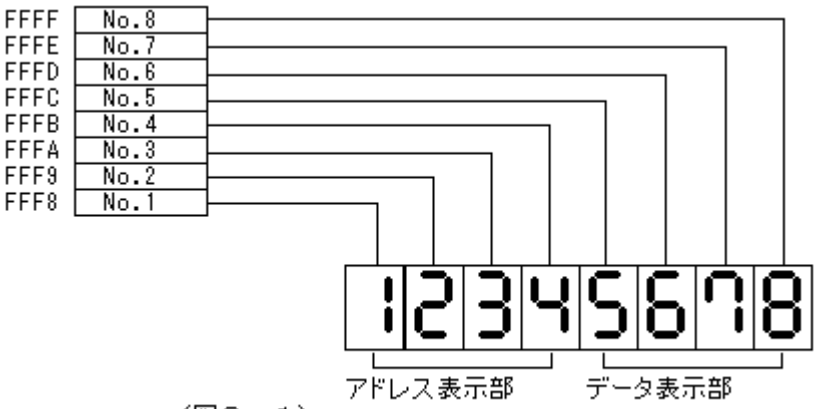
2.　LED表示

2．1　セグメント表示バッファとLED表示の関係

LEDに何かを表示させるには、RAM内のセグメント表示バッファ（FFF8～FFFF）にセグメントデータを書き込みます。セグメント表示バッファはDMA回路によって毎秒数百回読み出され、7セグメント表示回路にラッチされ、自動的にLEDにダイナミック表示されます。

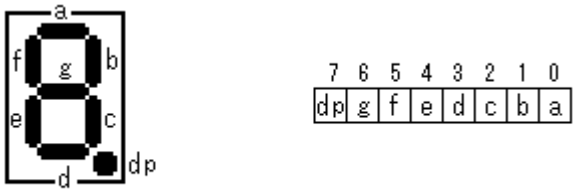
セグメント表示バッファから表示データを読み出してLEDにダイナミック表示するまでのプロセスはハードウェアが機械的に行いますから、ソフトウェアでは単にセグメント表示バッファに表示データを書き込むだけで、そのほかの作業は必要ありません。

セグメント表示バッファとLED表示器の各桁とは、下図のように対応しています。



（図８－１）

セグメント表示バッファ内のデータの各ビットはLED表示器1桁のセグメントと下図のように関係しています。対応するビットが1のとき、そのセグメントが点灯します。



（図８－２）

たとえば　2　という表示に対応するデータは、a、b、d、e、g＝1なので、01011011（5B）になります。

一般的には0～Fを表示するという使い方になるのですが、セグメント表示バッファはそこに書き込まれたデータのビット情報をそのままLEDのセグメントに置き換えて表示しますから、セグメントで表現できる任意の表示パターンを表示させることができます。

例）Hという文字を表示させるには、b、c、e、f、gを1にする、つまり01110110（76）をセグメント表示バッファに書きます。

2．2　セグメントデータ変換ルーチン

開始アドレス　01C0

使用レジスタ　A、F、B、C、D、E、H、L

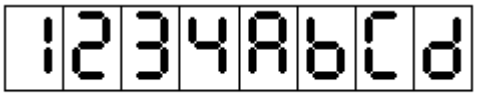
表示用データレジスタ（FFF4～FFF7）の内容を、16進数からセグメント表示データに変換してセグメント表示バッファ（FFF8～FFFF）に転送します。

2．1で説明したセグメント表示のプロセスのみを利用して、メモリの値などをLEDに表示させようとすると、表示する各桁ごとに16進数をビットデータに変換しなければなりません。

実際にはその変換は必要不可欠なものなのですが、そのためのプログラムの負担を軽減するために、あらかじめLEDの2桁を表示用データレジスタ1個に割り当てておいて、各表示用レジスタにデータを書き込んだあと、このルーチンをCALLすることで、16進数がセグメント表示データに変換されてLEDに表示されます。

表示用データレジスタがそれぞれ次の内容であったとき、このサブルーチンをCALLすると、LEDには図8-3のように表示されます。

表示用データレジスタ　No.4　FFF7=CD
表示用データレジスタ　No.3　FFF6=AB
表示用データレジスタ　No.2　FFF5=34
表示用データレジスタ　No.1　FFF4=12



