FlashSta.exe の書き込み速度の高速化

正確には、パソコン　R8Cマイコン間の通信速度の高速化となります。 直接的に プログラム書き込み用フラッシュROMの書き込み速度が遅い訳ではなく、パソコン　R8Cマイコン間の通信速度が、デフォルトで 9600 bps で　データ通信速度が 遅いのです。 たまたま　R8C前のM16C時代の FlashSta.exeの 説明書を見つけました。 その説明書を見るとボーレイトの設定ダイアログが あるので、探してみたらありました。 私は、57600bpsで試してみましたが、問題なく書き込めました。 非常に速い通信速度で行うと、エラー　あるいは動作が不安定になる恐れが あるらしいです。書き込みデータは 101 bz_sound.motファイル 29,642byteです。

9600bps：　76秒です。

57600bps：　16秒です。 4.75倍 速いです。

57600bps / 9600bps は、6 になるのに4.75倍なんだと思われる方もいるかもしれません。

プログラム書き込み時、R8Cマイコンが　やる仕事は、通信だけではなく、受信したデータをプログラムフラッシュに 書き込む仕事も行います。

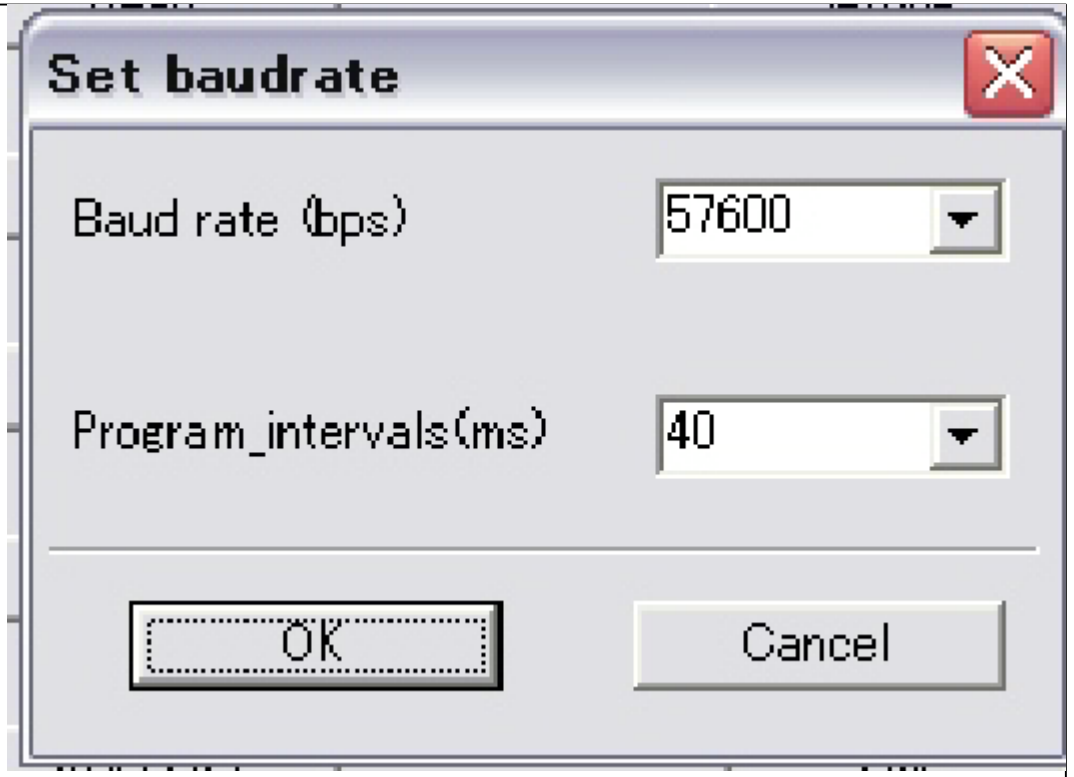
このプログラムフラッシュにデータを書き込む際は、RAM書き込みのように単純ではなく

