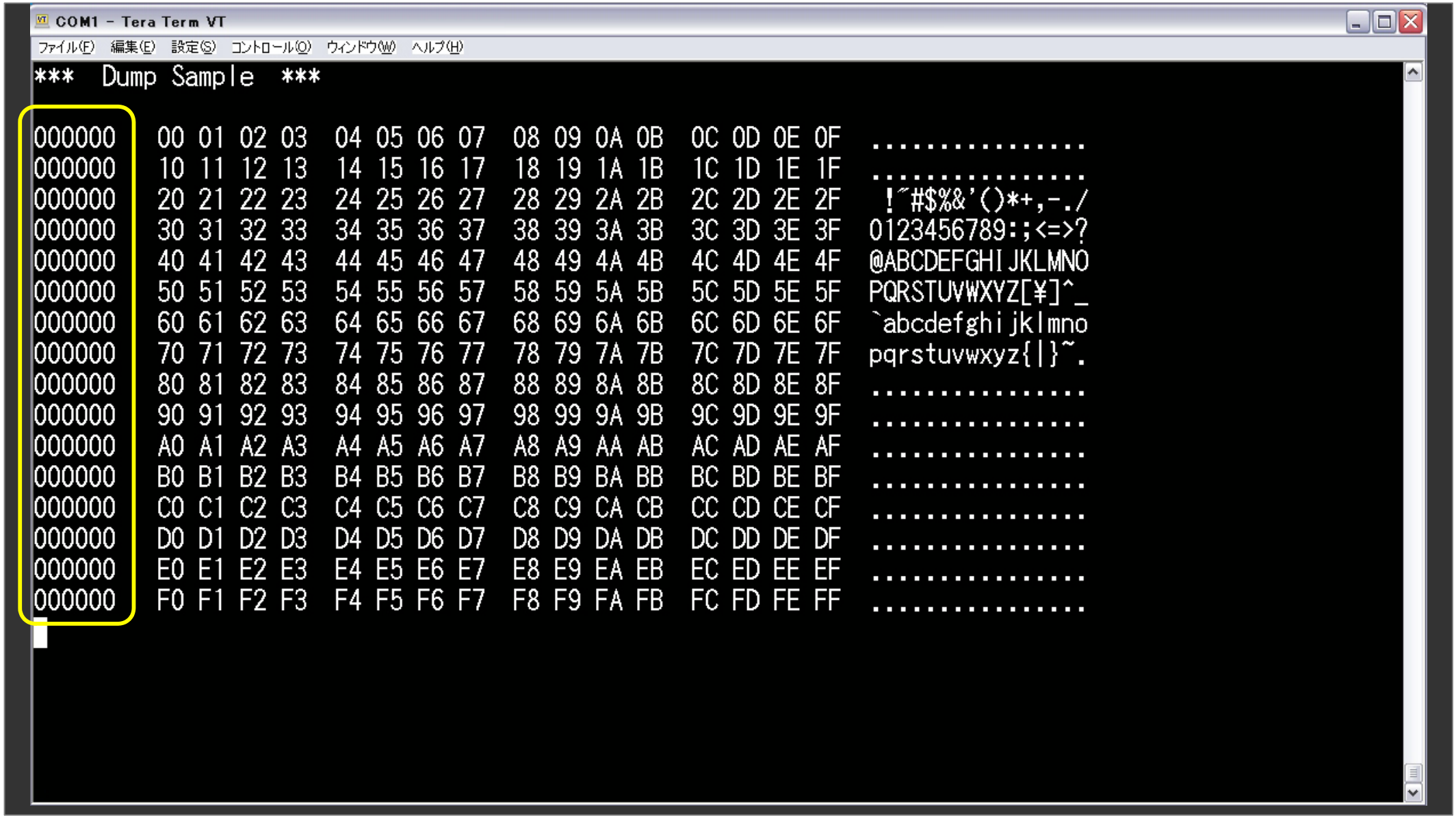