（図8－3）

2．3　アドレスレジスタ、データレジスタ表示ルーチン

開始アドレス　01A1

使用レジスタ　A、F、B、C、D、E、H、L

LED表示を行うメモリアドレスとそのデータや、LOAD、SAVEを行う場合の開始アドレスと終了アドレスなどは、必ずRAMのアドレスレジスタ（FFEE～FFEF）、データレジスタ（FFEC～FFED）にまず入れられます。

キーから入力されるデータやADRS　SETキーによってLEDのアドレス表示部に表示されたデータは、じつはこの4バイトのレジスタエリアの内容がこの表示ルーチンによって表示されていたのです。

この取扱説明書の前の章で、「アドレス表示部にアドレスをセットする」とか「データ表示部にデータを入れる」などと表現してきましたが、それはその方が理解し易いと判断したためで、正しい表現ではそれぞれ「アドレスレジスタにアドレスをセットする」、「データレジスタにデータを入れる」になります。

このサブルーチンはアドレスレジスタとデータレジスタの内容をまず、表示用データレジスタに転送したあと、セグメントデータ変換ルーチンをCALLします。

アドレスレジスタ、データレジスタと表示用データレジスタとの関係は次のようになります。

アドレスレジスタ(H)FFEF　→　表示用データレジスタ　No.1　FFF4
アドレスレジスタ(L)FFEE　→　表示用データレジスタ　No.2　FFF5
データレジスタ(H)FFED　→　表示用データレジスタ　No.3　FFF6
データレジスタ(L)FFEC　→　表示用データレジスタ　No.4　FFF7

3．　キー入力

3．1　キー入力ルーチン①

開始アドレス　0216

使用レジスタ　A、F、B、D、E

キーボードの入力をチェックし、どのキーも押されていなければ、押されるまで待ちます。

キーが押されると、そのキーに対応する数値（キーコード）をAレジスタに入れてリターンします。

キーとキーコードの対応を下に示します。

ｺｰﾄﾞ	キー	ｺｰﾄﾞ	キー	ｺｰﾄﾞ	キー	ｺｰﾄﾞ	キー
00	0	06	6	0C	C	12	ADRSSET
01	1	07	7	0D	D	13	RD DEC
02	2	08	8	0E	E	14	RD INC
03	3	09	9	0F	F	15	WR INC
04	4	0A	A	10	RUN	16	STORE
05	5	0B	B	11	RET	17	LOAD

3．2　キー入力ルーチン②

開始アドレス　0223

使用レジスタ　A、F、B、D、E

キー入力ルーチン①はキー入力があるまで待ちつづけますが、このキー入力ルーチン②はキーをスキャンしてキーが押されていなければBレジスタにFFを入れてリターンします。

キーが押されていれば、対応するキーコードをBレジスタに入れてリターンします。

4.　タイマー

4．1　タイマールーチン①(4．727ms)

開始アドレス　02DD
使用レジスタ　F、D、E

4．2　タイマールーチン②(9．432ms)

開始アドレス　02EA
使用レジスタ　F、D、E

4．3　タイマールーチン③(28．307ms)

