2014-12-20　ITS 5 課 坂本

# DEBUG.EXE を利用した実行ファイルの作り方

## 【概要】

- Windows に標準装備(※1)の「DEBUG.EXE」を利用して実行ファイルを作成する手順を紹介します。ここでは“Hello,world”を表示するだけの 16bit プログラムを作ってみます。
- DEBUG コマンドによるプログラム作成法は、現在では実用的ではないものの、BadUSB（※2）など、キーボードを偽装した攻撃では利用される可能性があり、手法を知っておくことは無駄ではないと思います。

(※1) Windows 7 (32bit 版) までは DEBUG コマンドは標準添付されているが、Windows 7 (64bit 版) 以降では添付されていない。 また、Windows 2000 以降の 64bit 版 OS では添付されていないとのこと。

参考： MS-DOS から続く「DEBUG」コマンド、Windows 7 (64 bit) には搭載されず （スラッシュドット）
http://slashdot.jp/story/09/05/13/1516240/MS-DOS-%E3%81%8B%E3%82%89%E7%B6%9A%E3%81%8F%E3%80%8CDEBUG%E3%80%8D%E3%82%B3%E3%83%9E%E3%83%B3%E3%83%89%E3%80%81Windows-7-64-bit-%E3%81%AB%E3%81%AF%E6%90%AD%E8%BC%89%E3%81%95%E3%82%8C%E3%81%9A

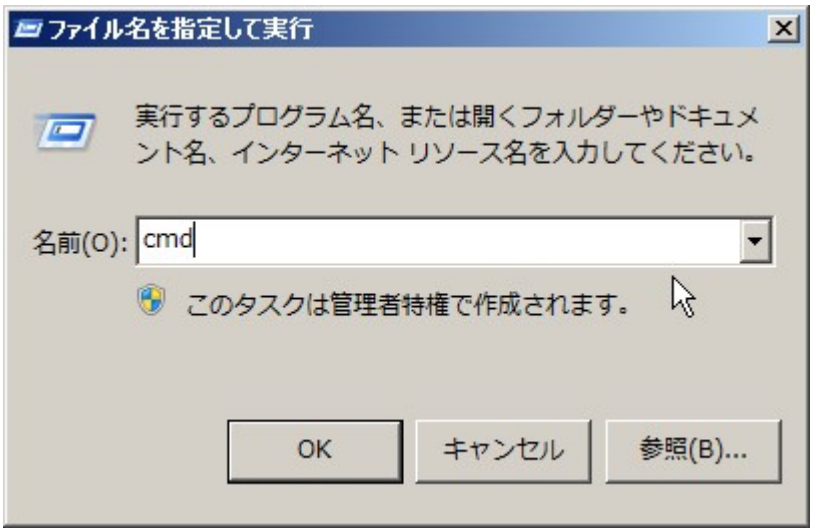
(※2) BadUSB. . . USB メモリ等のファームウェアを書き換えて キーボードやマウスなどの HID 機器として動作させることでワクチンソフトのチェックをかいくぐり、悪意のある操作を実行する攻撃手法。

## 【手順】

ここでは Windows 7 (32bit 版) の動いている PC 上で、“Hello,world” を表示するだけのプログラム（実行ファイル） を作ってみます。 (64bit 版 OS では実行できません)