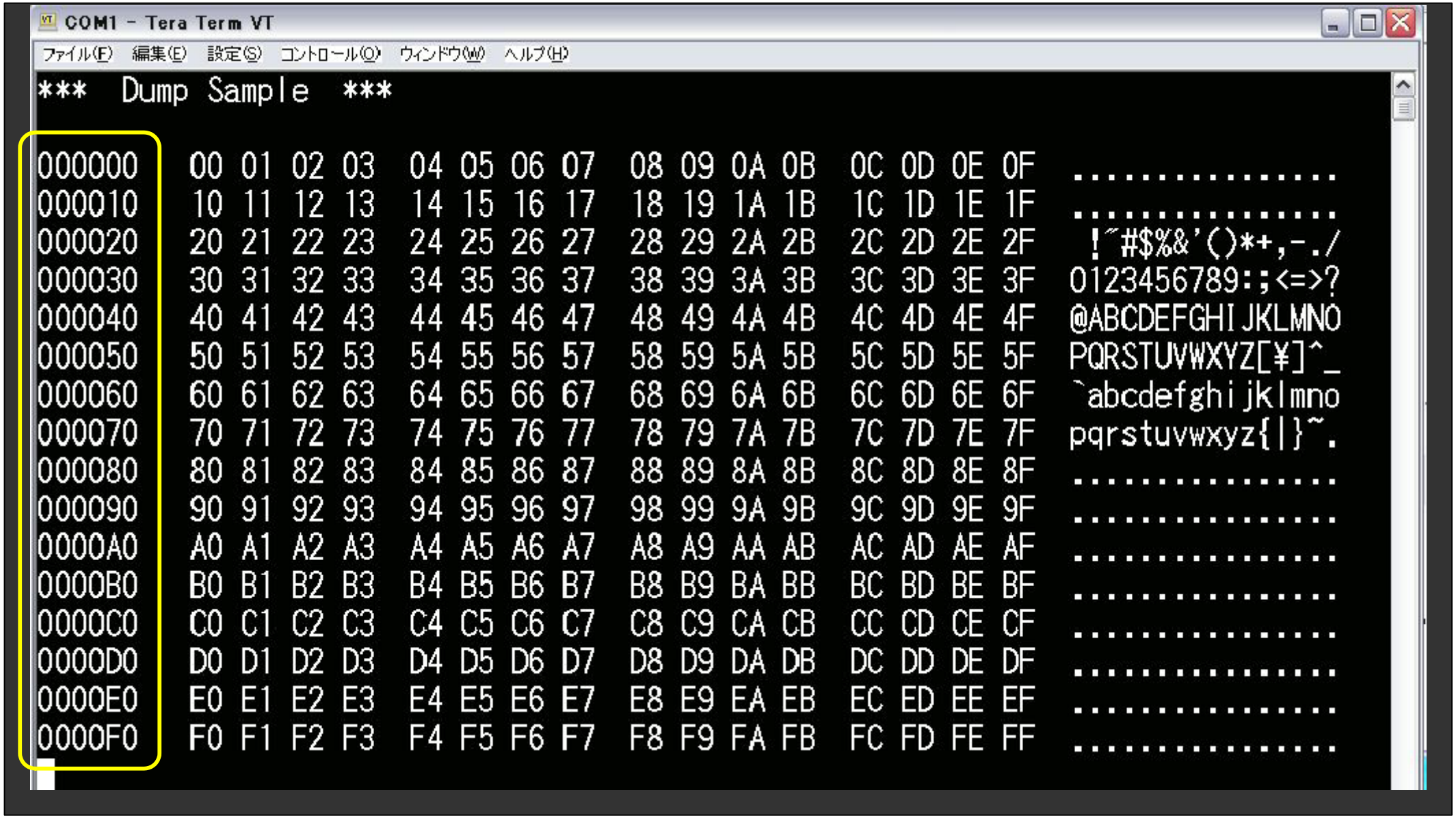
R8Cと H8 マイコン間のプログラム移植の 注意点