開始アドレス　02EF
使用レジスタ　F、D、E

```
2009/6/5  17:51  TK80MON4.TXT
END=02F3
              ;;; TK80 MONITOR PROGRAM FOR MC80
              ; 09/5/28 09/6/1 6/3 6/5
              ;
                ORG $0000
              ;
                USRSP=$FFC7
              ;
                MONSP=$FFD1
                RST2=$FFD1
                RST3=$FFD4
                RST4=$FFD7
                RST5=$FFDA
                RST6=$FFDD
                PSAVE=$FFE0
                SSAVE=$FFE2
                LSAVE=$FFE4
                HSAVE=$FFE5
                ESAVE=$FFE6
                DSAVE=$FFE7
                CSAVE=$FFE8
                BSAVE=$FFE9
                FSAVE=$FFEA
                DATA=$FFEC
                DATA1=$FFED
                ADRES=$FFEE
                ADRES1=$FFEF
                BRKAD=$FFF0
                BRKCT=$FFF2
                KFLAG=$FFF3
                DISP=$FFF4
                DIG=$FFF8
              ;

0000 C33B00     JMP MONST
              ;
                ORG $0008
0008 C35100     JMP START
              ;
                ORG $0010
0010 C3D1FF     JMP RST2
              ;
                ORG $0018
0018 C3D4FF     JMP RST3
              ;
                ORG $0020
0020 C3D7FF     JMP RST4
              ;
                ORG $0028
0028 C3DAFF     JMP RST5
              ;
                ORG $0030
0030 C3DDFF     JMP RST6
              ;
                ORG $0038
```

```
0038 C35101     JMP BRENT
                ;
                ; INITIALIZE ROUTINE
                ;
003B 21ECFF   MONST:LXI H,DATA
003E 060C       MVI B,0C
0040 AF         XRA A
0041 77       MONST2:MOV M,A
0042 23         INX H
0043 05         DCR B
0044 C24100     JNZ MONST2
0047 21C7FF     LXI H,USRSP
004A 22E2FF     SHLD SSAVE
004D C35100     JMP START
0050 00         NOP
                ;
                ; MONITOR START
                ;
0051 3EF7     START:MVI A,F7
0053 D398       OUT 98;PIC reset
0055 31D1FF     LXI SP,MONSP
0058 CDC001     CALL SEGCG
005B CD1602     CALL KEYIN
005E 47         MOV B,A
005F E610       ANI 10
0061 CA8400     JZ DIGIT
0064 78         MOV A,B
0065 E60F       ANI 0F
0067 0600       MVI B,00
0069 87         ADD A
006A 4F         MOV C,A
006B 217400     LXI H,TABL
006E 09         DAD B
006F 7E         MOV A,M
0070 23         INX H
0071 66         MOV H,M
0072 6F         MOV L,A
0073 E9         PCHL
                ;
0074 CC00     TABL:DW GOTO
0076 F901       DW RESRG
0078 9400       DW ADSET
007A B800       DW ADDCX
007C 9D00       DW ADINX
007E C200       DW MEMW
0080 D500       DW SDATA
0082 0701       DW LDATA
                ;
0084 CDB501   DIGIT:CALL SHIFT
0087 3AECFF     LDA DATA
008A B0         ORA B
008B 32ECFF     STA DATA
008E CDA101     CALL RGDSP
0091 C35100     JMP START
                ;
                ; ADDRESS SET
                ;
0094 2AECFF   ADSET:LHLD DATA
```

```
0097 22EEFF     SHLD ADRES
009A C3A100     JMP ADINX2
                ;
                ; MEMORY READ & ADDRESS INCREMENT
                ;
009D 2AEEFF   ADINX:LHLD ADRES
00A0 23         INX H
00A1 CDAD00   ADINX2:CALL MEMR
00A4 22EEFF   ADSTR:SHLD ADRES
00A7 CDA101     CALL RGDSP
00AA C35100     JMP START
                ;
00AD 3AECFF   MEMR:LDA DATA
00B0 32EDFF     STA DATA1
00B3 7E         MOV A,M
00B4 32ECFF     STA DATA
00B7 C9         RET
                ;
                ; MEMORY READ & ADDRESS DECREMENT
                ;