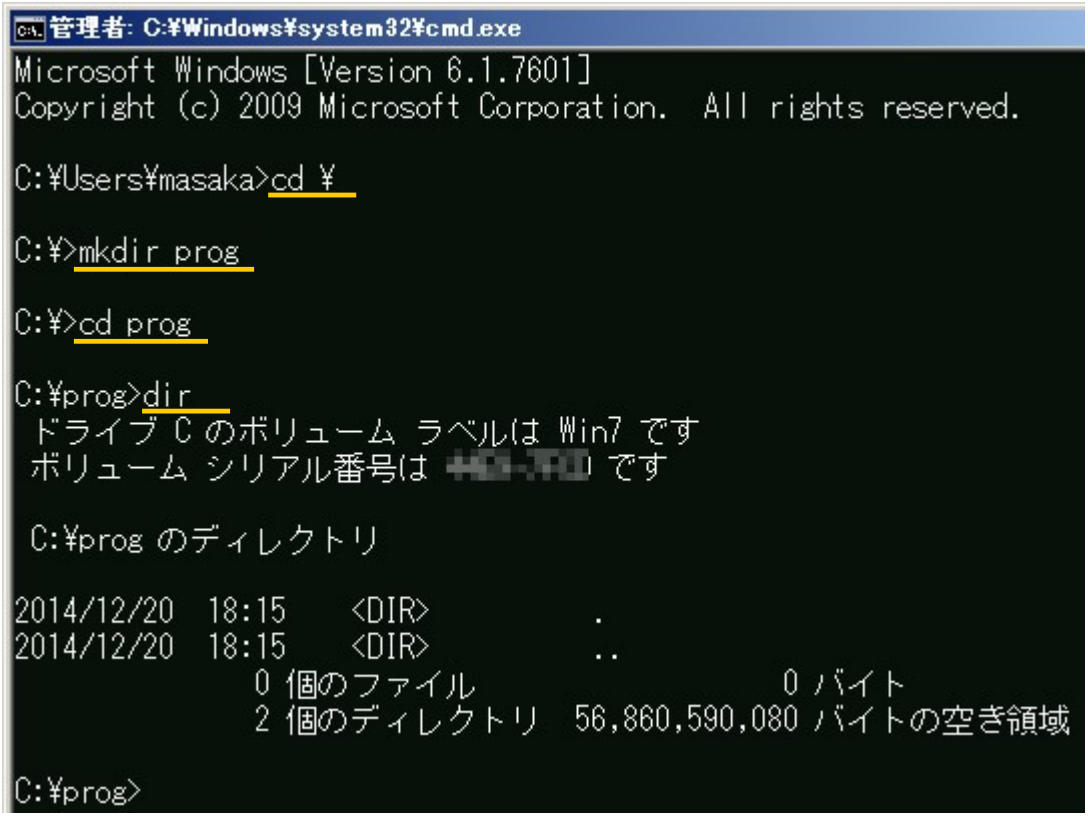
とりあえず作業用のフォルダ 「C:¥prog」 を作って、この中でファイルを一つ作ることにします。

(1) スタートボタンで「ファイル名を指定して実行」で “cmd” と入れてコマンドプロンプトのウィンドウを開いて下さい。

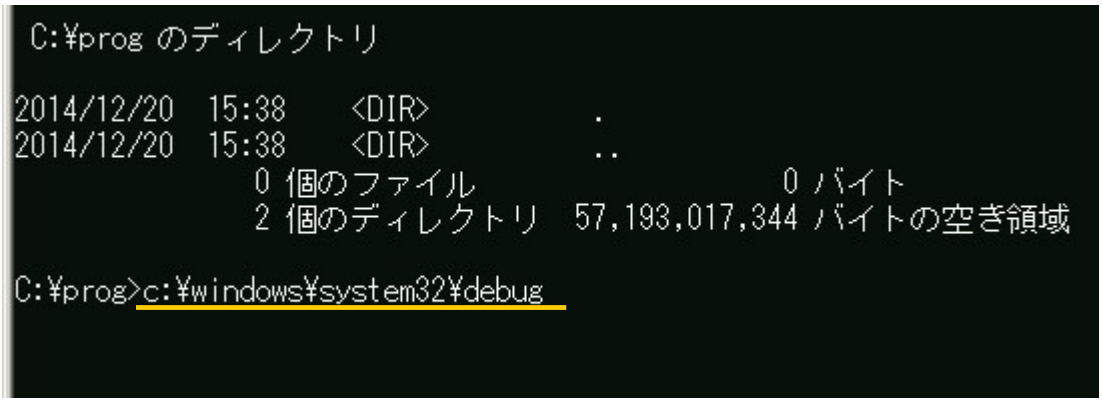



(管理者権限がないと、以降の手順は実行できないかも知れません. . . 参考：http://answers.microsoft.com/ja-jp/windows/forum/windows_7-windows_programs/windows-7/efef116a-2ea7-43f5-ac06-f553737870fa )

(2) 作業用の「C:¥prog」 フォルダ(ディレクトリ) を作ります。 (以下、オレンジの下線部を入力して Enter キーを叩きます)