今回は、前回の動画で発覚した現象で、H8マイコンの メモリアドレス表示用で 用意していた 3byte整数の 16進文字列表示処理にて、最初、16行全てが 000000h を 表示していました。 テラタームに表示した画像を お見せします。

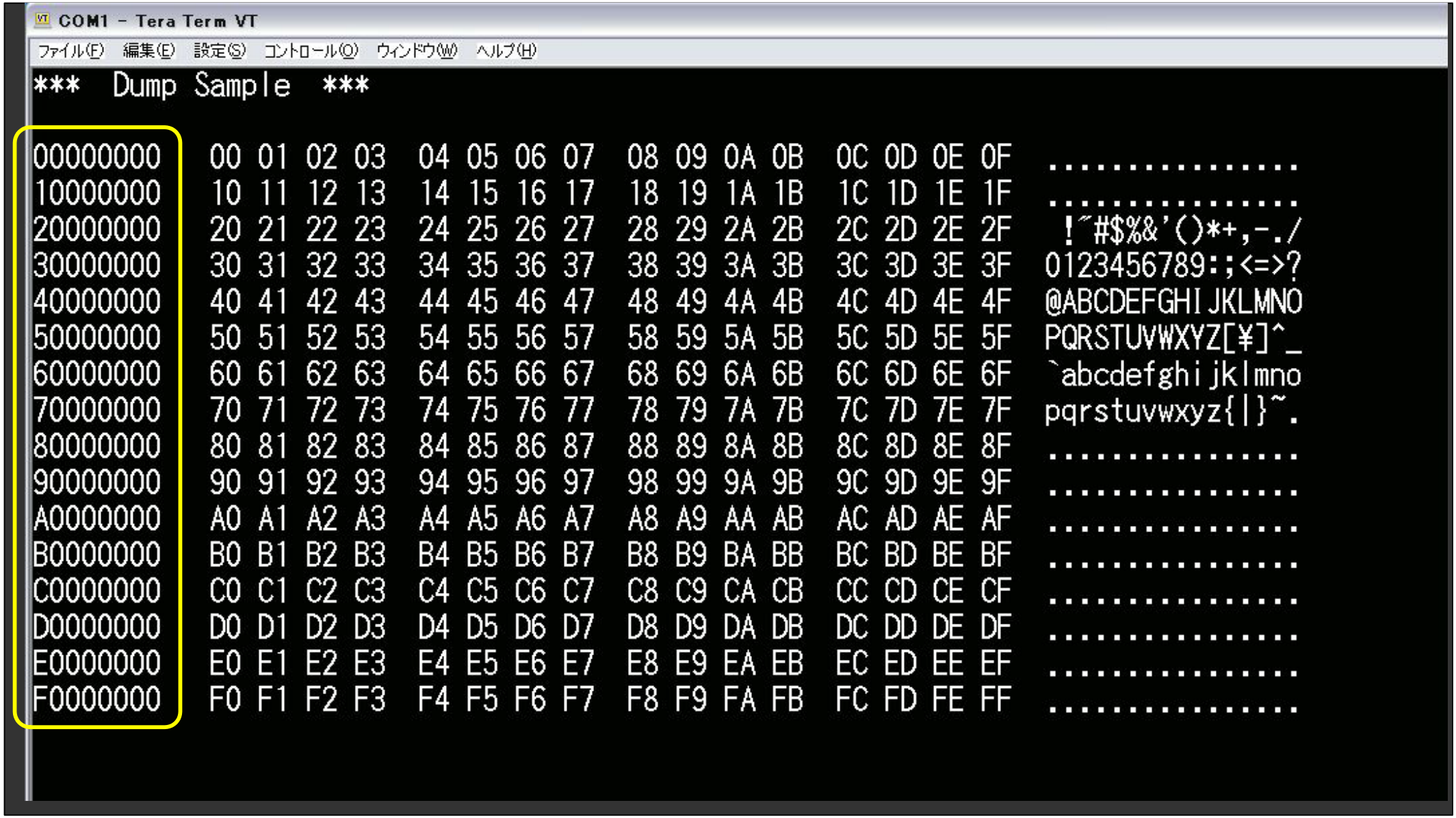

COM1 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
*** Dump Sample ***
000000 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F .....
000000 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F .....
000000 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F !"#$%&'()*+,-./
000000 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 0123456789:;<=>?
000000 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F @ABCDEFGHIJKLMNO
000000 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F PQRSTUVWXYZ[¥]^_
000000 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F `abcdefghijklmno
000000 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F pqrstuvwxyz{|}~.
000000 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F .....
000000 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F .....
000000 A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF .....
000000 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF .....
000000 C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF .....
000000 D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF .....
000000 E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF .....
000000 F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF .....


