ND8080　TK80モニタプログラム操作説明書

（有）中日電工

目次

〒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
℡052-791-6254 Fax052-791-1391
E-mail thisida@alles.or.jp
Homepage http://www.tyunitidenko.x0.com/

2016．4．29 Rev. 1．0

[memo]

1章　基礎知識

1．はじめに

ここではND8080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（またはZ80）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進法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進法と同じように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のときに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進数の3は2進数では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進数の1桁を表記するのに2進数では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＋とーの2通りしかありません。＋を1 、ーを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進数の１０以上の数は2進数ではどう表現されるでしょうか。
９＋１＝10の計算は、2進数では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進数←→10進数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進数の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進数←→16進数の換算は比較的簡単に行うことができます。

10進数→16進数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進数の1234を16進数に換算します。

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

上の計算により10進数の1234は16進数では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進数→10進数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進数の4D2 を10進数に換算します。

```
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、ND8080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（またはZ80）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進数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進数で処理しているために、16進数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。
ND8080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やZ80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、16進数は3CHというように最後にHをつけて表したりしますが、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進数に換算すると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。

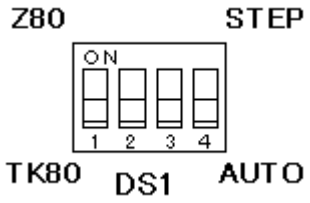
それによく使う命令は数が限られているので、それさえ覚えてしまえば、あとはうんと楽になります。（参考までに代表的なものを下にあげておきます）

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。

2章　基本操作

電源を入れる前に、ディップスイッチ（DS1）が図のように全ビットOFFになっていることを確認してください。
もしOFFになっていないビットがあれば、小型のマイナスドライバなどで、OFFにしてください。



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)

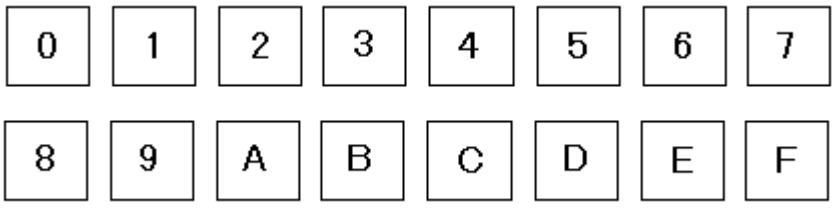
（図2－2）

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（ND8080のCPU、8080は8ビットなので、データは16進2桁です）。
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)

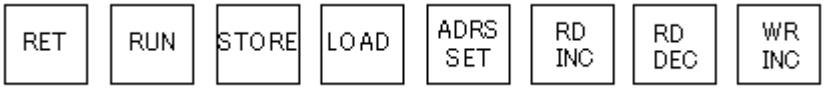
2．　キーボード

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。
ND8080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



ND8080のキーはND80Zモニタプログラムに合わせたキーシールが貼ってあるため、一部のキーの表示（およびその機能）がこの操作説明書での表示とは異なっています。

ファンクションキーのうち、[RDINC]、[RDDEC]、[WRINC]は、それぞれ[READINC]、[REDDEC]、[WRITEINC]というキーシールになっていますが、これらはわずかな表記の違いですから、問題はないと思います。

[RET]は[CONT]、[STORE]は[＊(I／O)]、[LOAD]は[REG]というように表記が全く異なっていますから、この説明書を参照するうえで、操作しにくいと思われた場合には、キーシールにエンピツなどで書き加えるようにしてください。アクリルの透明キャップははめ込んであるだけですから、小型のマイナスドライバなどで側面から上に向けて軽く起こすようにすると、取り外すことができます。

③リセットキー

キーシールは[MON]になっています。

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（READ　INCREMENT、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　DECREMENT、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　INCREMENT、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）　キーシールは［CONT］になっています

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）　キーシールは［MON］になっています

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（0000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

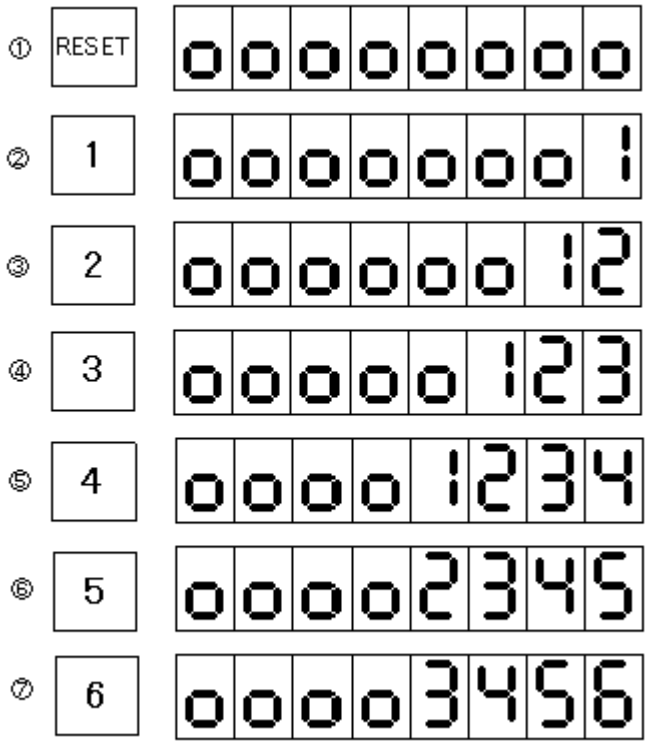
３．１　アドレス

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、IC11の62256がそのRAMです。
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83C7～83FF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（注記）。
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、83C7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83C7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。

（注記）F800～FFFFの範囲のRAMエリアはTK80モニタでは使いませんが、ND80Zモニタでは作業エリアとして使用されます。
TK80用に作成したプログラムをND80Zモニタに切り換えて使用するかもしれないことを考えると、F800～FFFFも使用しないようにした方がよいでしょう。

３．２　キー入力とLED表示

まず図（次ページ）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。

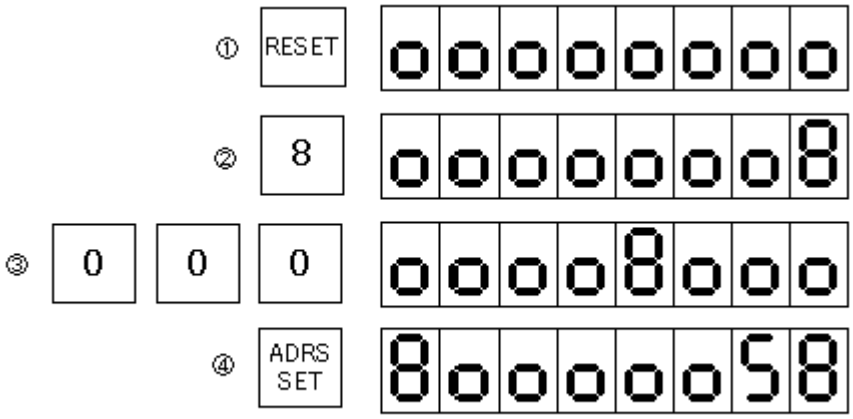
```
8000  3E00        MVI  A, 00
8002  3C          INR  A
8003  C30280      JMP  $8002
```
（リスト2－1）

　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。

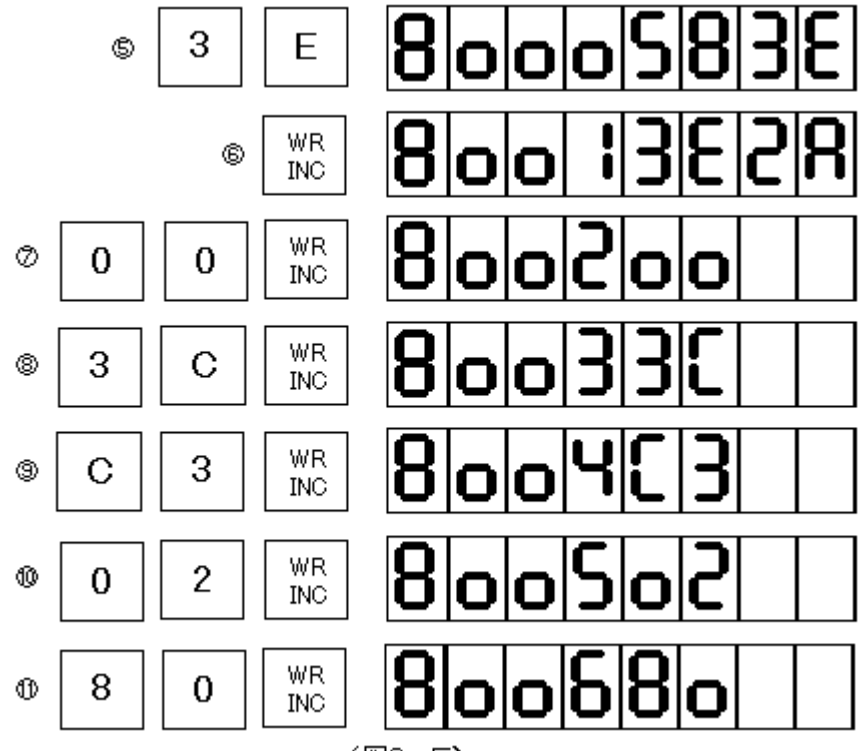