00B8 2AEEFF   ADDCX:LHLD ADRES
00BB 2B         DCX H
00BC CDAD00     CALL MEMR
00BF C3A400     JMP ADSTR
                ;
                ; MEMORY WRITE
                ;
00C2 2AEEFF   MEMW:LHLD ADRES
00C5 3AECFF     LDA DATA
00C8 77         MOV M,A
00C9 C39D00     JMP ADINX
                ;
                ; MONITOR TO USER CONTROL ROUTINE
                ;
00CC 2AEEFF   GOTO:LHLD ADRES
00CF 22E0FF     SHLD PSAVE
00D2 C3F901     JMP RESRG
                ;
                ; STORE DATA
                ;
00D5 2AECFF   SDATA:LHLD DATA
00D8 EB         XCHG
00D9 2AEEFF     LHLD ADRES
00DC 3EFB       MVI A,FB;PIC active & I/O ADDRESS 94 "out"
00DE D398       OUT 98
00E0 7C         MOV A,H
00E1 CD7C02     CALL SOUT
00E4 7D         MOV A,L
00E5 CD7C02     CALL SOUT
00E8 7A         MOV A,D
00E9 CD7C02     CALL SOUT
00EC 7B         MOV A,E
00ED CD7C02     CALL SOUT
00F0 2B         DCX H
00F1 23       SDATA2:INX H
00F2 7E         MOV A,M
00F3 CD7C02     CALL SOUT
00F6 7D         MOV A,L
```

```
00F7 BB         CMP E
00F8 C2F100     JNZ SDATA2
00FB 7C         MOV A,H
00FC BA         CMP D
00FD C2F100     JNZ SDATA2
0100 CDB302     CALL SOUTEND
0103 C35100     JMP START
0106 00         NOP
                ;
                ;LOAD DATA
                ;
0107 3EFF       LDATA:MVI A,FF;PIC active
0109 D398         OUT 98
010B CDA002     CALL SIN
010E 67         MOV H,A
010F CDA002     CALL SIN
0112 6F         MOV L,A
0113 CDA002     CALL SIN
0116 57         MOV D,A
0117 CDA002     CALL SIN
011A 5F         MOV E,A
011B 22EEFF     SHLD ADRES
011E EB         XCHG
011F 22ECFF     SHLD DATA
0122 EB         XCHG
0123 2B         DCX H
0124 23       LDATA2:INX H
0125 CDA002     CALL SIN
0128 77         MOV M,A
0129 7D         MOV A,L
012A BB         CMP E
012B C22401     JNZ LDATA2
012E 7C         MOV A,H
012F BA         CMP D
0130 C22401     JNZ LDATA2
0133 CDA101     CALL RGDSP
0136 C35100     JMP START
                ;
                ; BREAK ENTRY
                ; BREAK & ONE STEP OPERATION
                ;
                  ORG $0151
                ;
0151 E3       BRENT:XTHL
0152 22E0FF     SHLD PSAVE
0155 F5         PUSH PSW
0156 210400     LXI H,$0004
0159 39         DAD SP
015A F1         POP PSW
015B 22E2FF     SHLD SSAVE
015E E1         POP H
015F 31ECFF     LXI SP,DATA
0162 F5         PUSH PSW
0163 C5         PUSH B
0164 D5         PUSH D
0165 E5         PUSH H
0166 31D1FF     LXI SP,MONSP
0169 3AF2FF     LDA BRKCT
```

```
016C A7          ANA A
016D CA8B01      JZ BSTOP
0170 2AF0FF      LHLD BRKAD
0173 EB          XCHG
0174 2AE0FF      LHLD PSAVE
0177 7D          MOV A,L
0178 BB          CMP E
0179 C28501      JNZ NOBRK
017C 7C          MOV A,H
017D BA          CMP D
017E C28501      JNZ NOBRK
0181 21F2FF      LXI H,BRKCT
0184 35          DCR M
0185 CD9101    NOBRK:CALL ADDSP
0188 C3F901      JMP RESRG
018B CD9101    BSTOP:CALL ADDSP
018E C35100      JMP START
0191 2AEAFF    ADDSP:LHLD FSAVE
0194 22ECFF      SHLD DATA
0197 2AE0FF      LHLD PSAVE
019A 22EEFF      SHLD ADRES
019D CDA101      CALL RGDSP
01A0 C9          RET
                 ;
                 ;
                 ;;; SUBROUTINE
                 ;
01A1 21EFFF    RGDSP:LXI H,ADRES1
01A4 11F4FF      LXI D,DISP
01A7 0604        MVI B,04
01A9 7E        RGDSP2:MOV A,M
01AA 12          STAX D
01AB 2B          DCX H
01AC 13          INX D
01AD 05          DCR B