一連の書き込みシーケンスがあると思います。その関係で、書き込みには、ちょっと手間がかかる(時間が かかる)のです。

通信速度は、最大　115200bpsですが、2番目の 57600bpsが、安全と思います。

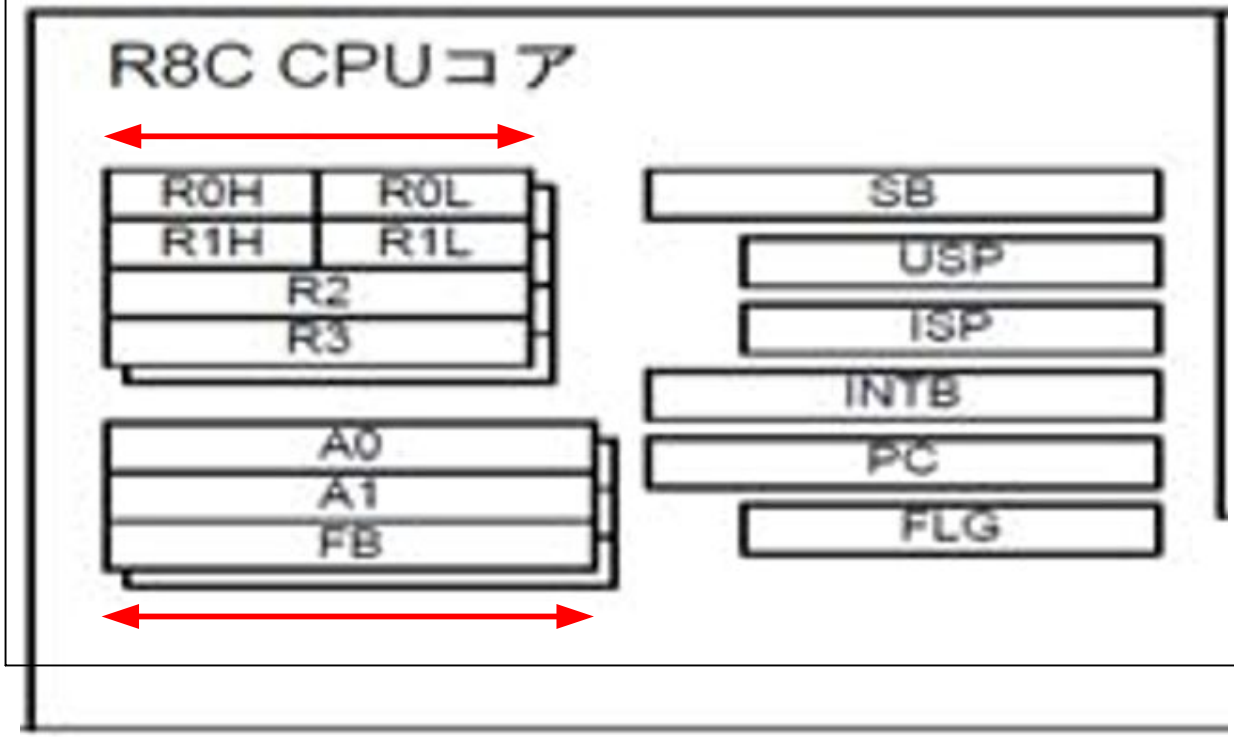
では、設定操作の動画を、お見せします。

先ほどの、ボーレイト設定のダイアログですが、下のパラメータは、Program_intervals(ms)と、書いてあります。 これは、プログラムフラッシュに書き込む、タイミング調整と思われますので、下手に変更すると プログラムフラッシュの 書き込み不良の原因に なりかねません。
という事で、変更しないで下さい。



64Kbyteを 越えるメモリのアクセス

この件については、前々から気になっては いたのですが、以前の 105の動画で、インターネットから画像コピーした R8CマイコンのCPUコアのレジスタ構成です。 何故か R0H, R0L ～ R3 の四角の横の長さに比べ、A0, A1, FB の 横の長さが 長く表示してあります。PCと INTB 以外は、16 bit のはずです。 この図は、おかしいぞ。？



という事で、R8Cマイコンの ハードウェア データシートの レジスタ構成図と比べると 明らかに レジスタ長の 図形上の表示が、A0、A1、FB、SB の長さが、間違いです。 繰り返しますが、PCと INTB 以外は 16bit です。