④つぎにADRSSETキーを押します。するといままで下4桁（データ表示部）にあった8000が、上4桁（アドレス表示部）に移動します。
そしてこのとき、データ表示部の下2桁が、8000番地のメモリの内容を表示しています。
ここでは58が表示されていますが、これは例であってこのときどういう数が表示されるかは状況によります。
ND8080のRAMは電池でバックアップされているため、通常はこのとき以前に書かれていたデータがそのまま読み出されます。



（図2－4）

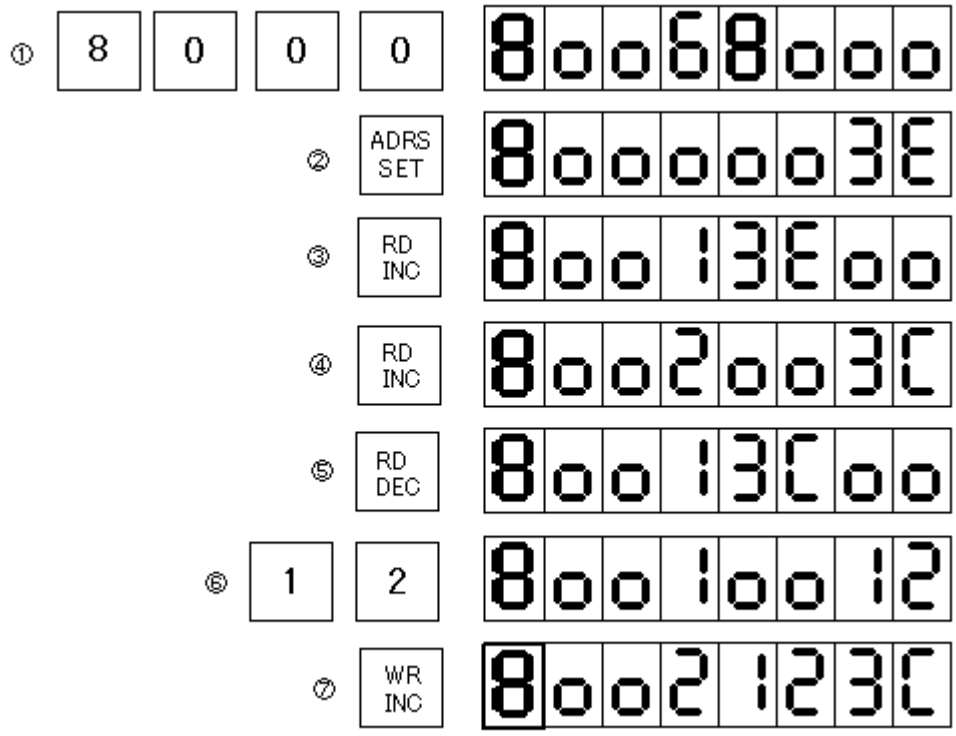
●データ（プログラム）を入れます（図2－5）

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（キーシールは[WRITEINC]になっています）。
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）

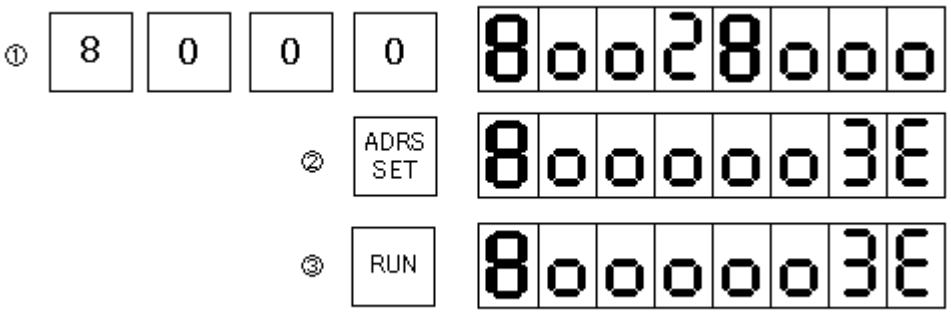
●これであとは実行させれば良いのですが、念のため正しくメモリに書き込まれているか、チェックしてみます。(図 2-6)
①8000と入力します。
②続いてADRSSETキーを押すと、データ表示部の下2桁に8000番地の内容が表示されます。
③ここでRDINCキーを押すと、アドレスの表示が8001になって、データ表示部の下2桁には8001番地の内容が表示されます。
④もう一度RDINCキーを押すとアドレス8002とそのメモリ内容が表示されます。
　このようにRDINCキーは押す度にアドレスが＋1されて、順にそのメモリ内容を見ていくことができます。
⑤これに対してRDDECキーを押すと、アドレスが逆に－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モニタには、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

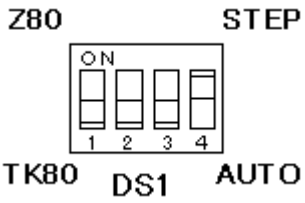
1．　はじめに

　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。
 そのようなときに、これから説明するステップ動作機能とブレイク動作機能は、強力な助けになります。

2．　ステップ動作

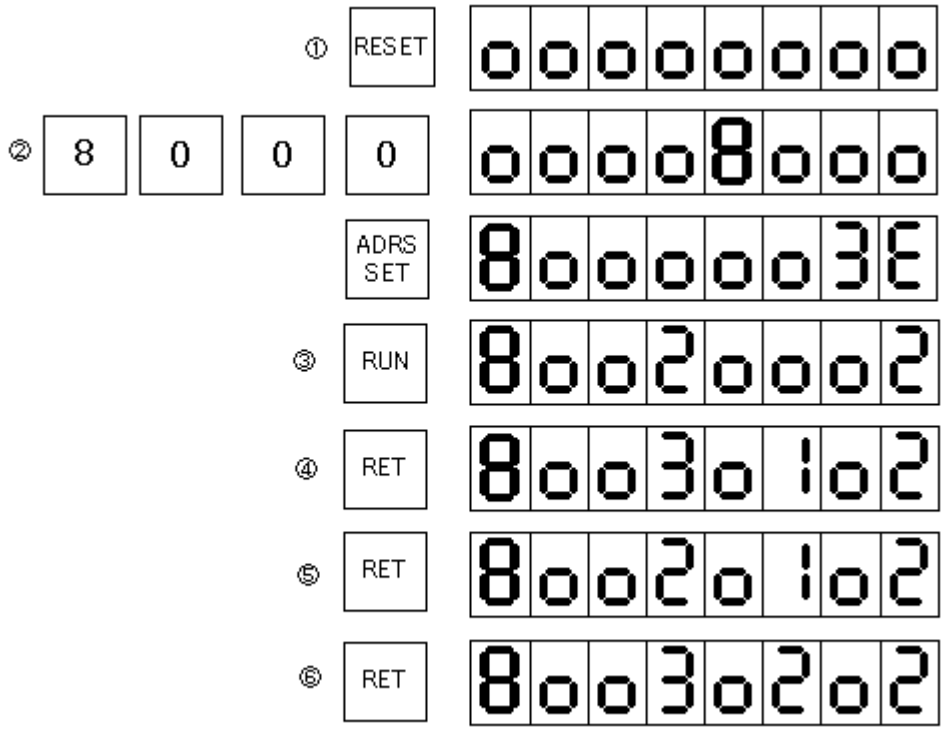
　2章の終りのところで、CPUは非常に速いスピードでプログラムを実行するので確認できない、と書きました。
　ステップ動作の機能を使うと、プログラムを1ステップずつ進めることができるので、その途中の状態を確認することができます。
 2章で作ったプログラムを実行させてみます。念の為リスト 2-1 の通りにメモリに入っていることを確認しておいて下さい（2章の終わりのところで8001番地を12に直した人は、00に戻して下さい）。

　ステップ動作をさせるためにはディップスイッチの設定が必要です。
　最初にディップスイッチDS1の№4をON（STEP側）にしておきます。



　№4がOFFになっているとステップ動作はできませんから、必ず忘れないようにしてください。

　以下に操作方法を説明します。（図 3-1 参照）
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。



（図３－１）

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（CONT）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（CONT）を押すとアドレスは8002に戻ります。
⑥このあとはRET（CONT）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ます）

　〔注意1〕（削除）
　〔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。
　TK80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。

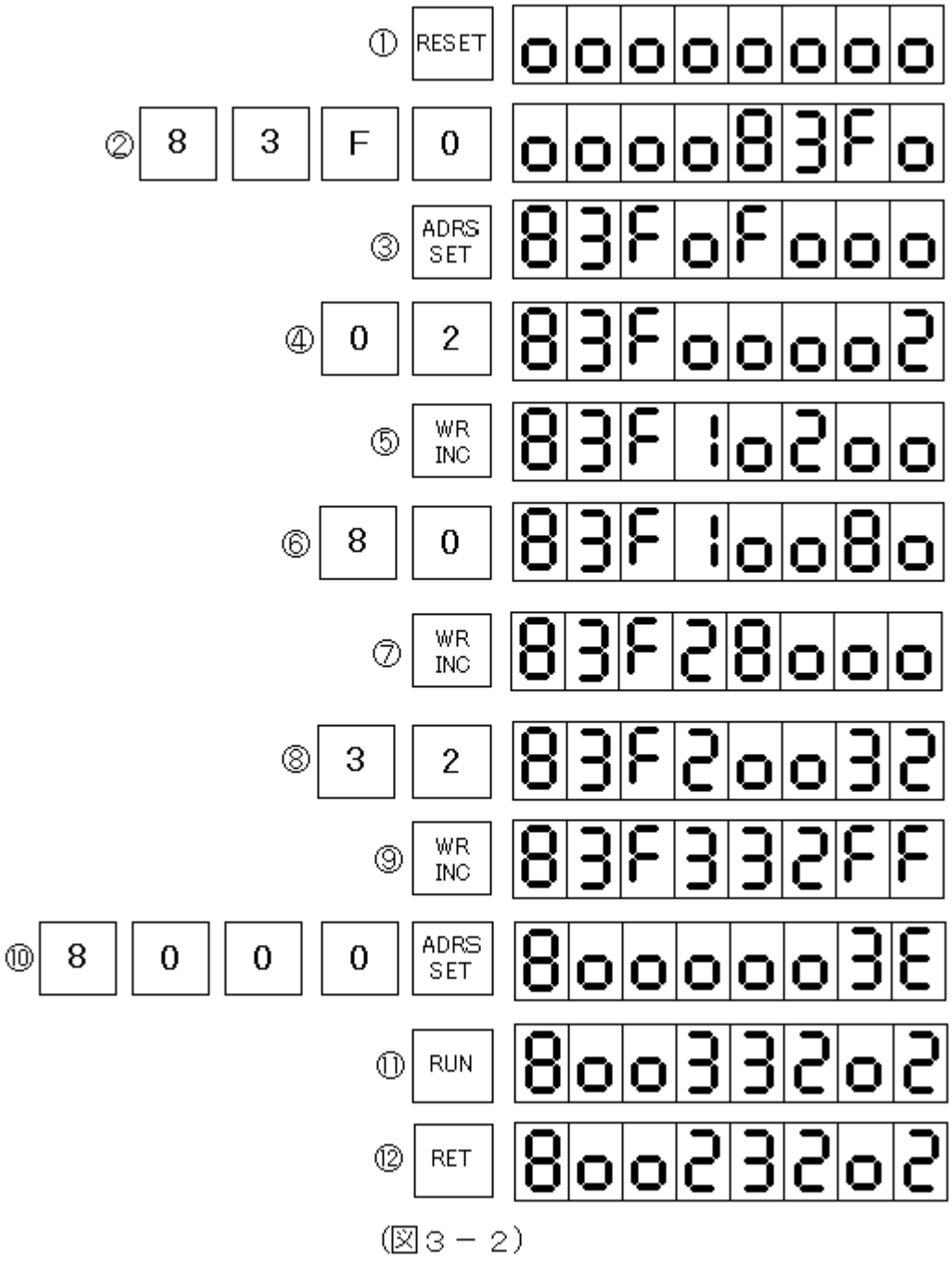
①まずリセットします。（ブレイクの時は必ずしもリセットから始めなくてもよいのですが、この方が確実です）
②③ブレイクアドレスをセットします。
　［8］［3］［F］［0］［ADRSSET］の順にキーを押してください。
　アドレス表示部に83F0と表示され、データ表示部はF000になります。
④83F0にはプレイクアドレスの下位2桁が入るのですがリセット後は00になっています。
　ここにブレイクしたいアドレスの下位2桁を入れます。今回は8002をセットしますから、その下位2桁の02を書き込みます。
　［0］［2］の順にキーを押してください。
⑤［WRINC］キーを押します。
　83F0に02が書き込まれ、アドレス表示部には83F1が表示されます。
⑥83F1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00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。
　83F1に80が書き込まれ、アドレス表示部には83F2が表示されます。
⑧83F2にはプレイクするまでの繰り返し回数を入れます。
　83F2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00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。
　83F2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DS1の4がON（STEP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は指定できますが、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）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8080のレジスタについて簡単に説明しておきます。

4．1　8080のレジスタ

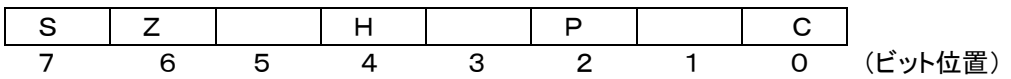
8080は内部に下記のレジスタを持っていて、これらのレジスタはプログラムの中で色々な処理に利用されます（図3－3）。

（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ます。
[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します。各ビットの意味は4．2で説明します。
[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ですが、つないで 16 ビットのレジスタ[BC]として使うこともできます。その場合には[B]が上位8ビット、[C]が下位8ビットになります。
[D][E]
B、Cと同じ。
[H][L]
B、Cと同じですが、16ビットレジスタ[HL]は、メモリアドレスを入れて、メモリ[M]を間接的に示すときにも使われます（間接アドレッシングモード）。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）としても使われます。
[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ます。スタックについては、5．3で説明します。
[PC]
プログラムカウンタ。現在実行中のアドレス(正しくは、次に実行する予定のアドレス)を管理しています。

4．2　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。



（図3－4）

各記号の意味は下の通りです（ビット1、3、5は使用されません）。
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ます。ローテイト命令でもセット、リセットされます。
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ます。

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ます。このフラグはCフラグとともに、BCD演算後のDAA命令で利用されます。

Z

ゼロ・フラグ。結果がゼロのときセットされます。

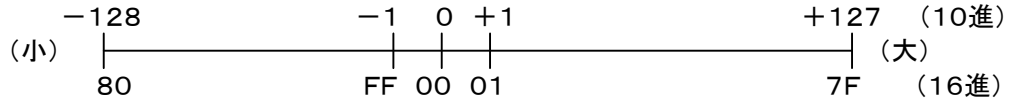
S

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ます。

[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として扱われますが、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になります（ビット7が0のときはその数は正で、ビット7が1のときは負になります）。

●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、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）。

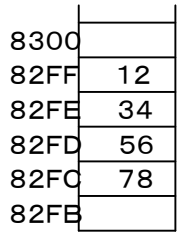
取り出すときは、入れたときと逆の順番で取り出します（POP命令を使います）。

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
```