01AE C2A901      JNZ RGDSP2
01B1 CDC001      CALL SEGCG
01B4 C9          RET
                 ;
                 ;DATA REG SHIFT(4 BITS)
                 ;
01B5 2AECFF    SHIFT:LHLD DATA
01B8 29          DAD H
01B9 29          DAD H
01BA 29          DAD H
01BB 29          DAD H
01BC 22ECFF      SHLD DATA
01BF C9          RET
                 ;
                 ; SEGMENT CONVERT SUB
                 ;
01C0 21F4FF    SEGCG:LXI H,DISP
01C3 11F8FF      LXI D,DIG
01C6 01E901      LXI B,SEGD
01C9 7E        SEGCG2:MOV A,M
01CA 23          INX H
01CB E5          PUSH H
01CC F5          PUSH PSW
```

```
01CD E6F0       ANI F0
01CF 0F         RRC
01D0 0F         RRC
01D1 0F         RRC
01D2 0F         RRC
01D3 2600       MVI H,00
01D5 6F         MOV L,A
01D6 09         DAD B
01D7 7E         MOV A,M
01D8 12         STAX D
01D9 13         INX D
01DA F1         POP PSW
01DB E60F       ANI 0F
01DD 2600       MVI H,00
01DF 6F         MOV L,A
01E0 09         DAD B
01E1 7E         MOV A,M
01E2 12         STAX D
01E3 E1         POP H
01E4 1C         INR E
01E5 C2C901     JNZ SEGCG2
01E8 C9         RET
                ;
                ; SEGMENT DATA
                ;
01E9 5C       SEGD:DB 5C
01EA 06         DB 06
01EB 5B         DB 5B
01EC 4F         DB 4F
01ED 66         DB 66
01EE 6D         DB 6D
01EF 7D         DB 7D
01F0 27         DB 27
01F1 7F         DB 7F
01F2 6F         DB 6F
01F3 77         DB 77
01F4 7C         DB 7C
01F5 39         DB 39
01F6 5E         DB 5E
01F7 79         DB 79
01F8 71         DB 71
                ;
                ; REGISTER RESTORE
                ;
01F9 2AE2FF   RESRG:LHLD SSAVE
01FC F9         SPHL
01FD 2AE0FF     LHLD PSAVE
0200 E5         PUSH H
0201 2AE4FF     LHLD LSAVE
0204 E5         PUSH H
0205 2AEAFF     LHLD FSAVE
0208 E5         PUSH H
0209 2AE8FF     LHLD CSAVE
020C 4D         MOV C,L
020D 44         MOV B,H
020E 2AE6FF     LHLD ESAVE
0211 EB         XCHG
0212 F1         POP PSW
```

```
0213 E1          POP H
0214 FB          EI
0215 C9          RET
                 ;
                 ; KEY INPUT
                 ;
0216 CD2302    KEYIN:CALL INPUT
0219 47          MOV B,A
021A 3AF3FF      LDA KFLAG
021D A7          ANA A
021E CA1602      JZ KEYIN
0221 78          MOV A,B
0222 C9          RET
                 ;
                 ; KEY INPUT SUB
                 ;
0223 CD4702    INPUT:CALL KEY
0226 3C          INR A
0227 CA4202      JZ NOKEY
022A CDEA02    INPUT2:CALL D2
022D CD4702      CALL KEY
0230 47          MOV B,A
0231 3C          INR A
0232 CA4202      JZ NOKEY
0235 3AF3FF      LDA KFLAG
0238 A7          ANA A
0239 C22A02      JNZ INPUT2
023C 3D          DCR A
023D 32F3FF    INPUT3:STA KFLAG
0240 78          MOV A,B
0241 C9          RET
0242 06FF      NOKEY:MVI B,FF
0244 C33D02      JMP INPUT3
                 ;
                 ; KEY SCAN & CONVERT HEX DATA SUB
                 ;
0247 1600      KEY:MVI D,00
0249 42          MOV B,D
024A 3EFE        MVI A,FE
024C D39C        OUT 9C
024E DB9C        IN 9C
0250 EEFF        XRI FF
0252 C27102      JNZ KEYI
0255 0608        MVI B,08
0257 3EFD        MVI A,FD
0259 D39C        OUT 9C
025B DB9C        IN 9C
025D EEFF        XRI FF
025F C27102      JNZ KEYI
0262 0610        MVI B,10
0264 3EFB        MVI A,FB
0266 D39C        OUT 9C
0268 DB9C        IN 9C
026A EEFF        XRI FF
026C C27102      JNZ KEYI
026F 3D          DCR A
0270 C9          RET
0271 0F        KEYI:RRC