COM1 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
*** Dump Sample ***
000000 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ....
000010 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F ....
000020 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F !"#$%&'()*+,-./
000030 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 0123456789:;<=>?
000040 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F @ABCDEFGHIJKLMNO
000050 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F PQRSTUVWXYZ[¥]^_
000060 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F `abcdefghijklmno
000070 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F pqrstuvwxyz{|}~.
000080 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F ....
000090 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F ....
0000A0 A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF ....
0000B0 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF ....
0000C0 C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF ....
0000D0 D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF ....
0000E0 E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF ....
0000F0 F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF ....

何故、アドレス表示が、全て 00 00 00hに なるのか 最初、全く見当が 付きませんでした。

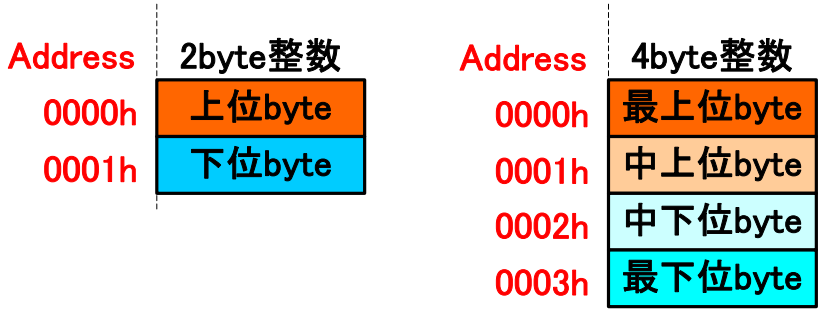
で、longの値を 16進 6桁で表示しているので試しに、16進 8桁で表示すると 原因が 見えてきました。
察しのいい人なら 分かると思います。

では アドレス表示を 8桁で表示した、テラタームの画面をお見せします。

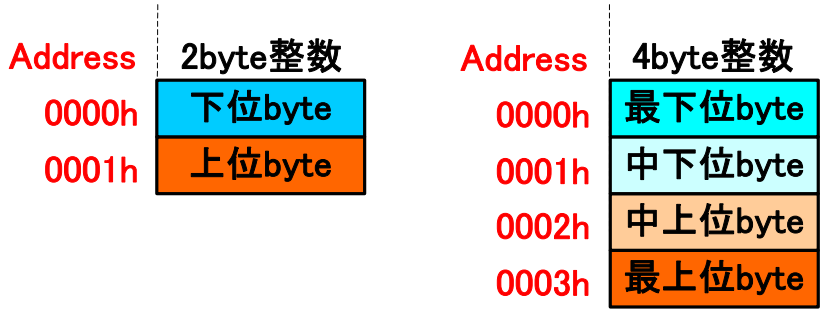

COM1 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
*** Dump Sample ***
00000000 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ....
10000000 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F ....
20000000 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F !"#$%&'()*+,-./
30000000 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 0123456789:;<=>?
40000000 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F @ABCDEFGHIJKLMNO
50000000 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F PQRSTUVWXYZ[¥]^_
60000000 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F `abcdefghijklmno
70000000 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F pqrstuvwxyz{|}~.
80000000 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F ....
90000000 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F ....
A0000000 A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF ....
B0000000 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF ....
C0000000 C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF ....
D0000000 D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF ....
E0000000 E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF ....
F0000000 F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF ....

ビッグエンディアンと リトルエンディアン

まず、ビッグエンディアンは メモリの先頭から上位byteから順に、下位バイトと並びます。



逆に、リトルエンディアンは メモリの先頭から下位byteから順に、上位バイトと並びます。



何故このようなバイト単位の並びの違いが出来たのかは、よく分かりませんが、パソコンが出てくる前は、無かったと思います。

パソコン以前は、メインフレームコンピュータ、ミニコンが 使用されていました。 メインフレームは触った事はありませんが、20代半ば頃に ミニコンは 触ってました。 では、ミニコンは ビッグエンディアンか リトルエンディアンかというと、どちらでもありません。

アドレスの単位が byteではなくて、wordなのです。 だから、ミニコンで メモリ容量 64Kというと64Kbyteではなくて、64Kwordなのです。 1回にアクセスする単位が wordしかないのです。