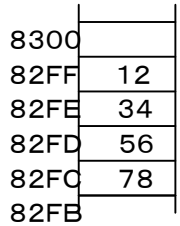
を実行すると、メモリ内容は下のようになります。



←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）。

ここではスタックの特殊な使い方として説明しました。

一般的な使い方としては、一時的に退避しておきたいレジスタを、PUSH命令でスタックに保存しておいて、あとでそれをもとのレジスタに戻すときは、PUSHとは「逆の順序」でPOPを実行します。

PUSH BC、PUSH DE の順で実行したら、それをもとに戻すときはPOP DE、POP BCのようにPUSHのときの逆の順序にします。
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、ND8080はモニタプログラムによってリセット後はSP＝83C7（ND80Zモードでは F800）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（CONT）］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（CONT）］キーを押したときには83E0、83E1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83C7が入れられます。

83EB	CPUレジスタセーブエリア　A
83EA	CPUレジスタセーブエリア　F
83E9	CPUレジスタセーブエリア　B
83E8	CPUレジスタセーブエリア　C
83E7	CPUレジスタセーブエリア　D
83E6	CPUレジスタセーブエリア　E
83E5	CPUレジスタセーブエリア　H
83E4	CPUレジスタセーブエリア　L
83E3	CPUレジスタセーブエリア　SP(H)
83E2	CPUレジスタセーブエリア　SP(L)
83E1	CPUレジスタセーブエリア　PC(H)
83E0	CPUレジスタセーブエリア　PC(L)