```

```
0272 DA7902     JC KEYI2
0275 14         INR D
0276 C37102     JMP KEYI
0279 7A       KEYI2:MOV A,D
027A B0         ORA B
027B C9         RET
              ;
              ;SERIAL OUTPUT ROUTINE
              ;
027C 4F       SOUT:MOV C,A
027D DB98     SOUT2:IN 98
027F E602       ANI 02
0281 CA7D02     JZ SOUT2
0284 79         MOV A,C
0285 D394       OUT 94
0287 3EF9       MVI A,F9; I/Oaddress 94 "out" & STROBE ON
0289 D398       OUT 98
028B DB98     SOUT3:IN 98
028D E602       ANI 02
028F C28B02     JNZ SOUT3
0292 3EFB       MVI A,FB; I/Oaddress 94 "out" & STROBE OFF
0294 D398       OUT 98
0296 C9         RET
              ;
              ;SERIAL INPUT ROUTINE
              ;
                ORG $02A0
              ;
02A0 DB98     SIN:IN 98
02A2 0F         RRC
02A3 DAA002     JC SIN
02A6 3EFE       MVI A,FE;BUSY
02A8 D398       OUT 98
02AA DB94       IN 94
02AC 4F         MOV C,A
02AD 3EFF       MVI A,FF;READY
02AF D398       OUT 98
02B1 79         MOV A,C
02B2 C9         RET
              ;
              ;0D0A OUT
              ;
02B3 DB98     SOUTEND:IN 98
02B5 E602       ANI 02
02B7 CAB302     JZ SOUTEND
02BA 3EFE       MVI A,FE;DATA END
02BC D398       OUT 98
02BE DB98     SOUTEND2:IN 98
02C0 E602       ANI 02
02C2 C2BE02     JNZ SOUTEND2
02C5 3EFF       MVI A,FF
02C7 D398       OUT 98
02C9 DB98     SOUTEND3:IN 98
02CB E602       ANI 02
02CD CAC902     JZ SOUTEND3
02D0 C3EA02     JMP D2
              ;
              ;CHATTERING TIMER
```

```
                ;
                  ORG $02DD
02DD 1624       D1:MVI D,24
02DF 1E0C       D1_2:MVI E,0C
02E1 1D         D1_3:DCR E
02E2 C2E102       JNZ D1_3
02E5 15           DCR D
02E6 C2DF02       JNZ D1_2
02E9 C9           RET
                ;
02EA 1648       D2:MVI D,48
02EC C3DF02       JMP D1_2
                ;
02EF 16D8       D3:MVI D,D8
02F1 C3DF02       JMP D1_2
                ;
ADDCX         =00B8  ADDSP         =0191  ADINX         =009D
ADINX2        =00A1  ADRES         =FFEE  ADRES1        =FFEF
ADSET         =0094  ADSTR         =00A4  BRENT         =0151
BRKAD         =FFF0  BRKCT         =FFF2  BSAVE         =FFE9
BSTOP         =018B  CSAVE         =FFE8  D1            =02DD
D1_2          =02DF  D1_3          =02E1  D2            =02EA
D3            =02EF  DATA          =FFEC  DATA1         =FFED
DIG           =FFF8  DIGIT         =0084  DISP          =FFF4
DSAVE         =FFE7  ESAVE         =FFE6  FSAVE         =FFEA
GOTO          =00CC  HSAVE         =FFE5  INPUT         =0223
INPUT2        =022A  INPUT3        =023D  KEY           =0247
KEYI          =0271  KEYI2         =0279  KEYIN         =0216
KFLAG         =FFF3  LDATA         =0107  LDATA2        =0124
LSAVE         =FFE4  MEMR          =00AD  MEMW          =00C2
MONSP         =FFD1  MONST         =003B  MONST2        =0041
NOBRK         =0185  NOKEY         =0242  PSAVE         =FFE0
RESRG         =01F9  RGDSP         =01A1  RGDSP2        =01A9
RST2          =FFD1  RST3          =FFD4  RST4          =FFD7
RST5          =FFDA  RST6          =FFDD  SDATA         =00D5
SDATA2        =00F1  SEGCG         =01C0  SEGCG2        =01C9
SEGD          =01E9  SHIFT         =01B5  SIN           =02A0
SOUT          =027C  SOUT2         =027D  SOUT3         =028B
SOUTEND       =02B3  SOUTEND2      =02BE  SOUTEND3      =02C9
SSAVE         =FFE2  START         =0051  TABL          =0074
USRSP         =FFC7
```