よって ビッグエンディアン、リトルエンディアンという概念は ありませんでした。

よって、8bitマイコンの出現により 8bitのデータバス幅になり、アドレス値で 指定するデータ幅も byte単位になったと思われます。 インテルの i8080の前に i4004、i8008というCPUがありました。 さすがに i8080以前のCPUは使った事は ありませんが、コントローラ的なマイコンだったようで、2byte整数を扱う機能が無かったと思われます。 それと インテルのCPU戦略というか、過去の互換性を重視していたように思います。その関係で、1byte 整数のデータを 出力した後に 追加的に 上位byteを 付け加えた事が リトルエンディアンの始まりではないかと思います。

では、ビッグエンディアンは どのように出て来たのかと いうと、パソコンでデータ入力したデータを メインフレームや ミニコンに渡すとき何故かビッグエンディアンで 渡す事に なっていました。 メインフレーム等に データを渡す時は ビッグエンディアンの方が 都合がいいようです。 私も遥か昔 メインフレームを使用している事業所に データを送る時、上下バイトを並べ直してから 送った記憶があります。

今は、パソコンが メインになって来ているので、リトルエンディアンが 使われる事が 多くなってきていると思います。

マイコンでは、パソコンに使用される インテルの CPUが リトルエンディアンの代表格でしょう。 MC68000、H8は ビッグエンディアンで、R8Cは リトルエンディアンです。

MIPS、ARM、SH、RXは リトルエンディアン、ビッグエンディアンの 両方が 設定できます。

バイエンディアンというようです。 RXマイコンは デフォルト設定で リトルエンディアンだったと思います。

ビッグエンディアンのCPUと リトルエンディアンのCPU ソフトでの対応

前回、R8Cマイコンから H8マイコンに 持って来て 問題が 起こったソースですが、文字列変換処理の string_sub.c です。

最初、R8Cの I/Oポートも 内蔵周辺機器も全くアクセスしてないし、intも 同じ 16bitなので移植性は あるだろうと思い H8に 持ってきました。一つだけ、バイトオーダーというか ビッグエンディアンと リトルエンディアンは 見落していました。

で、今回影響を受けたのは、整数データを16進文字列に変換する処理です。この処理は

① shortの 2byte整数を 4桁の 16進文字列に変換する処理

② long の 4byte整数の 下位 3byteを 6桁の16進文字列に変換する処理

③ long の 4byte整数を 8桁の 16進文字列に変換する処理

の、3つです。 それと、16進文字列から整数にに変換する、逆変換処理も 3つ あります。
よって、今回 計 6個の 変換関数に 処置を行いました。

次のページから、任意バイト位置データを 取り出すデータ型宣言と それを利用した整数から Hex文字列変換関数の 変更箇所を説明します。

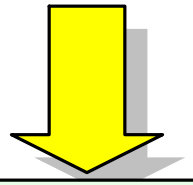
今回のプログラムにて使用する long変数内の任意位置の byteを取り出すために 以下の 構造体 struct と 共用体 union の変数を使用します。

```
typedef struct  {
    _UBYTE  b0, b1, b2, b3;   // Byte  *4
} BYTE_4;

typedef struct  {
    _UWORD  w0, w1;     // Word  *2
} WORD_2;

typedef union   {
    BYTE_4  b;       // Byte  *4
    WORD_2  w;       // Word  *2
   _UDWORD  dw;      // DWord  *1
} TYP_CHG4;
```

