2005.1.17

1　マシン語とは?

1 バイトのデータが並んだものがマシン語プログラムの姿であり, マシン語プログラムはメモリの上によみこまれてから CPU により直接実行される. CPU にはいろいろな種類のものがあるが, ここでは Window 機で必ず利用できる Intel 8086 CPU について解説する. [1] Windows では.exe や .com 拡張子がついたファイルがマシン語のプログラムである.

マシン語の例をみてみよう:

```
B4, FF, B9, 34, 12
```

なる 16 進数の列 (5 バイトある) はマシン語のプログラムであり, 8086CPU はこれを読んで実行する. “CPU の `AH` レジスタ (変数) に (16 進数の) `FF` を代入し (B4 FF の部分), それから `CX` レジスタに (16 進数の) `1234` を代入せよ (B9 32 12 の部分)” という意味である. これを C 言語風に書くと

```
ah = 0xff;
cx = 0x1234;
```

である.

さて, メモリの上のマシン語プログラムを編集, 操作するのが `debug` コマンドである. 次の節では debug コマンドおよびコマンドプロンプトの使い方を説明しよう.

2　コマンドプロンプトと debug

`debug` コマンドを使いこなすには, コマンドプロンプト (command prompt) を使いこなせることが望ましい.

コマンドプロンプトの起動:
スタート ⇒ プログラム ⇒ アクセサリ ⇒ コマンドプロンプト

コマンドプロンプトが起動したらコマンドプロンプトのウインドウに `notepad` と入力してみよう. メモ帳が起動する. `notepad.exe` という機械語ファイルが実行されておなじみのメモ帳が動き出す.

```
C:¥> notepad
```

[1] 8086 CPU と Intel Pentium CPU の関係については後述する.

次に "c:\Program files\Internet Explorer\iexplore" と入力してみよう. (\ は ¥.) インターネットエクスプローラが起動する. iexplore という機械語ファイルが実行されておなじみのインターネットエクスプローラが動き出す.

```
C:¥> "c:¥Program files¥Internet Explorer¥iexplore"
```

このようにコマンドプロンプトは機械語のプログラムをスタートする機能を持っている. 機械語のプログラムは環境変数 PATH に指定されているフォルダのなかから検索される. 環境変数 PATH の値を知るにはコマンドプロンプトから次のように入力する.

```
C:¥> echo %PATH%
```

コマンドプロンプトは機械語のプログラムをスタートする機能の他にいろいろと有用なコマンドが組み込まれている. 主なものをいくつかあげて置こう.

dir
現在のディレクトリ (フォルダ) のファイル名の一覧を表示する.
dir は directory.

del ファイル名
“ファイル名” を消す (del は delete).

copy ファイル1 ファイル2
“ファイル 1” を “ファイル 2” にコピーする.

cd フォルダ名
現在のディレクトリを “フォルダ名” へ変更する.
cd は change directory.

コマンドプロンプトはマウスによる操作はできない. キーボードからの何らかの文字列の入力をおこない操作を行う.

コマンドプロンプトについて, 詳しく知りたい場合は http://www.confrage.com/dos/ とか http://ykr414.com/dos/ を読んでみよう.

参考:

● Windows の起動ができないときに F8 キーを押し続けると復旧用のメニューがでるが, そのとき XP の回復コンソール を選択すると起動するのが

コマンドプロンプトである.
● Windows の前身 MSDOS ではコマンドプロンプトがすべての基本であった. Windows はハードディスクにインストールする必要があるが, MSDOS は Floppy 一枚で動作させることが可能. 興味がある場合は google で freedos を検索.

3 最小限のハードウエアの知識

キーワード: CPU, アドレスバス, データバス, I/O ポート. クロック. メモリ. 16 bit モードと 32 bit モード.