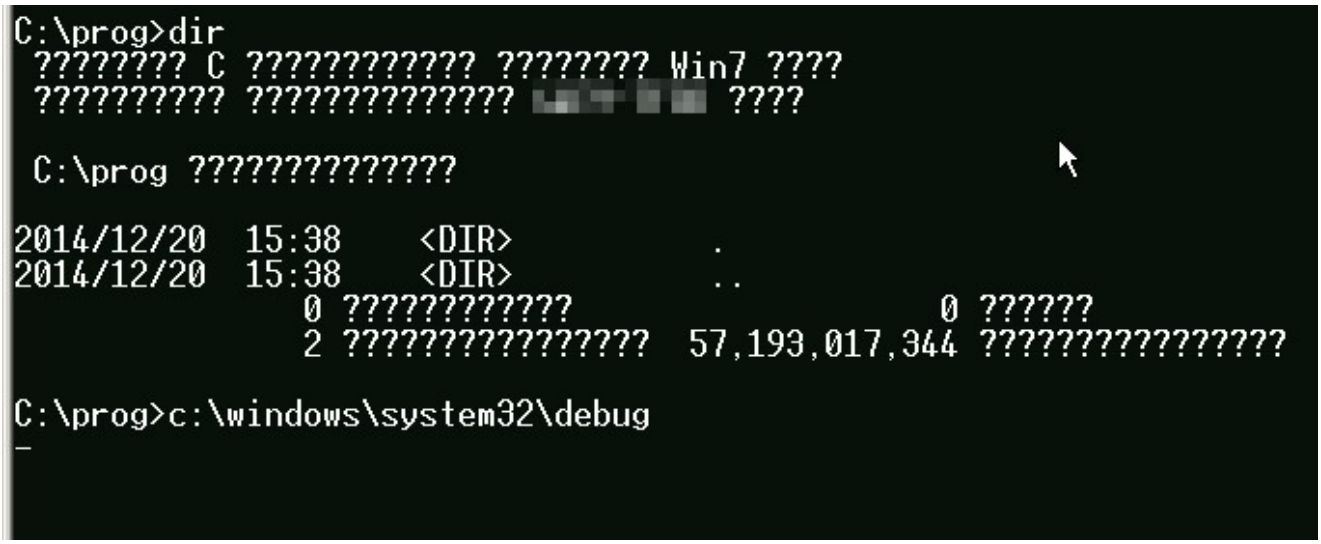

(3) DEBUG コマンドを起動します。
(絶対パス 「 C:¥windows¥system32¥debug」を指定して起動)