上記の データ型宣言で 共用体 unionが 馴染みの薄い宣言と思われますので、説明しておきます。

左の TYP_CHG4は union 共用体です。
左の TYP_CHG4では、bと Wと dwの3つのメンバー変数を 宣言しています。で、構造体と 大きく異なるのは、この3つのメンバー変数が、3つとも 同じアドレスに 配置される事です。 よって実態が 1つの変数に 3つの 型と名前を付けているという事です。 こうする事により、longの値を dwに代入して、そのうちの先頭バイトを取り出す場合は b.b0 で アクセスすれば 先頭アドレスのバイト値を取り出せます。b.b1 で アクセスすれば 先頭アドレス+1 の バイト値を取り出せます。 つまりバイト単位で 任意位置のバイト値を取り出せる事になります。

```
*tx = bin_hex1( tc.b.b3 >> 4 );   tx++; // リトルエンディアン仕様
*tx = bin_hex1( tc.b.b3 );        tx++; // long -> 16進 8文字表示
*tx = bin_hex1( tc.b.b2 >> 4 );   tx++;
*tx = bin_hex1( tc.b.b2 );        tx++;
*tx = bin_hex1( tc.b.b1 >> 4 );   tx++;
*tx = bin_hex1( tc.b.b1 );        tx++;
*tx = bin_hex1( tc.b.b0 >> 4 );   tx++;
*tx = bin_hex1( tc.b.b0 );        tx++;
```



```
*tx = bin_hex1( tc.b.b0 >> 4 );   tx++; // ビッグエンディアン仕様
*tx = bin_hex1( tc.b.b0 );        tx++; // long -> 16進 8文字表示
*tx = bin_hex1( tc.b.b1 >> 4 );   tx++;
*tx = bin_hex1( tc.b.b1 );        tx++;
*tx = bin_hex1( tc.b.b2 >> 4 );   tx++;
*tx = bin_hex1( tc.b.b2 );        tx++;
*tx = bin_hex1( tc.b.b3 >> 4 );   tx++;
*tx = bin_hex1( tc.b.b3 );        tx++;
```

16進数表示を行うやり方は、バイト単位で データを取り出し16進数 2桁を表示しています。

左の上側が リトルエンディアン仕様です。

左の下側が ビッグエンディアン仕様です。

b.b0 ～ b.b3 の並びが上側と 下側で 逆になっているのが、分かると思います。これで リトルエンディアン仕様とビッグエンディアン仕様のバイト位置の取り出し方が 分かったと思います。

最後に、1本のソースにて ビッグエンディアンと リトルエンディアンの 両方の CPUに対応するための コーディングを紹介します。 ソース先頭で 以下の記述を入れます。 下のコメントの通りですが、#define LITTLE_EN を 宣言するか しないかで設定します。

```
// ★  使用するCPUの  バイトオーダーの切り替え
// ================================================================
//      リトルエンディアンの場合：以下の #define  LITTLE_EN を 宣言する
//      ビッグエンディアンの場合：以下の #define  LITTLE_EN を コメントに する
// ----------------------------------------------------------------
#define  LITTLE_EN   // ＜ 現在 リトルエンディアン ＞
// ================================================================
```

今回、整数値 <-> Hex文字列変換箇所は ソース内に 6ヶ所あります。
6か所とも表示すると長くなるので、先ほど long -> 16進 8文字表示のソースを表示したので、次ページでは、1ヶ所 long -> 16進 6文字表示のソースをどのように バイトオーダーを切り替えているかを 表示します。

```
    tc.dw = dw;
//***  ★ バイトオーダー切り替え ②  ***
#ifdef LITTLE_EN
    *tx = bin_hex1( tc.b.b2 >> 4 );   tx++;  // リトルエンディアン
    *tx = bin_hex1( tc.b.b2 );        tx++;
    *tx = bin_hex1( tc.b.b1 >> 4 );   tx++;
    *tx = bin_hex1( tc.b.b1 );        tx++;
    *tx = bin_hex1( tc.b.b0 >> 4 );   tx++;
    *tx = bin_hex1( tc.b.b0 );        tx++;
#else
    *tx = bin_hex1( tc.b.b1 >> 4 );   tx++;  // ビッグエンディアン
    *tx = bin_hex1( tc.b.b1 );        tx++;
    *tx = bin_hex1( tc.b.b2 >> 4 );   tx++;
    *tx = bin_hex1( tc.b.b2 );        tx++;
    *tx = bin_hex1( tc.b.b3 >> 4 );   tx++;
    *tx = bin_hex1( tc.b.b3 );        tx++;
#endif
    *tx = NULL;
```

バイトオーダーの切り替えは、
#ifdef LITTLE_EN
#else
#endif
の コンパイル制御にて 行います。

これは、LITTLE_ENが 宣言されていると、リトルエンディアンのコードのみ生成されます。

逆に LITTLE_ENが 宣言されてないと、ビッグエンディアンのコードのみ生成されます。