4 ニーモニック

最初にみたマシン語の例では

```
B4, FF, B9, 34, 12
```

なる 16 進数の列 (5 バイトある) がマシン語のプログラムであり, 8086CPU はこれを読んで実行する, と説明した. 機械語のプログラムを 8086CPU の命令一覧を見ながら数字で書いていってもいいのであるが, それは大変なので覚えやすくかつ読みやすい機械語の表記方法としてつかわれているのが, ニーモニックである. 上の機械語をニーモニックで書くと次のようになる.

```
mov ah, FF
mov cx,1234
```

debug の a (assemble) コマンドはニーモニックを機械語に変換する. u (unassemble) は機械語をニーモニックに変換する. メモリを覗くには d (dump) コマンドを用いる. 次の例をためしてみよう.

```
C:¥>> debug   (debug を起動する)
-   a 100
080A:100   mov ah,ff
080A:102   mov cx,1234
080A:105 (なにも入力せず ENTER キーを押す.)
-  d 100,105
080A:0100  B4 FF B9 34 12 46      -                              ...4.F
```

```
-u 100,104

080A:0100 B4FF          MOV     AH,FF
080A:0102 B93412        MOV     CX,1234
```

5 レジスタ, mov と int

CPU には名前のついた変数があり レジスタ と呼ばれている. それぞれのレジスタには特有の役割がある. 8086 にはレジスタ ax, bx, cx, dx などの汎用レジスタの他に ds, cs などのセグメントレジスタがある. これらのレジスタはすべて 16 bit の大きさをもつ. [課題: レジスタ名を記憶せよ.]

ax レジスタの上の 8 bit 分を ah レジスタ, ax レジスタの下の 8 bit 分を al レジスタと呼ぶ. 他の汎用レジスタについても同様である.

さてセグメントレジスタとは何であろうか? 8086 CPU ではメモリのアドレスの指定に セグメント方式を採用している. この方式ではアドレスの指定にはセグメントアドレスとオフセットアドレスと二つの数字の組を用いる. アドレスは次の式で与えられる.

$$\text{セグメントアドレス} * \texttt{0x10} + \text{オフセットアドレス} = \text{アドレス}$$

たとえば セグメント 080A の オフセット 0100 番地というのは実際には 081A0 番地を表す.

上の unassemble の例をもう一度読んでみよう.

```
080A:0100 B4FF          MOV     AH,FF
080A:0102 B93412        MOV     CX,1234
```

最初の 080A:0100 BF 44 はどういう意味であろうか? これは実はマシン語プログラム BF 44 がセグメント 080A の オフセット 0100 番地から書かれていることを意味している. つまり 081A0 番地にマシン語プログラム BF が, つまり 081A1 番地にマシン語プログラム FF が書かれているわけである. このようにセグメント方式を用いることにより, 20 bit のアドレスを指定できる. 問題: セグメント 080A の オフセット 0102 のアドレスはいくつか?

mov を用いてレジスタにデータを移動することが可能であるが, メモリとレジスタの間でデータのやりとりをするにはどのようにすればよいのであろうか?

```
mov [bx],ah
```

という命令はセグメントアドレス ds, オフセットアドレス bx (ds:bx 番地) に, ah レジスタ (8bit) の中身を書き込めという命令である. このようにメモ

リへデータを書き込む方法を 間接アドレッシング によるデータの書き込みという. なお上のマシン語プログラムを C 言語風に書くと

```
*bx = ah
```

C のポインタはマシン語の間接アドレッシングに由来している. ちなみに変数 ax, bx, cx, dx のみで書いた C のプログラムは容易にマシン語に変換できる.

10AF:1234 番地にデータ FF を書き出すプログラムは以下のとおり.

```
mov ax,10AF
mov ds,ax
mov bx,1234
mov ah,FF
mov [bx],ah
```

次のように書けると簡潔だが無い命令があるのでできない.

```
mov ds,10AF
mov bx,1234
mov [bx],FF
```

int 命令はオペレーティングシステムが提供しているサブルーチンを呼び出すのに用いる.

int 20 が終了, int 21 が MSDOS のシステムコール (組み込み関数みたいなもの) の呼出, int 10 が graphic bios の呼出, である. bios (basic input output system) の仕様については, http://www.oadg.or.jp の Dos/V 仕様書を見よ. MSDOS のシステムコールについては随時解説する.