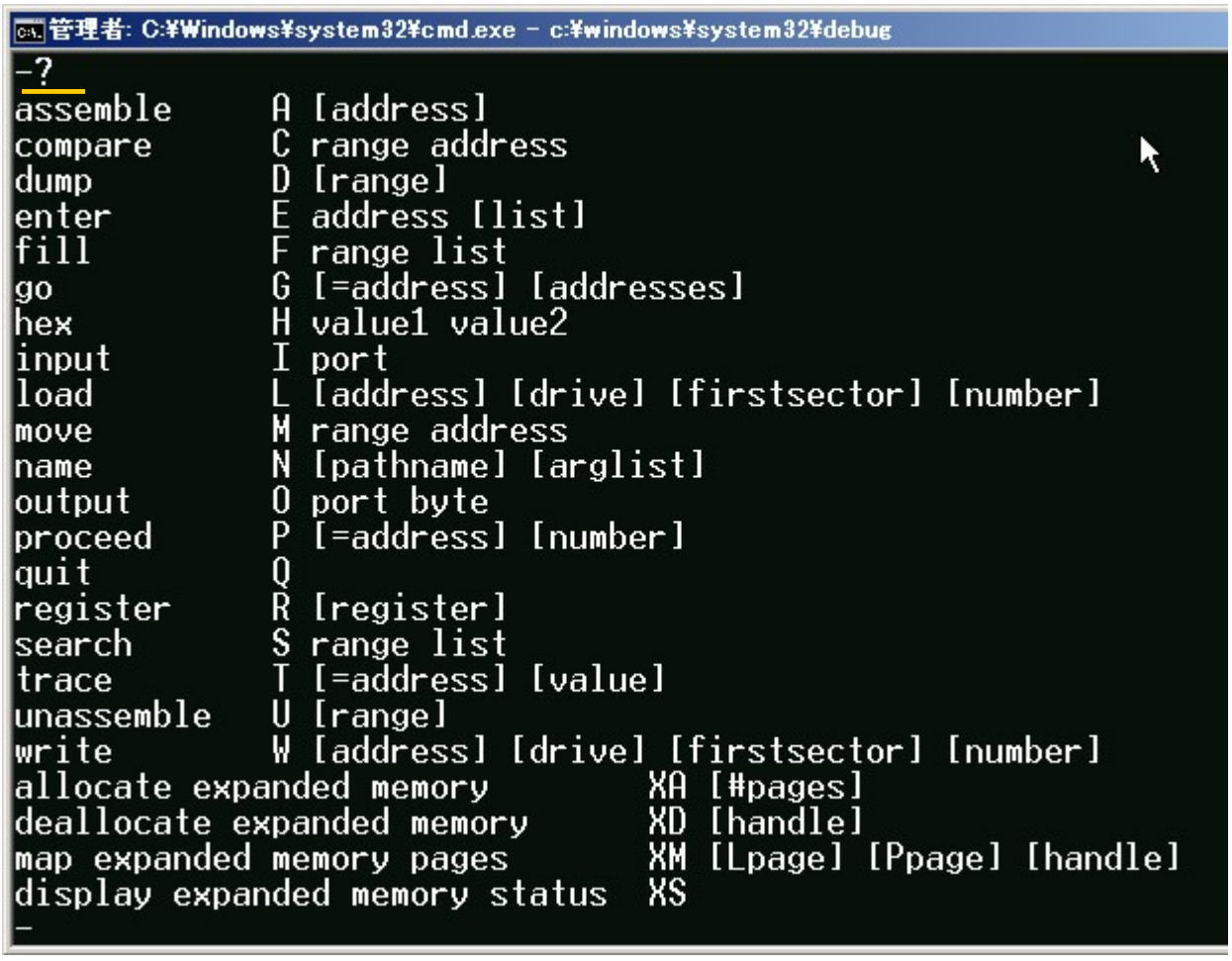

ここで注意ですが、起動するとフォントが変わってしまいます。
DEBUG コマンドは 16bit アプリケーション (MS-DOS 時代の 16bit のプログラム)であるため、コマンドプロンプト画面もそれに応じているのですが、Windows Vista 以降では 16bit 環境用の日本語フォントが無いため、「英語モード」になってしまうそうです。