という事で、左の図は 間違いあるいは紛らわしい表示のR8Cマイコンのレジスタ表示の図となります。

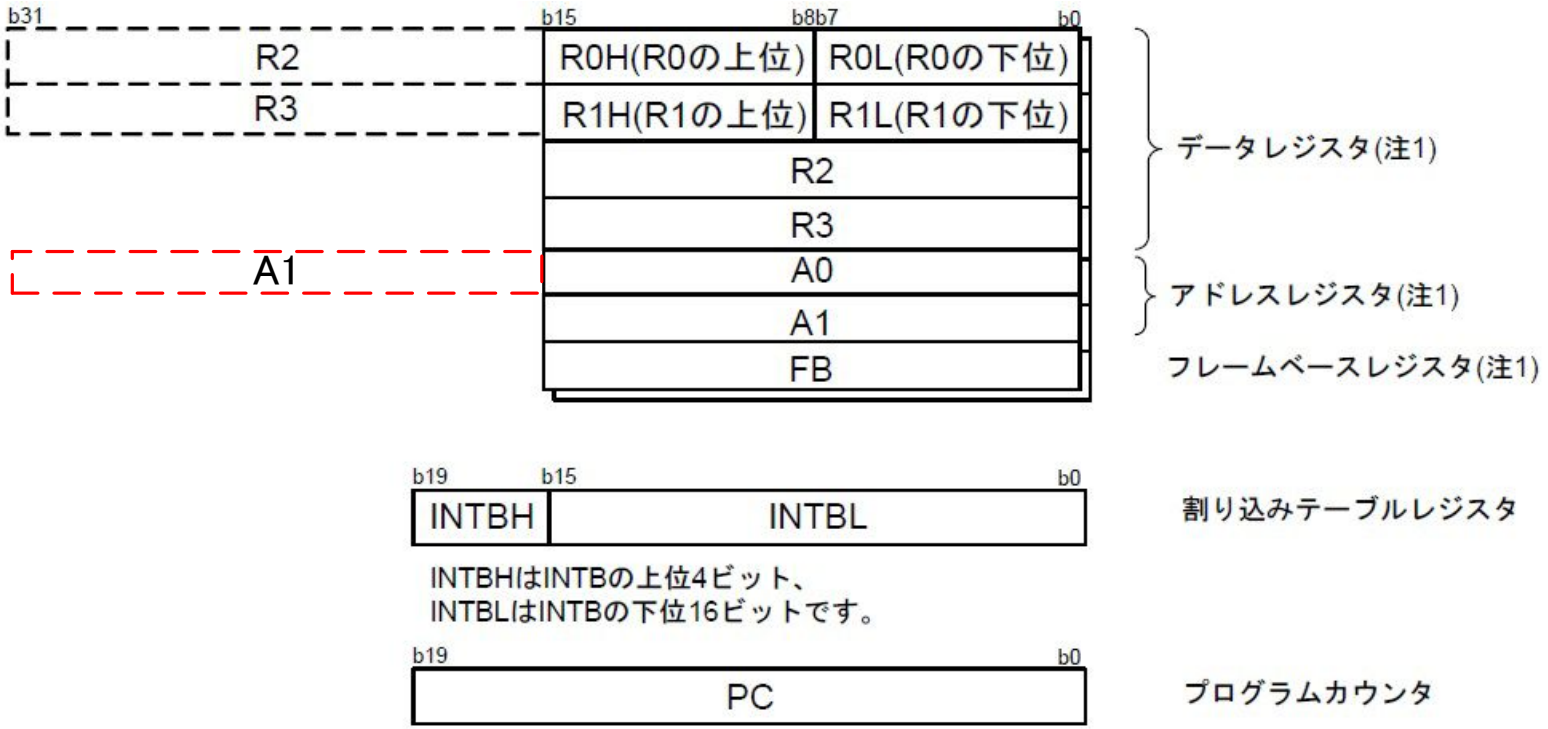
この図を よく確認せずに、使用した私も 間違った事を した事になります。

申し訳ありませんでした。

この件にてレジスタ長の 再確認等を、行った関係で、左側レジスタセットは 全て 16 bit として 再確認しましたが

ここで、A0、A1に関して 新たな発見がありました。

ルネサスのハードウェアデータシートのレジスタ構成図



R0レジスタは、R2レジスタを 上位WORDとして連結して、32 bit レジスタとして扱う事が出来ます。 R3 レジスタと R1レジスタも 連結して 32 bit レジスタとして扱う事が出来ます。 元の図には書いて無かったので、私が A0 レジスタの左に 赤の点線で A1 レジスタを 連結する 図を 追加しました。 これにより、A1 : A0の 32 bitの ポインタとして扱えます。

図では、示されて無かったのですが、単独の A0、A1 ポインタレジスタの説明欄にて、連結して 32 bit のポインタとして使える事が、目立たない文章で 書いてありました。