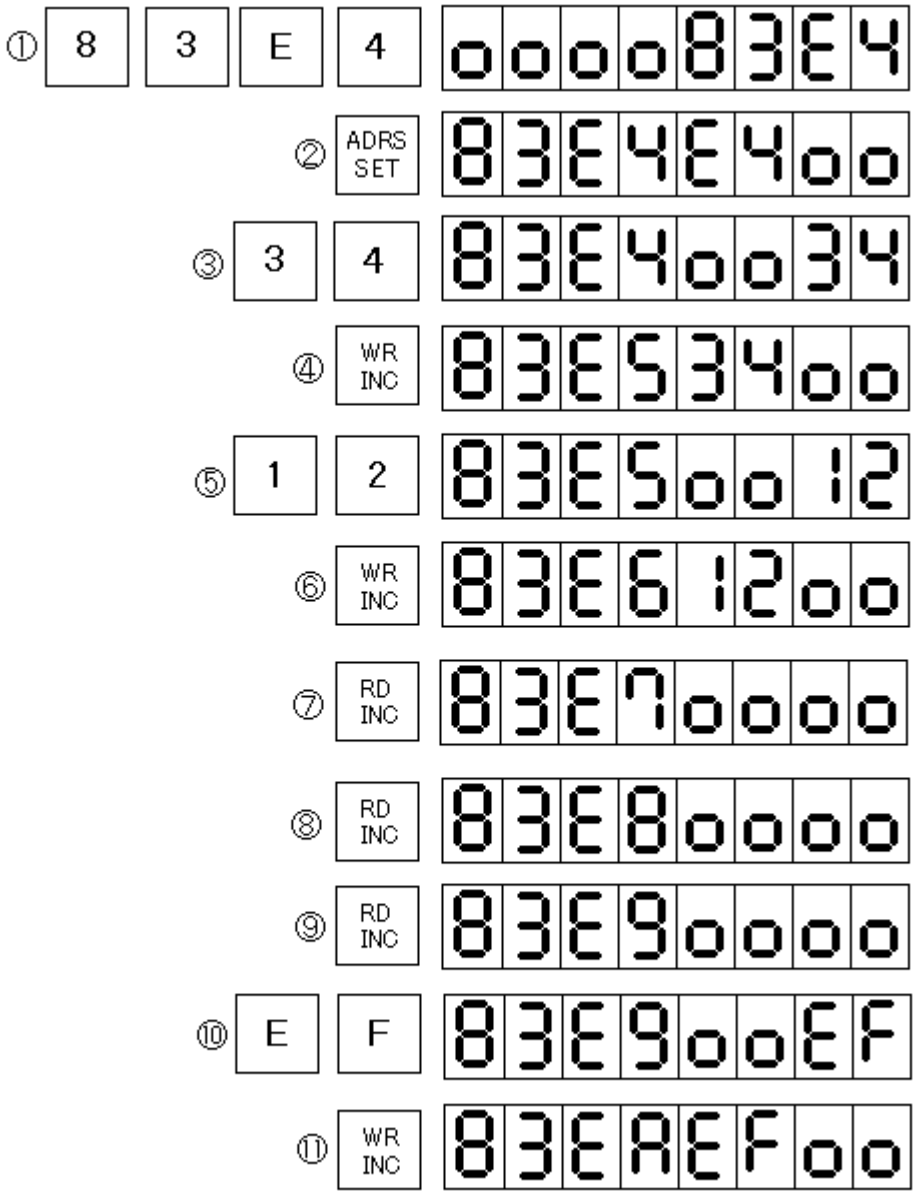
（図３－７）

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83E4、ND80Zモードでは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。
［8］［3］［E］［4］［ADRSSET］の順にキーを押してください。
アドレス表示部に83E4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83E4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。
83E4に34が書き込まれ、アドレス表示部には83E5が表示されます。
⑤83E5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83E5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。
83E5に12が書き込まれ、アドレス表示部には83E6が表示されます。
⑦⑧⑨アドレスが83E9になるまで［RDINC］キーを押します。
またはここで［8］［3］［E］［9］［ADRSSET］と操作することもできます。
⑩83E9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B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83E9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。
83E9にEFが書き込まれます。



（図３－８）

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（CONT）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キーを押すことにより、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。

[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、83E4～83EA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（83E0～83EB）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8080の21番ピン（HLDA）はH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RST0（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。
③RST1命令（コードはCF）
最後にRST1（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RST7（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。
RST7命令（FF）を実行したことによるブレイクはディップスイッチDS1の№4の状態とは無関係です。
ステップ動作やブレイク動作もRST7命令を利用していますが、それはプログラムに書かれている命令ではなくて、外部から「割込み」という方法で強制的に実行させるものなので、ハード回路（ディップスイッチの設定）が必要なのですが、プログラムの中にもともと書かれているRST7をCPUが実行する場合には、ハード回路の設定とは関係無く、いつでも0038番地のステップ処理プログラムが実行されることになるからです。

4章　プログラムのSAVE、LOAD

1.　はじめに

　ND8080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。
　TK80モニタにはプログラムやデータをSAVE、LOADする機能があります。
　USBコネクタにUSBケーブルをつないで、パソコンと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、ファイルとして保存することができます。
　逆にパソコン上で作成したマシン語のプログラムをUSB接続でND8080のRAMに送ることもできます。
　そのためには、パソコン側でもUSB（HID）READ、WRITEをするプログラムを用意する必要があります。
　そのようなプログラムを自作することも可能ですが、ND8080組立キットには附属ソフトウェアとして、ND8080をUSB（HID）で接続して、パソコンのキーボードからND8080を操作する、リモートモード＋ZB3BASICプログラムがついていますから、それを使った方が簡単です。
　しかしここでは、TK80モニタの基本的な機能として、リモートプログラムではなくて、ただのUSB（HID）送信、受信プログラムをパソコン側で実行する場合について、説明をします。

　ここではパソコン側のハードディスクにHID受信プログラム（HIDRD. EXE）、HID送信プログラム（HIDWR. EXE）がND8080附属CDROMからCOPY済みで、ND8080とWindowsパソコンがUSBケーブルで接続されているものとして説明します。
　それらのプログラムをCDROMからハードディスクにCOPYする作業については、「USB接続説明書」を参照してください。

2. プログラムのSAVEの仕方

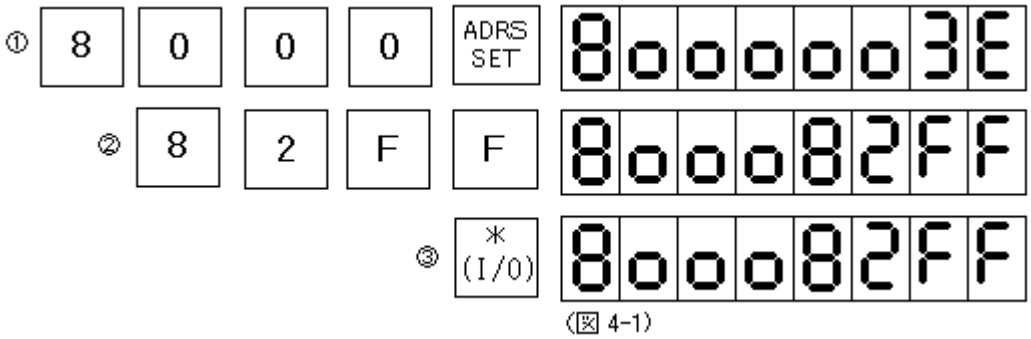
［Windowsパソコン側の操作］
　ND8080の操作をする前に、Windowsパソコン側でHID受信プログラムを実行します。
　コマンドプロンプト画面で、

　hidrd8　xxxxx. btk［Entr］

　と入力します。xxxxx. btkはプログラムを保存したい任意のファイルネームにします。
　. 以下の拡張子はbtkでなければならない、ということではありませんが、TK80モニタのSAVE／LOAD機能では、データの先頭に4バイトの開始アドレス、終了アドレスが置かれますから、一般的なバイナリファイル（拡張子bin）と区別する意味で、btkを使うことをおすすめします。btkは、binary file　for　TK80の意味です。
　画面は以下の表示になって、データ受信待ちになります。

　［入力例］
　C:¥nd8080>hidrd8　test. btk[Enter]
　送信を開始してください

［ND8080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）

①SAVE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）。

②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!!）。
③最後に[STORE]キー（キーシールは[＊（I／O）]になっています）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、Windowsパソコンのコマンドプロンプト画面には、受信が完了する以下のように表示されます。

```
s=8000,e=82ff
received data 772(=304) bytes
```

3. プログラムのLOADの仕方

SAVEの場合とは逆に、先にND8080のキー操作を行います。

[ND8080の操作]
LOADの操作は非常に簡単で、ただ[LOAD]キー（キーシールは[REG]になっています）を押すだけです。
[LOAD]を押しても7セグメントLEDの表示は変化しませんが、[RESET]キー以外は受け付けなくなります（[REG]キーはすぐに離してください）。
下はリセット後に[＊（I／O）]（[LOAD]キー）を押したときの例です。

（図4－2）

[Windowsパソコン側の操作]
hidwr8 xxxxx. btk[Entr]
と入力します。xxxxx. btkはND8080に送りたいプログラム、データのファイルです。

[入力例]
C:¥nd8080>hidwr8 test. btk[Enter]

受信が完了すると、ND8080の7セグメントLEDには、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）

Windowsパソコン側は次のように表示されます。

```
s=8000,e=82ff
send data 772(=304)bytes
```

[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]
TK80モニタプログラムのSAVEプログラムは、送信データの先頭に2バイトの送信開始アドレスと同じく2バイトの送信終了アドレスを送ります。
[注記3]
LOADは受信したアドレス情報に従って、指定されたメモリアドレスに受信したデータを書き込みます。
指定されたアドレスがROMのアドレス（0000～7FFF）であってもエラーにはなりませんが、結果としては受信しなかったのと同じことになります。
また受信したアドレス情報が、モニタプログラムのワークエリア（83xxまたはF800～FFFF）を示していた場合にはデータの受信によってモニタプログラムが暴走してしまうことがあります。
暴走したときはリセットすれば初期状態に戻ります。

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ND8080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（Z80も同じ）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。
　ND8080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、特定の目的、7segmentLED表示のON、OFFであるとか、スピーカーの出力であるとか、に使われています。
　またキーを押したとき、どのキーが押されたかをプログラムで知る仕組みも、I／Oインターフェース回路を応用しています。
　I／Oインターフェースを介して行うI／Oデータの入出力もメモリに対するのと同じように、I／Oアドレスを与えて、データバスを通じてデータの入出力を行います。
　しかし今ま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

2．1　I／Oアドレス

　ND8080のI／Oインターフェース回路のI／Oアドレスは94～9Fです（詳細は「ND8080取扱説明書」を参照してください）。
　I／Oアドレスの下位2ビットはデコードしてないため、94～97、98～9B、9C～9F、はそれぞれ同じ回路を選択することになります。

入力回路と出力回路で同じI／Oアドレスが割り付けられているものもありますが、入力はIN命令（IORD信号）、出力はOUT命令（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。
I／Oアドレス94～9Fはモニタプログラムの動作に必要な回路がつながっていますから、一部を除いて、通常はユーザーが使うことはできません。

I／Oアドレス98のビット5出力はスピーカ出力回路につながっていますから、ユーザーが自由に使うことができます。
I／Oアドレス98のビット4出力は7セグメントLEDの表示ON、OFF（DMAの許可、禁止）を制御しています。1出力で通常表示、0出力で表示OFFになります。
I／Oアドレス9Cの出力、入力はキーのアクセスに使いますが、通常はモニタプログラムのキー入力ルーチンを使いますから、特殊なプログラム以外では、ユーザーが直接使うことはありません。