画面に残っていた漢字やカナは化けていますが、以降の作業に影響はありませんので、続けます。

[画面] 16bit アプリケーションでは英語モードになる（日本語が文字化けする）

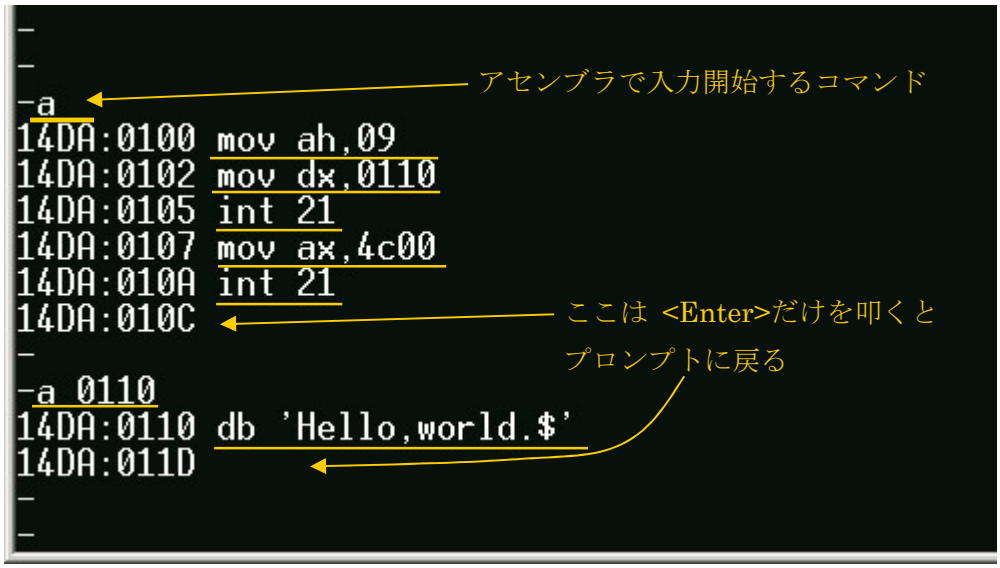



(4) DEBUG コマンド から プロンプト “-“ (ハイフン) が表示されて、コマンド待ち状態になっています。
ここで “?”を 入力すると、コマンドの簡単なヘルプが表示されます。




とりあえず「抜けるのは q 」だけは覚えましょう。

(5) 下記のアセンブラのソースの通りに、打ち込んでみて下さい。




このアセンブラ・プログラムの内容を説明します。

MS-DOS 時代の 16bit プログラムでは、CPU のレジスタに機能番号やパラメータを設定して、「int 21h」命令を実行することで、OS の機能を呼び出します。（ファンクションコールと言う）

OS のサポートする機能は、文字のキーボード入力や画面出力、環境変数の取得、ファイルの読み出し／書き込み等、色々あります。

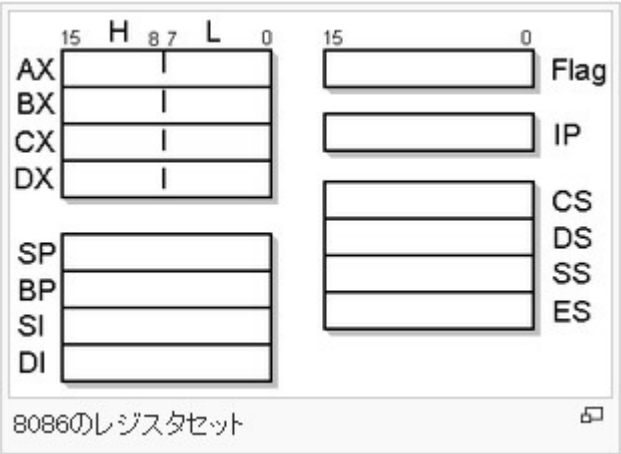
このプログラムでファンクションコールを二つ呼び出しています。ひとつ目は、「文字列を表示する」ファンクションコールで、AH レジスタ（16 ビット幅の AX レジスタの上位 8 ビット）に機能番号の 09h を書き込み、 DS:DX レジスタに文字列の先頭アドレスを入れて int 21h を実行すると、文字列が画面に表示されます。 ここでは表示したい文字列は 0110h 番地から入れておくことにしました。終端は “$” にするというルールです。

「DS:DX レジスタ」でアドレスを指し示す方式は、DS レジスタにセグメントアドレス、DX レジスタにオフセットアドレスを入れる必要があります。今作っているプログラムは最後に「COM ファイル」という実行ファイル形式で保存しますが、この形式の実行ファイルではメモリにロードされたとき、DS レジスタにはセグメントアドレスが自動的に設定されるので、DX レジスタにオフセットアドレスを入れるだけで OK です。

二つ目のファンクションコールは「プログラムを終了して OS に戻る」というものです。 AH レジスタに機能番号の 4Ch (16 進数の 4C) 、AL レジスタに終了コードをいれます。

AL レジスタに何を入れても、入れなくても良いのですが、ここは(お作法として) 正常終了を意味する 00h を入れることにします。 AX レジスタに 4C00h を入れることで AH,AL レジスタ合わせて一気に書き込んでいます。

CPU のレジスタの様子は、下記の図を参照して下さい。




8086 CPU のレジスタセット
(Wikipedia より)

プログラムが、オフセットアドレス 0 ではなく、0100h 番地から始まっているのを不思議に思う人もいると思います。これは、COM 形式のプログラムがメモリにロードされるとき、0～00FFh 番地には、コマンドライン引数や環境変数などの情報を、OS が用意して書き込んでくれるという仕組のためです。プログラムはオフセットアドレスの 0～00FFh の内容を参照すれば、それら情報を取得できるというわけです。この領域は、PSP (プログラム・セグメント・プレフィックス) と呼びますが、詳しい説明は省略します。(ネットで検索下さい)