よって 32bit のポインタとして使えるのは、間違いないと思います。

但し、PC(プログラムカウンタ)が、20bit なので、レジスタ メモリ間のアドレスバスラインは あったとしても 20bit 以内と思います。

具体的に　64Kbyteを超えるアドレス空間をどのようにアクセスするのか。？

PCは、20bitで　64Kbyteを超えるメモリ領域にあるサブルーチンを　呼びだす事は可能ですが、64Kbyteを超えるメモリ領域に(プログラム フラッシュ)エリアに、プログラムを書き込む時どうしているのだろうか。？　プログラムを書き込む時は、書き込むプログラムは、データとして扱うので、16bit を 超えるポインタレジスタが ないと　書き込めないはず。　と、前から不思議に思っていました。　謎が解けました。

A1 A0 の ポインタレジスタペアを　扱うのは アセンブラで プログラムを作成するしか手段が無いと思います。　最初 MOV命令で、オペランドを、[A1A0]で、指定してダメで、[A1： A0]でもダメ　[A1-A0]でダメ、[A1 A0]でダメ、[A1/A0]でもダメで、どのようにオペランドを 指定するんだろうと、しばらく悩んでいました。

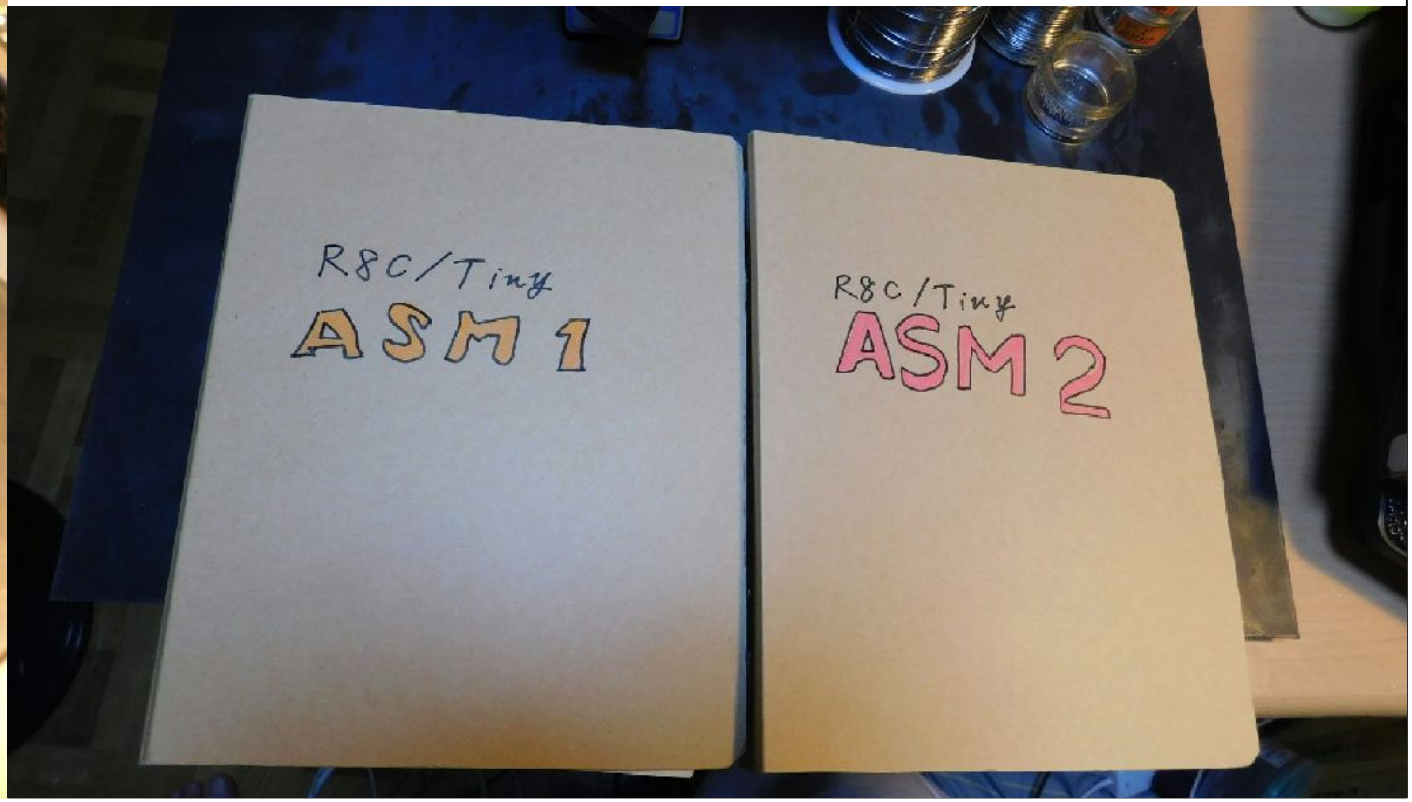
これは、MOV命令では、使えなくて特別なオペコードが、あるのではないかという事に なりました。　で、オペランドに　A1A0が使えるオペコードという事で、ルネサスR8C アセンブラの　資料を探してみたら、有りました。　LDE命令、STE命令です。