[注意]I／Oアドレス98のビット5はスピーカ出力に使いますが、その他のビットは通常は1を出力してください。ビット4は7セグメント表示のためのDMA制御に使いますから、必要ならば0を出力しても構いません。
ビット0～3はPIC18F14K50を制御していますから、必ず1を出力するようにしてください。

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5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5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 1. 電子オルガンプログラム を参照してください。

4. 82C55の使い方

4. 1 82C55のアドレス

82C55には入力、出力どちらでも指定できる8ビットのI／Oポートが3組あります。これらはAポート、Bポート、Cポートとよんでおり、それぞれ異なったアドレスによって選択されます。
ND8080に実装されている82C55には次のアドレスが与えられています。

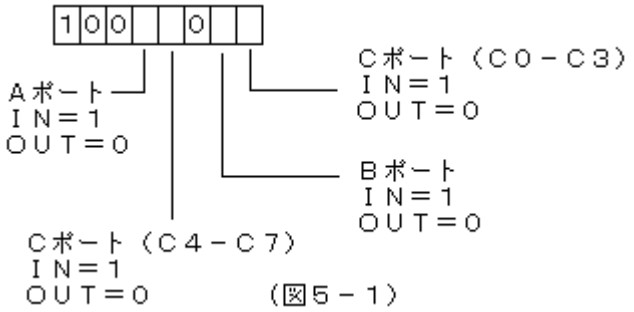
80 Aポート
81 Bポート
82 Cポート
83 コントロールワード

コントロールワードは、各ポートの向き（入力か出力か）を設定したり、Cポートに対する特殊なアクセスの場合に使います。

4. 2 82C55の各ポートの入出力指定の仕方

82C55のポートはコントロールワードの設定によって入力か出力のいずれかに設定することができますが標準的なモードでは双方向の設定はできません。電源投入後にはまず各ポートの向き（入力か出力か）を設定しなければなりません。向きの設定はコントロールワードアドレスに必要なデータを送ることによって行われます。

入出力の指定は図5－1のコントロールワードを、OUT命令で82C55のコントロールワードアドレス（83）に送ることで行われます。



（図5－1）

たとえばAポートとCポートを出力に、Bポートを入力に設定するには、I／Oアドレス83にコントロールワード82を出力します。

```
3E82   MVI A,82
D383   OUT 83
```

〔注意1〕コントロールワード出力によって、出力に設定されたポートはその時点から、全ビットが0になります。
〔注意2〕ここで説明した82C55のコントロールワードは、モード0とよばれる動作についてのものです。82C55にはこのモード0のほかに、モード1、モード2という動作モードがありますが、一般的ではないので説明を省略します。

4．3 各ポートに対するデータ入出力の方法

コントロールワードによって入力、出力の指定が行われたあとは、どのような時点でもその指定にしたがってデータの入出力ができます。（入力ポートならIN命令、出力ポートならOUT命令を使います）

（1）データの出力
Aレジスタのデータが OUT命令によって、指定したポートから出力されます。
命令は下のようにコーディングします。

```
D380  OUT 80  ……Aレジスタの内容をAポートから出力する
```

〔注記〕出力に設定されたポート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か、または入出力の設定をし直すまではもとのデータの出力が維持されます。

（2）データの入力
IN 命令によって、指定したポートからのデータが 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。

```
DB81  IN 81   ……Bポートに入力されたデータがAレジスタに入る。
```

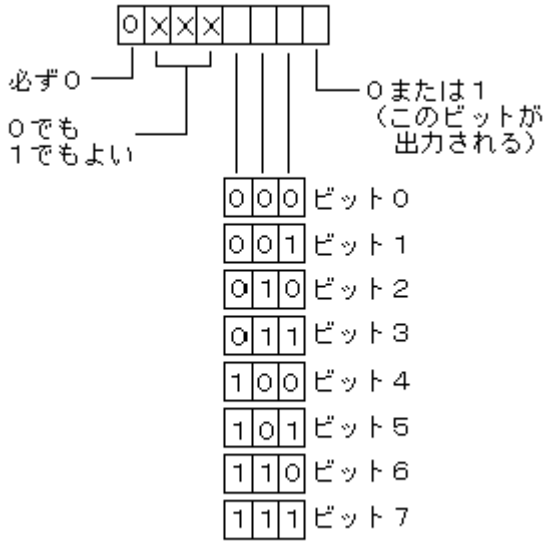
〔注意〕入力データはラッチされません。

4．4 Cポートだけに許される特殊なデータ出力方法

Cポートも4．3（1）で説明した使い方でデータ出力ができますが、Cポートに限って特殊なデータ出力が可能です。
4．3（1）のOUT命令では、8ビットのデータが出力されますが、作業によっては他の出力は変化させずに特定の1ビットだけ出力を変えたい、という場合があります。
AポートやBポートはソフトウェアで工夫するしかないのですが、Cポートは図5－2（次ページ）のコントロールワードを、コントロールワードアドレス（Cポートアドレスではありません）に出力することで、任意の1ビットだけ出力を変化させることができます。

例えば PC5 から0を出力したければ、00001010 つまり0Aを83（82ではありません）に出力します。
下のようにコーディングします。

```
3E0A    MVI A,0A
D383    OUT 83
```



（図５－２）

〔注意〕この動作は出力に設定されているビットのみに当てはまります。

　Cポートは上位4ビットと下位4ビットで入出力を別個に設定できますが、例えば上位が出力で下位が入力に設定されている場合は、上の動作は上位4ビットに対しては有効ですが、下位4ビットに対しては無視されます。

6章　応用プログラム

1.　電子オルガンプログラム

ND8080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。

1.1 プログラムリスト

```
2016/4/29  15:46  n8sound.txt
END=804E
              ;;;SOUND5.TXT
              ;;; sound for ND80Z3 clock=6MHz
              ;;; 10.3.18 10.6.15
              ;16/4/17 for ND8080 clock=2MHz
              ;;;
              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 C5       PUSH B
8012 213780   LXI H,SNDTBL
8015 85       ADD L
8016 6F       MOV L,A
8017 46       MOV B,M
8018 1E1A     MVI E,1A
801A 50       SNDS1:MOV D,B
801B 3EFF     MVI A,FF;sp out=H
801D D398     OUT 98
801F 7F       SNDS2:MOV A,A;5---------
8020 15       DCR D;5         | 5+5+10=20  20/2=10microsec
8021 C21F80   JNZ SNDS2;10----
8024 50       MOV D,B
8025 3EDF     MVI A,DF;sp out=L
8027 D398     OUT 98
8029 7F       SNDS3:MOV A,A;5---------
802A 15       DCR D;5         | 5+5+10=20  20/2=10microsec
802B C22980   JNZ SNDS3;10----
802E 1D       DCR E
802F C21A80   JNZ SNDS1
8032 C1       POP B
8033 D1       POP D
8034 E1       POP H
8035 F1       POP PSW
8036 C9       RET
              ;
              ; SOUND TABLE
```

```
8037 7F        SNDTBL:DB 7F;so4
8038 77         DB 77;so#4
8039 71         DB 71;ra4
803A 6A         DB 6A;ra#4
803B 5F         DB 5F;do5
803C 59         DB 59;do#5
803D 54         DB 54;re5
803E 4F         DB 4F;re#5
803F 47         DB 47;fa5
8040 43         DB 43;fa#5
8041 3F         DB 3F;so5
8042 3B         DB 3B;so#5
8043 35         DB 35;ra#5
8044 32         DB 32;si5
8045 2F         DB 2F;do6
8046 2C         DB 2C;do#6
8047 25         DB 25;mi6
8048 27         DB 27;re#6
8049 2A         DB 2A;re6
804A 4B         DB 4B;mi5
804B 38         DB 38;ra5
804C 64         DB 64;si4
804D 23         DB 23;fa6
804E 21         DB 21;fa#6
                ;END
KEY            =0247  SND           =8000  SNDS1         =801A
SNDS2          =801F  SNDS3         =8029  SNDSB         =800E
SNDTBL         =8037
```

1．2 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。

1．3 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。

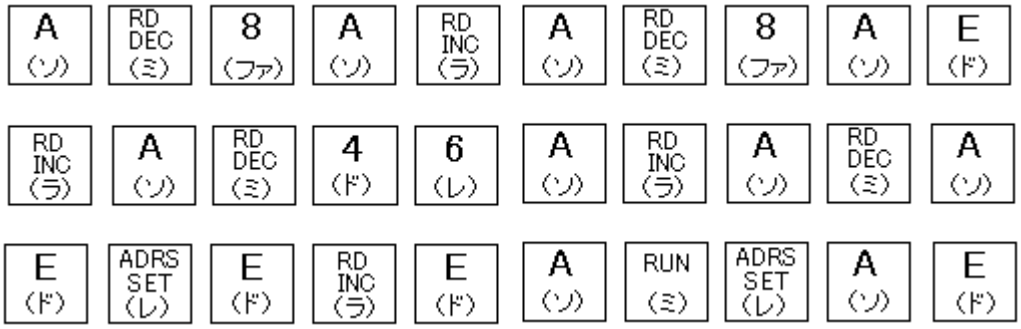
なおモニタサブルーチン0247は、キーの状態を一回だけスキャンしてチェックしどのキーも押されていなければAレジスタにFFを入れてリターンします。