6 キャラクタ VRAM への直接書き込みをしてみよう

debug では e コマンド (edit) を用いてメモリにデータを書き込むことが可能である. セグメント, オフセットアドレスの理解を深めるため, キャラクタ VRAM への直接書き込みを試みてみよう.

us を入力してまず 英語モードにする. debug を起動して, 次の命令を書き込む.

```
C:¥> us    (英語モードに)
C:\> debug    (debug を起動する)
- e    b800:0000
b800:0000          00. 42 [SPACE]
```

画面の左上に表示された文字が “B” に変わったであろう. これは debug の e コマンドによりキャラクタ VRAM へ書き込んだデータ 41 (16 進数, “B” のアスキーコード) が表示されたものである.

http://www.oadg.or.jp の Dos/V 仕様書のメモリマップに関する資料を参照.

例題: “TAKA” を画面に書くプログラムを作れ. なお T のアスキーコードは 54H, A のアスキーコードは 41H, K のアスキーコードは 4BH である.

次のファイルを メモ帳 (notepad) で作る. 名前は name.txt とする.

```
a 100
   mov ax,b800
   mov ds,ax
   mov bx,0000
   mov al,54
   mov [bx],al
   inc bx
   inc bx
   mov al,41
   mov [bx],al
   inc bx
   inc bx
   mov al,4b
   mov [bx],al
   inc bx
   inc bx
   mov al,41
   mov [bx],al
   jmp 100

g =100
quit
```

inc bx は bx の中身を一増やせという意味である.

次に debug を起動して下のようにプログラムを実行する.

```
C:¥>  us
C:\>  debug <name.txt
```

プログラムが終了して, 画面がスクロールすると書いたものが消えてしまうので, このプログラムでは無限ループをしている. プログラムを停止するには, コマンドプロンプト自身を停止する必要がある.

メモ帳からコマンドプロンプトへカットアンドペーストでプログラムを貼り付けて実行している場合は `jmp 100` を

```
mov ah,08
int 21
int 20
```

とおきかえるとよい.

`g =100` で 100 番地からの実行を開始したあと, Enter キーをおすと終了する. `ah` レジスタに 8 を入れて, `int 21` をよぶとキーボードからの入力待ちになる. (MSDOS のシステムコール)

課題: 自分の名前をローマ字で画面に書くプログラムを作れ.

7 グラフィック VRAM への直接書き込みをしてみよう

us モードにしてから, debug を起動して, 次の命令を書き込む.

```
C:\> debug   (debug を起動する)
- a   100
080A:100 mov  ax,12
080A:103 int  10
080A:105 int  20
080A:107 (なにも入力せず ENTER キーを押す.)
- g   =100   (100 番地から実行)
```

Windows 2000 や Windows XP の場合, 画面が真っ黒になるが異常はない.

```
- quit
C:\> exit   (コマンドプロンプトを抜ける)
```

で Windows のおなじみの画面へ戻ることができる.

さて, 真っ黒の画面の状態で, 次のように入力してみよう.

```
- e    a000:7f00
a000:7f00          00. ff SPACE
```

画面の下の方に白い線が出現したであろう. これは debug により VRAM へ書き込んだデータが表示されたものである.

ノート `mov ax,12` で 640 x 400 モード, `mov ax,13` で 320 x 200 モード.

簡単な機械語のプログラムを書くことにより, 画面をすべて “B” で埋め尽くすことや, 特定の模様で埋め尽くすことも可能である.

課題: そのような動作をするプログラムを書いてみよう. (loop 命令)