このアセンブラの資料は、私の動画105にて、ダウンロードコーナーの ZIPファイル内にある「R8C_アセンブラ_リンカ資料.pdf」です。　ややページ数があるので、2ページを A4に縮小印刷して、百均のボール紙バインダーに挟みました。

ちなみに、バインダーも、A4サイズの物を、ハサミで、周りを切って A5サイズにして、コピー用紙を2つ折りして挟みました。2冊になりました。

R8Cマイコン アセンブラ資料の印刷物

この資料から、どうやって目的の命令コードを見つけ出したかの手順を、動画でお見せします。 資料で、見つけたい機能を 探し出す見方が、何かの役に立てばと思い、動画にしてみました。
通常、使うのは 殆どが 前半の ASM1だけです。



64Kbyteを 超える アクセス サブルーチン 1/3

```
; ************************************************************
; **  ○  64Kbyteのアドレスレンジを超えるメモリアクセス      **
; **      先頭アドレスの 設定処理                            **
; ** ------------------------------------------------------ **
; **  引数  ( 上位 4bit , 下位 16bit )                       **
; **  R1  :  上位 メモリアドレス ( b19 ～ b16 )              **
; **  R2  :  下位 メモリアドレス ( b15 ～  b0 )              **
; **  関数値 :    Void                                       **
; ************************************************************
    .glb    _set_extadr
_set_extadr:
           mov.b   r1l, eadr_hi        ; b19 ～ b16  R1L --> eadr_hi
           mov.w   r2,  eadr_low       ; b15 ～  b0  R2 --> eadr_low
           rts

～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～

;  ***  データ セクション
; ------------------------------------------------------
eadr_hi:    .blkb 1             ; 拡張アドレス上位  4 bit
eadr_low:   .blkw 1             ; 拡張アドレス下位 16 bit
```

64Kbyteを 超える アクセス サブルーチン 1/3 です。

.glbは グローバル宣言です。
.glb が無いと ローカル宣言で そのソースファイル内だけで有効な名前というか ラベルに なります。

_set_extadr: は ラベルです。
この場合、Cから呼び出される関数名になります。C側の コーディングでは 先頭の _ は 必要ありません。eadr_hi:、eadr_low: も ラベルです。 .blkb は byte変数宣言です。 右の 1 は 1 個を意味します。.blkw は word変数宣言です。因みに mov.b は、左から右への byte幅の転送命令です。
mov.wは、word幅の 転送命令です。 rts は リターン命令です。

64Kbyteを 超える アクセス サブルーチン 2/3

```
; ********************************************************
; **  ○  64Kbyteのアドレスレンジを超えるメモリ読み出し      **
; ** ---------------------------------------------------- **
; **  引数  ( 無し )                                       **
; **  関数値 :    byte データ                              **
; ********************************************************
    .glb    _get_ead_byte
_get_ead_byte:
        push.w   a0             ; A0 ポインタレジスタ退避
        push.w   a1             ; A1 ポインタレジスタ退避

       mov.w    eadr_hi, a1     ; A1 = メモリ上位アドレス  4 bit
       mov.w    eadr_low, a0     ; A0 = メモリ下位アドレス 16 bit
       lde.b    [a1a0], r0l     ; R0L = 読み出したデータ 関数値として返す
       inc.w    a0               ; A0 = A0 + 1
        mov.w    a0, eadr_low     ; eadr_low に A0値を 格納

        pop.w    a1              ; A1 ポインタレジスタ復帰
        pop.w    a0              ; A0 ポインタレジスタ復帰
        rts                      ; リターン
```

64Kbyteを 超える アクセス サブルーチン 2/3 です。

関数名は get_ead_byteです。
アセンブラに関わる追加の説明は、ラベルの後ろに付く : は、ラベルを 宣言する物です。 ; は、そこから右が、コメントである事を示します。
push.w命令は ワード単位のレジスタを スタックに積み上げる命令です。
pop.w命令は ワード単位でスタックに積み上げた値をレジスタに戻す命令です。
inc.w命令は、ワード単位のレジスタ、または変数を +1 する命令です。 そして、今回 lde.b [a1a0],r0l 命令を初めて 使用しました。