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

終了するときは[MON]キーを押してリセットします。

［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　RS232C通信

ND8080はPIC18F14K50を使って外部機器やパソコンにRS232Cでメモリ内のデータを送信したり、外部機器やパソコンからデータを受信してRAMに書き込むことができます。
送信に限っては、RAMだけではなくて、モニタROMのデータを送ることもできます。
ND8080のRS232C機能の詳細については、ND8080取扱説明書（最初にお読みください）を参照してください。
ここでは、ND8080のモニタROMに書き込んである、RS232C送信・受信プログラムの使い方を説明します。

［注記］
送信データはASCIIコードに変換されません。
メモリに書かれている通りの8ビットデータとして送信します。
また先頭に送信開始、終了アドレスは付加されません。
受信についても同様です。

1．RS232C送信プログラム
　　開始アドレス　034E
　　使用レジスタ　A、F、B、C、D、E、H、L

RS232C送信プログラムの実際の開始アドレスは0347ですが、そこから開始すると、送信を開始するアドレスと終了するアドレスが7セグメントLEDのアドレス表示部とデータ表示部から読み込まれてしまいます。
この送信ルーチンは、もともとND80Zモニタのためのものなので、そのような動作をするように作られています。
ND80Zモニタではキーを［＊（I／O）］［2］と操作することで、この送信プログラムが実行できますが、TK80モニタではそのようにできません。
TK80モニタでは、以下のようなプログラムを、現在使用していないRAMの空きエリアに書いた上で、そのプログラムを実行することで、RS232C送信を行うことができます。
8000～8FFFのデータ（プログラムでもよい）をRS232Cで送信するためのプログラムを、E000から書いた例です。

```
E000  210080    LXI  H, ＄8000
E003  11FF8F    LXI  D, ＄8FFF
E006  C34E03    JMP  ＄034E
```

このように書いたあと、［E］［0］［0］［0］［ADRSSET］［RUN］と操作すると、RS232C送信が行われます。
送信中は7セグメントLEDのデータ表示部に現在送信中のアドレスがモニタ表示されます。
アドレス表示部は変わりません（上のプログラム例ではE000が表示されたままです）。
送信が完了すると、データ表示部に終了アドレスが表示された状態で静止しますが、この時点でRS232C送信プログラムは終了してTK80モニタのエントリルーチンに戻っていますから、普通にキー操作をすることができます。

2．RS232C受信プログラム
　　開始アドレス　036C
　　使用レジスタ　A、F、B、C、D、E、H、L

RS232C受信ルーチンの実際の開始アドレスは0369ですが、そこから開始すると、受信を開始するアドレスが7セグメントLEDのアドレス表示部から読み込まれてしまいます。
この受信ルーチンは、もともとND80Zモニタのためのものなので、そのような動作をするように作られています。
ND80Zモニタではキーを［＊（I／O）］［3］と操作することで、この受信プログラムが実行できますが、TK80モニタではそのようにできません。
TK80モニタでは、以下のようなプログラムを、現在使用していないRAMの空きエリアに書いた上で、そのプログラムを実行することで、RS232C受信を行うことができます。
RS232Cで受信するデータをRAMのアドレス8000から格納したいときのプログラムを、E010から書いた例です。

```
E010  210080    LXI  H, ＄8000
E013  C36C03    JMP  ＄036C
```

このように書いたあと、［E］［0］［1］［0］［ADRSSET］［RUN］と操作すると、RS232C受信が行われます。
受信中は7セグメントLEDのデータ表示部には現在受信中のアドレスがモニタ表示されます。
アドレス表示部は変わりません（上のプログラム例ではE010が表示されたままです）。

受信が終了（送信データが途切れる）しても、そのまま受信待ちで停止しています。ふたたび送信が開始されると、受信動作が再開されます。

送信プログラムと違い、受信の場合には受信データが完了したことを知ることができません。

RS232Cの送信、受信では、［STORE］［LOAD］キーを使ってUSB通信を行うときのように、データの先頭に開始アドレスと終了アドレスを置くことはしません。

実際に送信、受信するデータのみを送受信しますから、受信プログラムでは送信が完了したのか、それとも送信側の都合で一時的に送信データが途切れているのかを判断することができません。

もし送信が完了していて、これ以上受信を待つ必要がない状態になったら、リセットすることで、通常のモニタ操作に戻ることができます。

8章　モニタサブルーチン

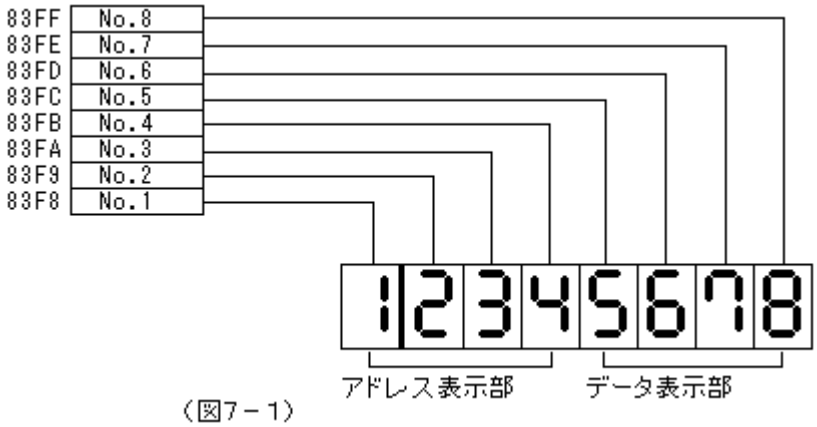
1.　はじめに

　TK80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

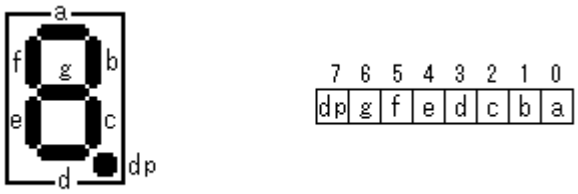
2．1　セグメント表示バッファとLED表示の関係

　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（83F8～83FF）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7－1）

　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7－2）

　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

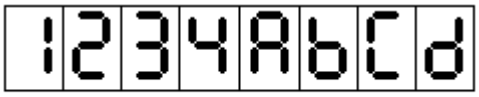
　　開始アドレス　01C0
　　使用レジスタ　A、F、B、C、D、E、H、L

　表示用データレジスタ（83F4～83F7）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（83F8～83FF）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7-3のように表示されます。

表示用データレジスタ　No.4　83F7=CD
表示用データレジスタ　No.3　83F6=AB
表示用データレジスタ　No.2　83F5=34
表示用データレジスタ　No.1　83F4=12



（図7－3）

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（83EE～83EF）、データレジスタ（83EC～83ED）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)83EF　→　表示用データレジスタ　No.1　83F4
アドレスレジスタ(L)83EE　→　表示用データレジスタ　No.2　83F5
データレジスタ(H)83ED　→　表示用データレジスタ　No.3　83F6
データレジスタ(L)83EC　→　表示用データレジスタ　No.4　83F7

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

下記のサブルーチンをCALLすると、それぞれ()に表記した時間だけウェイトしてからリターンします。
パルス出力や少しだけ時間待ちをしたいときなどに利用することができます。
TK80モニタプログラムのキー入力サブルーチン(INPUT、アドレス0223)でもタイマールーチン②をチャタリング回避のための時間待ちに使っています。
なおオリジナルのTK80のタイマールーチンとアドレスは同じですが、CPU、クロックの違いのため、ウェイトする時間はわずかですが違っています。
またND8080は(TK80も同じ)は約1msecに1回7セグメントLEDの表示のためにDMAアクセスが実行され、20μsec程度プログラムの実行が停止します。
その分だけ実際のウェイト時間は長くなります。下記()の時間はDMAを禁止したときのウェイト時間です。

4.1 タイマールーチン①(4.497ms)
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(8.992ms)
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(26.968ms)
開始アドレス 02EF
使用レジスタ F、D、E

5. DMA(7セグメントLEDの表示)の禁止

サブルーチンではありませんが、タイマールーチンなど、正確なプログラムの実行が求められるときのために、DMAを禁止する方法について、ここで説明をします。

4. タイマーで説明したように、ND8080で普通にプログラムを実行しているときは、7セグメントLEDの表示のために、約1msecに1回、DMAアクセスが行われ、20μsec程度CPUの命令実行動作が停止します。
何かの測定のためなどで正確な待ち時間を命令の実行クロック数から算出してプログラムしたものを実行する場合には、途中でDMAが行われたりすると、その分実行時間が余計にかかってしまうために、正確な測定ができなくなります。
そのようなときのために、プログラムで必要な期間、DMAを禁止することができます。
DMAを禁止すると、7セグメントLED表示の表示が行われなくなるため、LEDは消灯します。
プログラム内でDMA禁止を解除するか、RESETすると通常のように7セグメントLEDの表示が行われるようになります。

DMAを禁止するには、I/OアドレスC98のビット4に0を出力します。

```
3EEF    MVI  A, EF
D398    OUT  98
```

DMAの禁止を解除するには、I/OアドレスC98のビット4に1を出力します。

```
3EFF    MVI  A, FF
D398    OUT  98
```

[注記]I/OアドレスC98のビット0～3はPIC18F14K50との間でデータの受け渡しをするために使われています。ビット0～3に0を出力しないようにしてください。
またビット5はスピーカ出力に使われています。スピーカを使わないときは0でも1でも構いません。
ビット6、7は使われていませんから、0でも1でも構いません。