(6) さて、今はメモリ上に書き込んだプログラムを、逆アセンブルコマンド “u” で見てみましょう。

```
-u 100
14DA:0100 B409          MOV     AH,09
14DA:0102 BA1001        MOV     DX,0110
14DA:0105 CD21          INT     21
14DA:0107 B8004C        MOV     AX,4C00
14DA:010A CD21          INT     21
14DA:010C 0000          ADD     [BX+SI],AL
14DA:010E 0000          ADD     [BX+SI],AL
14DA:0110 48            DEC     AX
14DA:0111 65            DB      65
14DA:0112 6C            DB      6C
14DA:0113 6C            DB      6C
14DA:0114 6F            DB      6F
14DA:0115 2C77          SUB     AL,77
14DA:0117 6F            DB      6F
14DA:0118 726C          JB      0186
14DA:011A 64            DB      64
14DA:011B 2E            CS:
14DA:011C 2400          AND     AL,00
14DA:011E C9            DB      C9
14DA:011F 1457          ADC     AL,57
-
```

ここが
プログラムの
コード本体

ここは
Hello,world.の
文字列

ちゃんと、意図した通りのコードになっています。 “Hello～”の所まで逆アセンブルしてしまいましたが気にしなくて良いです。

(7) 16 進ダンプでも見てみましょう。 “d”コマンドを使います。

```
-
-d 100L20
14DA:0100  B4 09 BA 10 01 CD 21 B8-00 4C CD 21 00 00 00 00   ......!..L.!....
14DA:0110  48 65 6C 6C 6F 2C 77 6F-72 6C 64 2E 24 00 C9 14   Hello,world.$...
-
```

d コマンドの引数を「 100L20 」としていますが、これは「アドレス 100h から長さ 20h だけ」を意味します。

(8) ファイルに書き出します。

コマンドここで使うコマンドは 3 つです。

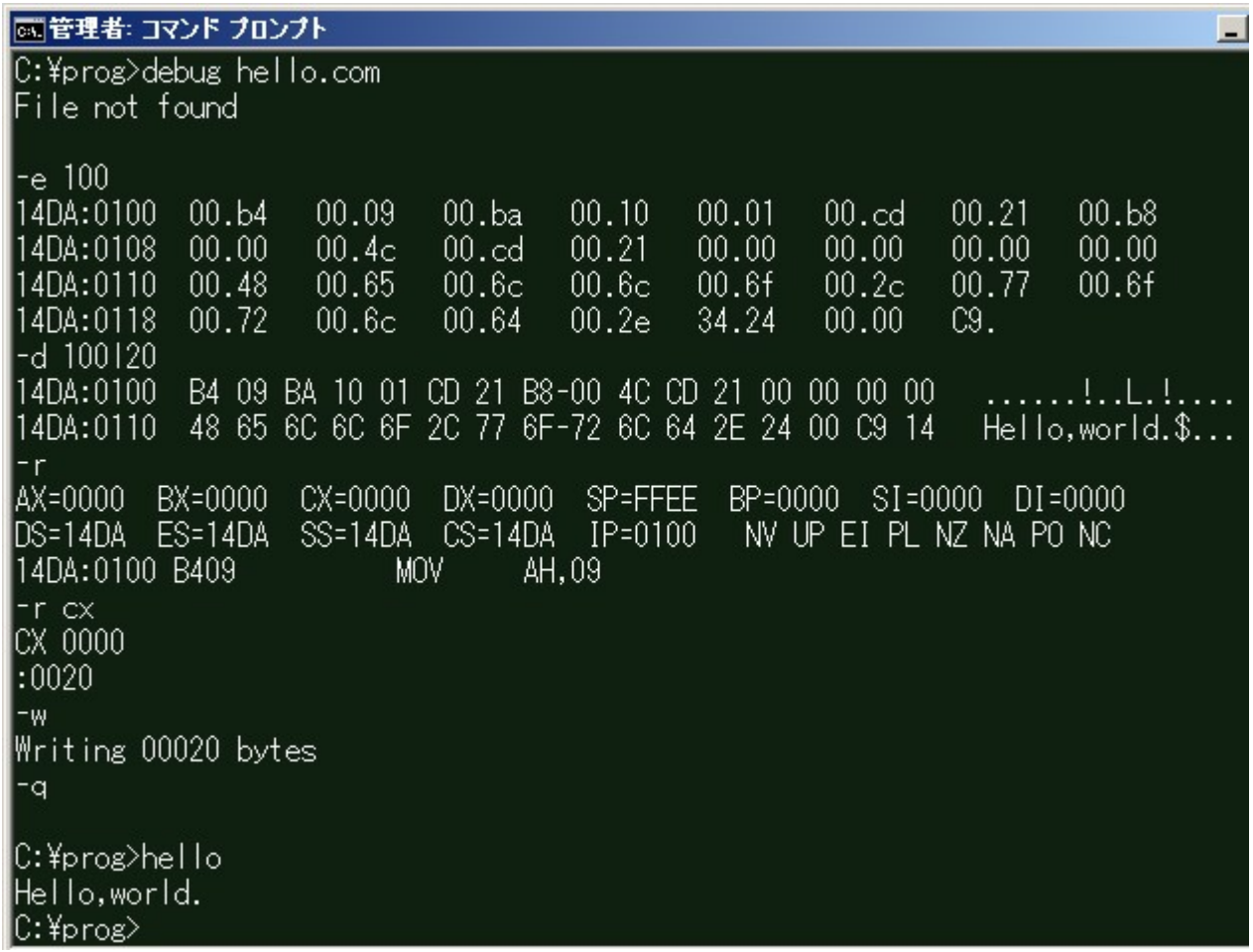
- n <ファイル名> ……ファイル名を設定する
- r <レジスタ名> ……レジスタを指定して値を書き込む。またはレジスタ値の一覧を表示する
- w ……ファイルに書き出す。。