64Kbyteを 超える アクセス サブルーチン 3/3

```
; ********************************************************************
; **  ◆  64Kbyteのアドレスレンジを超えるメモリ書き込み          **
; ** ------------------------------------------------------------ **
; **  引数：  ( Byte データ ( R1 ) )                                **
; **  関数値：  無し Void                                           **
; ********************************************************************
    .glb    _put_ead_byte
_put_ead_byte:
        push.w   a0             ; A0 ポインタレジスタ退避
        push.w   a1             ; A1 ポインタレジスタ退避

        mov.w   eadr_hi, a1     ; A1 = メモリ上位アドレス  4 bit
        mov.w   eadr_low, a0    ; A0 = メモリ下位アドレス 16 bit
        ste.b   r1l, [a1a0]     ; R1 = 書き込むデータ
        inc.w   a0              ; A0 = A0 + 1
        mov.w   a0, eadr_low    ; eadr_low に A0値を 格納

        pop.w   a1              ; A1 ポインタレジスタ復帰
        pop.w   a0              ; A0 ポインタレジスタ復帰
        rts                     ; リターン
```

64Kbyteを 超える アクセス サブルーチン 3/3 です。

関数名は put_ead_byteです。

lde.b [a1a0],r0l 命令が
ste.b r1l,[a1a0] 命令に
変った以外は、前の処理と同じです。

今回のアセンブラルーチンは、 一応 アセンブルしてエラーが、出ない事は確認してます。

後は、実際動かしてみないと 分からないですね。

視聴者の皆様方 先着16名さまに 基板を 送ります。

前回の基板作成動画の企画の続きで
今回の動画のコメント欄では 9番から16番の8名様を 先着順に受け付け基板を 発送する予定です。

申し訳ありませんが、発送料が 有料です。
ゆうパケット 250円 ＋ CD-R等入れるプチプチ封筒 50円 で、計: 300円です。
ゆうパケットは、追跡番号が 付いてます。

手続きの順番:

① 「R8C/Mの小基板、送って下さい。」と
コメントに 書き込んで下さい。

② コメント書き込み後、私の所にメールが届くので書き込み時刻が 分かります。

③ 私が、先着番号と、アルファベット3文字を
コメント欄の返信に 書き込みます。

④ 107動画の ダウンロードコーナーに、私の
メールアドレスを 書いてますので、
1. YouTubeの チャンネル名
2. 先着番号と アルファベット3文字
3. 基板 送り先 郵便番号と 住所
4. 送り先 名前
以上を、メールに記載して 私のメール
アドレスに送って下さい。

⑤ 送料送り先の ゆうちょ銀行の 記号、番号、
店番、口座番号を、追記してメールを 返信します。ゆうちょATMの場合 送金料金 100円です。最初の手続きが ちょっと面倒ですが
ゆうちょダイレクトで 送金すると、各月の
5回目までの送金は、無料との事です。

コメント欄には、絶対に個人情報は書かないで下さい。

⑥ 送る準備は、事前にしておきますが、発送料 300円の入金が 確認されたら、基板を送ります。
その後、メールにて、ゆうパケットの追跡番号を お知らせします。

コメント、メールでのやり取りの例を示します。

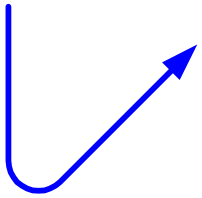
クマモン／コメント

R8C/Mの小基板、送って下さい。

道草職人Take／コメント

1、ABC （ 先着番号と アルファベット3文字 ）

アルファベット3文字は無いとは 思いますが、割り込み偽装メール対策です。



クマモン／メール

1. クマモン
2. 1、 ABC
3. 〒860-8570
   熊本県 熊本市 中央区 県庁内
4. クマモン

道草職人Take／メール

1. クマモン
2. 1、 ABC
3. 〒862-8570
   熊本県 熊本市 中央区 県庁内
4. クマモン
5. ゆうちょ銀行の 記号、番号
6. ゆうちょ銀行の 店番、口座番号

道草職人Take

入金確認後、基板発送、メールで追跡番号通知

ダウンロードしたPDFファイルにも、このページは入ってます。