9章　モニタプログラムリスト

```
2016/4/22  11:48  n8mon0d.txt
END=03D7
                ;FOR 8080 16/1/5 FROM NDMONOM
                ;
                ;;; TK80 MONITOR PROGRAM FOR ND80Z(WORK AREAR 83xx)
                ; 09/5/28 09/6/1 6/3 6/5
                ; 10/2/25 3/5 3/6 3/19 3/22 3/23
                ; 10/4/1 DIG=$83F8
                ;10/5/9 pic interface 4bits
                ;5/16 for pic18f14k50
                ;6/15 for nd80z3 6/16
                ;ndmon0j 10/8/21
                ;ndmon0k,L 10/8/23
                ;NDMONOM 8/24
                ;
                ;1/6 KEYIN D3="0"
                ;4/15 D2 for RND
                ;4/17 D1_2
                ;4/18 D1
                ;4/22 D2 for RND
                ;
                ; 83FF-83F8 7SEGMENT DISPLAY ADDRESS
                ;
                DIG=$83F8
                ;
                R=$FFD2
                RST7=$FFCC
                RST6=$FFC9
                RST5=$FFC6
                RST4=$FFC3
                RST3=$FFC0
                RST2=$FFBD
                RST1=$FFBA
                SINERMK=$FFB8
                ;
                DISP=$83F4
                KFLAG=$83F3
                BRKCT=$83F2
                BRKAD=$83F0
                ADRES1=$83EF
                ADRES=$83EE
                DATA1=$83ED
                DATA=$83EC
                FSAVE=$83EA
                BSAVE=$83E9
                CSAVE=$83E8
                DSAVE=$83E7
                ESAVE=$83E6
                HSAVE=$83E5
                LSAVE=$83E4
                SSAVE=$83E2
                PSAVE=$83E0
                ;
                ;       RST7=$0038
                ;       RST6=$83DD
```

```
                ;       RST5=$83DA
                ;       RST4=$83D7
                ;       RST3=$83D4
                ;       RST2=$83D1
                ;
                MONSP=$83D1
                USRSP=$83C7
                ;
                NDZMON=$0800
                LDIR=$0824
                LDDR=$0827
                SBCHLBC=$082A
                ;
                ;
                ORG $0000
                ;
0000 C30008     JMP NDZMON
                ;
                ORG $0008
0008 C3BAFF     JMP RST1
000B C32408     JMP LDIR
                ;
                ORG $0010
0010 C3BDFF     JMP RST2
0013 C32A08     JMP SBCHLBC
                ;
                ORG $0018
0018 C3C0FF     JMP RST3
001B C32708     JMP LDDR
                ;
                ORG $0020
0020 C3C3FF     JMP RST4
                ;
                ORG $0028
0028 C3C6FF     JMP RST5
                ;
                ORG $0030
0030 C3C9FF     JMP RST6
                ;
                ORG $0038
0038 C3CCFF     JMP RST7
                ;
                ; INITIALIZE ROUTINE
                ;
003B 3EFF       MONST:MVI A,FF
003D D398       OUT 98
003F 21EC83     LXI H,DATA
0042 060C       MVI B,0C
0044 AF         XRA A
0045 77         MONST2:MOV M,A
0046 23         INX H
0047 05         DCR B
0048 C24500     JNZ MONST2
004B 21C783     LXI H,USRSP
004E 22E283     SHLD SSAVE
                ;
                ; MONITOR START
                ;
```

```
0051 3EFF     START:MVI A,FF
0053 D398     OUT 98
0055 31D183   LXI SP,MONSP
0058 CDC001   CALL SEGCG
005B CD1602   START2: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 3AEC83   LDA DATA
008A B0       ORA B
008B 32EC83   STA DATA
008E CDA101   CALL RGDSP
0091 C35100   JMP START
              ;
              ; ADDRESS SET
              ;
0094 2AEC83   ADSET:LHLD DATA
0097 22EE83   SHLD ADRES
009A C3A100   JMP ADINX2
              ;
              ; MEMORY READ & ADDRESS INCREMENT
              ;
009D 2AEE83   ADINX:LHLD ADRES
00A0 23       INX H
00A1 CDAD00   ADINX2:CALL MEMR
00A4 22EE83   ADSTR:SHLD ADRES
00A7 CDA101   CALL RGDSP
00AA C35100   JMP START
              ;
00AD 3AEC83   MEMR:LDA DATA
00B0 32ED83   STA DATA1
00B3 7E       MOV A,M
00B4 32EC83   STA DATA
00B7 C9       RET
              ;
```

```
                ; MEMORY READ & ADDRESS DECREMENT
                ;
00B8 2AEE83     ADDCX:LHLD ADRES
00BB 2B         DCX H
00BC CDAD00     CALL MEMR
00BF C3A400     JMP ADSTR
                ;
                ; MEMORY WRITE
                ;
00C2 2AEE83     MEMW:LHLD ADRES
00C5 3AEC83     LDA DATA
00C8 77         MOV M,A
00C9 C39D00     JMP ADINX
                ;
                ; MONITOR TO USER CONTROL ROUTINE
                ;
00CC 2AEE83     GOTO:LHLD ADRES
00CF 22E083     SHLD PSAVE
00D2 C3F901     JMP RESRG
                ;
                ; STORE DATA
                ;
00D5 06F7       SDATA:MVI B,F7
00D7 2AEC83     LHLD DATA
00DA EB         XCHG
00DB 2AEE83     LHLD ADRES
00DE 78         MOV A,B
00DF D398       OUT 98
00E1 7C         MOV A,H
00E2 CD7C02     CALL SOUT
00E5 7D         MOV A,L
00E6 CD7C02     CALL SOUT
00E9 7A         MOV A,D
00EA CD7C02     CALL SOUT
00ED 7B         MOV A,E
00EE CD7C02     CALL SOUT
00F1 2B         DCX H
00F2 23         SDATA2:INX H
00F3 7E         MOV A,M
00F4 CD7C02     CALL SOUT
00F7 CD0103     CALL HDCMP
00FA C2F200     JNZ SDATA2
00FD C35100     JMP START
0100 00         NOP
0101 00         NOP
0102 00         NOP
0103 00         NOP
0104 00         NOP
0105 00         NOP
0106 00         NOP
                ;
                ;LOAD DATA
                ;
0107 06FF       LDATA:MVI B,FF
0109 CDA002     CALL SIN
010C 67         MOV H,A
010D CDA002     CALL SIN
0110 6F         MOV L,A
```

```
0111 CDA002    CALL SIN
0114 57        MOV D,A
0115 CDA002    CALL SIN
0118 5F        MOV E,A
0119 22EE83    SHLD ADRES
011C EB        XCHG
011D 22EC83    SHLD DATA
0120 EB        XCHG
0121 2B        DCX H
0122 23        LDATA2:INX H
0123 CDA002    CALL SIN
0126 77        MOV M,A
0127 CD0103    CALL HDCMP
012A C22201    JNZ LDATA2
012D CDA101    CALL RGDSP
0130 C35100    JMP START
               ;
               ; BREAK ENTRY
               ; BREAK & ONE STEP OPERATION
               ;
               ORG $0151
               ;
0151 E3        BRENT:XTHL
0152 22E083    SHLD PSAVE
0155 F5        PUSH PSW
0156 210400    LXI H,$0004
0159 39        DAD SP
015A F1        POP PSW
015B 22E283    SHLD SSAVE
015E E1        POP H
015F 31EC83    LXI SP,DATA
0162 F5        PUSH PSW
0163 C5        PUSH B
0164 D5        PUSH D
0165 E5        PUSH H
0166 31D183    LXI SP,MONSP
0169 3AF283    LDA BRKCT
016C A7        ANA A
016D CA8B01    JZ BSTOP
0170 2AF083    LHLD BRKAD
0173 EB        XCHG
0174 2AE083    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 21F283    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 2AEA83    ADDSP:LHLD FSAVE
0194 22EC83    SHLD DATA
0197 2AE083    LHLD PSAVE
019A 22EE83    SHLD ADRES
```

```
019D CDA101    CALL RGDSP
01A0 C9        RET
               ;
               ;
               ;;; SUBROUTINE
               ;
01A1 21EF83    RGDSP:LXI H,ADRES1
01A4 11F483    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 2AEC83    SHIFT:LHLD DATA
01B8 29        DAD H
01B9 29        DAD H
01BA 29        DAD H
01BB 29        DAD H
01BC 22EC83    SHLD DATA
01BF C9        RET
               ;
               ; SEGMENT CONVERT SUB
               ;
01C0 21F483    SEGCG:LXI H,DISP
01C3 11F883    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 2AE283    RESRG:LHLD SSAVE
01FC F9        SPHL
01FD 2AE083    LHLD PSAVE
0200 E5        PUSH H
0201 2AE483    LHLD LSAVE
0204 E5        PUSH H
0205 2AEA83    LHLD FSAVE
0208 E5        PUSH H
0209 2AE883    LHLD CSAVE
020C 4D        MOV C,L
020D 44        MOV B,H