コマンドここで使うコマンドは 3 つです。

```
-n hello.com
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=14DA  ES=14DA  SS=14DA  CS=14DA  IP=0100   NV UP EI PL NZ NA PO NC
14DA:0100 B409          MOV     AH,09
-r cx
CX 0000
:0020
-r
AX=0000  BX=0000  CX=0020  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=14DA  ES=14DA  SS=14DA  CS=14DA  IP=0100   NV UP EI PL NZ NA PO NC
14DA:0100 B409          MOV     AH,09
-w
Writing 00020 bytes
-
```

```
-w
Writing 00020 bytes
-q

C:¥prog>dir
 ドライブ C のボリューム ラベルは Win7 です
 ボリューム シリアル番号は [illegible] です

 C:¥prog のディレクトリ

2014/12/20  15:44    <DIR>          .
2014/12/20  15:44    <DIR>          ..
2014/12/20  15:44                32 HELLO.COM
               1 個のファイル                  32 バイト
               2 個のディレクトリ  57,193,201,664 バイトの空き領域

C:¥prog>
C:¥prog>
```

```
C:¥prog>
C:¥prog>hello
Hello,world.
C:¥prog>
```


管理者: コマンド プロンプト
C:¥prog>debug hello.com
File not found
-e 100
14DA:0100 00.b4 00.09 00.ba 00.10 00.01 00.cd 00.21 00.b8
14DA:0108 00.00 00.4c 00.cd 00.21 00.00 00.00 00.00 00.00
14DA:0110 00.48 00.65 00.6c 00.6c 00.6f 00.2c 00.77 00.6f
14DA:0118 00.72 00.6c 00.64 00.2e 34.24 00.00 C9.
-d 100l20
14DA:0100 B4 09 BA 10 01 CD 21 B8-00 4C CD 21 00 00 00 00 ......!..L.!....
14DA:0110 48 65 6C 6C 6F 2C 77 6F-72 6C 64 2E 24 00 C9 14 Hello,world.$...
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=14DA ES=14DA SS=14DA CS=14DA IP=0100 NV UP EI PL NZ NA PO NC
14DA:0100 B409 MOV AH,09
-r cx
CX 0000
:0020
-w
Writing 00020 bytes
-q
C:¥prog>hello
Hello,world.
C:¥prog>