020E 2AE683    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 3AF383    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 3AF383   LDA KFLAG
0238 A7       ANA A
0239 C22A02   JNZ INPUT2
023C 3D       DCR A
023D 32F383   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 3EF6     MVI A,F6
024C D39C     OUT 9C
024E DB9C     IN 9C
0250 EEFF     XRI FF
0252 C27102   JNZ KEYI
0255 0608     MVI B,08
0257 3EF5     MVI A,F5
0259 D39C     OUT 9C
025B DB9C     IN 9C
025D EEFF     XRI FF
025F C27102   JNZ KEYI
0262 0610     MVI B,10
0264 3EF3     MVI A,F3
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 DB94     SOUT2:IN 94
027F E640     ANI 40
0281 CA7D02   JZ SOUT2
0284 CD2603   CALL SOUTSB
0287 78       MOV A,B; I/Oaddress 94 "out" & STROBE ON
0288 E6FD     ANI FD;bit1=0
028A D398     OUT 98
028C DB94     SOUT3:IN 94
028E E640     ANI 40
0290 C28C02   JNZ SOUT3
0293 78       MOV A,B; I/Oaddress 94 "out" & STROBE OFF
```

```
0294 D398     OUT 98
0296 C9       RET
              ;
              ;SERIAL INPUT ROUTINE
              ;
              ORG $02A0
              ;
02A0 78       SIN:MOV A,B
02A1 D398     OUT 98
02A3 CD8D03   SIN2:CALL SINSB
02A6 CAA302   JZ SIN2
02A9 79       MOV A,C
02AA C9       RET
              ;
              ;CHATTERING TIMER
              ;
              ;D1=4.5112ms->4.6841ms
              ;D2=9.0176ms->9.3648ms
              ;D3=27.0176ms->28.0679ms
              ;
              ORG $02DD
02DD 1624     D1:MVI D,24;=36 ck=7 (7+266*36+10)/2.048=9593/2.048=4684.08microsec
02DF 1E10     D1_2:MVI E,10;=16 ck=7  7+15*16+15=266
02E1 1D       D1_3:DCR E;       ck=5
02E2 C2E102   JNZ D1_3; ck=10
02E5 15       DCR D;    ck=5
02E6 C2DF02   JNZ D1_2; ck=10
02E9 C9       RET;      ck=10
02EA 1648     D2:MVI D,48;=72  (7+266*72+10+10)/2.048=19179/2.048=9364.75microsec
02EC C3CF03   JMP D2_2
02EF 16D8     D3:MVI D,D8;=216 (7+266*216+20)/2.048=57483/2.048=28067.87microsec
02F1 C3DF02   JMP D1_2
              ;;;
02F4 22EC83   HLDTDP:SHLD DATA
02F7 C5       PSHRGDP:PUSH B
02F8 D5       PUSH D
02F9 E5       PUSH H
02FA CDA101   CALL RGDSP
02FD E1       POP H
02FE D1       POP D
02FF C1       POP B
0300 C9       RET
              ;
0301 7D       HDCMP:MOV A,L
0302 BB       CMP E
0303 C0       RNZ
0304 7C       MOV A,H
0305 BA       CMP D
0306 C9       RET
              ;
0307 32B8FF   LDERR:STA SINERMK
030A 211E03   LXI H,ERRT
030D CD1303   CALL SEGDP
0310 C35B00   JMP START2
              ;
0313 11F883   SEGDP:LXI D,DIG
0316 7E       SEGDP2:MOV A,M
0317 12       STAX D
```

```
0318 23       INX H
0319 1C       INR E
031A C21603   JNZ SEGDP2
031D C9       RET
              ;
031E 79       ERRT:DB 79;E
031F 50       DB 50;r
0320 50       DB 50;r
0321 5C       DB 5C;o
0322 50       DB 50;r
0323 80       DB 80
0324 80       DB 80
0325 80       DB 80
              ;
0326 79       SOUTSB:MOV A,C
0327 D394     OUT 94
0329 78       MOV A,B
032A E6FD     ANI FD
032C D398     OUT 98
032E DB94     SOUTSB2:IN 94
0330 E640     ANI 40
0332 C22E03   JNZ SOUTSB2
0335 78       MOV A,B
0336 D398     OUT 98
0338 DB94     SOUTSB3:IN 94
033A E640     ANI 40
033C CA3803   JZ SOUTSB3
033F 79       MOV A,C
0340 0F       RRC
0341 0F       RRC
0342 0F       RRC
0343 0F       RRC
0344 D394     OUT 94
0346 C9       RET
              ;
              ; RS232C SAVE & LOAD
0347 2AEC83   RSAVE:LHLD DATA
034A EB       XCHG
034B 2AEE83   LHLD ADRES
034E 06F3     MVI B,F3
0350 2B       DCX H
0351 23       RSAVE2:INX H
0352 7E       MOV A,M
0353 CD7C02   CALL SOUT
0356 3EFF     MVI A,FF
0358 D398     OUT 98
035A 22EC83   SHLD DATA
035D CDF702   CALL PSHRGDP
0360 CD0103   CALL HDCMP
0363 C25103   JNZ RSAVE2
0366 C35B00   JMP START2
              ;
0369 2AEE83   RLOAD:LHLD ADRES
036C 06FB     MVI B,FB
036E 2B       DCX H
036F 23       RLOAD2:INX H
0370 CD7D03   CALL RSIN
0373 77       MOV M,A
```

```
0374 22EC83    SHLD DATA
0377 CDF702    CALL PSHRGDP
037A C36F03    JMP RLOAD2
               ;
               ;
037D 78        RSIN:MOV A,B
037E D398      OUT 98
0380 CD8D03    RSIN1:CALL SINSB
0383 79        MOV A,C
0384 C0        RNZ
0385 FEFF      CPI FF
0387 C20703    JNZ LDERR
038A C38003    JMP RSIN1
               ;
038D DB94      SINSB:IN 94
038F E620      ANI 20
0391 CA8D03    JZ SINSB
0394 78        MOV A,B;BUSY
0395 E6FE      ANI FE;bit0=0
0397 D398      OUT 98
0399 DB94      SINSB2:IN 94
039B E620      ANI 20
039D C29903    JNZ SINSB2
03A0 DB94      IN 94
03A2 E610      ANI 10
03A4 F5        PUSH PSW
03A5 DB94      IN 94
03A7 E60F      ANI 0F
03A9 4F        MOV C,A
03AA 78        MOV A,B;READY
03AB D398      OUT 98
03AD DB94      SINSB3:IN 94
03AF E620      ANI 20
03B1 CAAD03    JZ SINSB3
03B4 78        MOV A,B
03B5 E6FE      ANI FE
03B7 D398      OUT 98
03B9 DB94      SINSB4:IN 94
03BB E620      ANI 20
03BD C2B903    JNZ SINSB4
03C0 DB94      IN 94
03C2 07        RLC
03C3 07        RLC
03C4 07        RLC
03C5 07        RLC
03C6 E6F0      ANI F0
03C8 B1        ORA C
03C9 4F        MOV C,A
03CA 78        MOV A,B
03CB D398      OUT 98
03CD F1        POP PSW
03CE C9        RET
               ;
03CF EB        D2_2:XCHG
03D0 21D2FF    LXI H,R
03D3 34        INR M
03D4 EB        XCHG
03D5 C3DF02    JMP D1_2
```

```
                ;END
ADDCX           =00B8   ADDSP           =0191   ADINX           =009D
ADINX2          =00A1   ADRES           =83EE   ADRES1          =83EF
ADSET           =0094   ADSTR           =00A4   BRENT           =0151
BRKAD           =83F0   BRKCT           =83F2   BSAVE           =83E9
BSTOP           =018B   CSAVE           =83E8   D1              =02DD
D1_2            =02DF   D1_3            =02E1   D2              =02EA
D2_2            =03CF   D3              =02EF   DATA            =83EC
DATA1           =83ED   DIG             =83F8   DIGIT           =0084
DISP            =83F4   DSAVE           =83E7   ERRT            =031E
ESAVE           =83E6   FSAVE           =83EA   GOTO            =00CC
HDCMP           =0301   HLDTDP          =02F4   HSAVE           =83E5
INPUT           =0223   INPUT2          =022A   INPUT3          =023D
KEY             =0247   KEYI            =0271   KEYI2           =0279
KEYIN           =0216   KFLAG           =83F3   LDATA           =0107
LDATA2          =0122   LDDR            =0827   LDERR           =0307
LDIR            =0824   LSAVE           =83E4   MEMR            =00AD
MEMW            =00C2   MONSP           =83D1   MONST           =003B
MONST2          =0045   NDZMON          =0800   NOBRK           =0185
NOKEY           =0242   PSAVE           =83E0   PSHRGDP         =02F7
R               =FFD2   RESRG           =01F9   RGDSP           =01A1
RGDSP2          =01A9   RLOAD           =0369   RLOAD2          =036F
RSAVE           =0347   RSAVE2          =0351   RSIN            =037D
RSIN1           =0380   RST1            =FFBA   RST2            =FFBD
RST3            =FFC0   RST4            =FFC3   RST5            =FFC6
RST6            =FFC9   RST7            =FFCC   SBCHLBC         =082A
SDATA           =00D5   SDATA2          =00F2   SEGCG           =01C0
SEGCG2          =01C9   SEGD            =01E9   SEGDP           =0313
SEGDP2          =0316   SHIFT           =01B5   SIN             =02A0
SIN2            =02A3   SINERMK         =FFB8   SINSB           =038D
SINSB2          =0399   SINSB3          =03AD   SINSB4          =03B9
SOUT            =027C   SOUT2           =027D   SOUT3           =028C
SOUTSB          =0326   SOUTSB2         =032E   SOUTSB3         =0338
SSAVE           =83E2   START           =0051   START2          =005B
TABL            =0074   USRSP           =83C7
```