2003年3月30日

Ｈ８シリーズ用
ＩＯＣＳ関数マニュアル

目　次

目　次

目　　次

分類番号：01-01	ソースファイル： Rst_Int.mar

機能：

CPUの割り込みマスクを解除し、割り込みを許可する。

関数プロトタイプ宣言：

```
void  enable_interupt( void );
```

説明：

分類番号：01-02	ソースファイル： Rst_Int.mar

機能：

CPUの割り込みマスクをセットし、割り込みを禁止する。

関数プロトタイプ宣言：

```
void  disable_interupt( void );
```

説明：

分類番号：01-03	ソースファイル： Rst_Int.mar

機能：

CPUの動作モードの取出し

関数プロトタイプ宣言：

```
int     get_cpu_mode( void );
```

説明：

関数値： 1 ～ 7 の値を返す。
CPUの動作モードについては、Ｈ８のマニュアルを参照の事。

分類番号：01-04	ソースファイル： Rst_Int.mar

機能：

トラップを生成する。（ ソフトによる割り込み処理 呼び出し ）

関数プロトタイプ宣言：

```
void    trap_0( void );
```

説明：

Ｈ８シリーズＣＰＵでは、命令によりトラップを生成する事が出来る。
トラップは、４つのベクトルが用意されている。
よって、 trap_0() 以外に trap_1(), trap_2(), reap_3() がある。

割り込みベクトルとの関係は

trap_0()	8	h'0020
trap_1()	9	h'0024
trap_2()	10	h'0028
trap_3()	11	h'002C

分類番号：01-05	ソースファイル： Rst_Int.mar

機能：

ウォッチドッグタイマーの起動

関数プロトタイプ宣言：

```
void    setup_wdt( void );
```

説明：

ＨＳＣＰＵのＷＤＴを起動する。
この関数を呼び出した後は、周期的にドッグタイマーを refresh_wdt(); にて リフレッシュしなければ、ＣＰＵリセットが発生する。
ドッグタイマのタイムアップ時間は、ＣＰＵクロックに依存し

16[Mhz]で　1/16 [秒]（ 62.5[ms] ）
20[Mhz]で　1/20 [秒]（ 50[ms] ）
25[Mhz]で　1/25 [秒]（ 40[ms] ） となる。

これより短い周期で、リフレッシュ動作を続けなければＣＰＵリセットが発生する。

分類番号：01-06	ソースファイル： Rst_Int.mar

機能：

ウォッチドッグタイマーの停止

関数プロトタイプ宣言：

```
void    stop_wdt( void );
```

説明：

ＨＳＣＰＵのＷＤＴを停止する。

分類番号：01-07	ソースファイル： Rst_Int.mar

機能：

ウォッチドッグタイマーのリフレッシュを行う

関数プロトタイプ宣言：

```
void    refresh_wdt( void );
```

説明：

ドッグタイマのタイムアップ時間は、ＣＰＵクロックに依存し

16[Mhz]で　1/16 [秒]（ 62.5[ms] ）
20[Mhz]で　1/20 [秒]（ 50[ms] ）
25[Mhz]で　1/25 [秒]（ 40[ms] ）　となる。

これより短い周期で、リフレッシュ動作を続けなければＣＰＵリセットが発生する。

16桁、2行のＬＣＤ表示を行ったらドッグタイマーにひっかかった。
やや、時間のかかる処理には、気を付ける事。

分類番号：01-08	ソースファイル： Rst_Int.mar

機能：

ＣＰＵのリセット要因を調べる。

関数プロトタイプ宣言：

```
int     check_reset( void );
```

説明：

ＣＰＵのリセット要因が、リセット端子によるものか、ＷＤＴのタイムアップにより発生したものであるかを調べるための関数。

関数値：　= 0 ： リセット端子によるもの
= 80H ： ＷＤＴのタイムアップによるもの

分類番号：02-01	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.0（ SCI 0 ） のオープン
この関数の呼び出しにより、RS-232C ch.0 が 使用出来るようになる。

関数プロトタイプ宣言：

```
void  rs0_open( unsigned int bps, char *fmt );
```

説明：

引数 1: bps は ボーレイトの値（ 以下の値を取る ）
（ 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 31250, 38400 ）

引数 2: fmt は シリアル通信の語構成を指定する文字列（ ３文字にて構成される ）

１文字目： パリティの指定（ N、E、O ）のどれか１文字
N = Nonパリティ 、E = Even パリティ 、O = Odd パリティ

２文字目： データ語長（ 7、8 ）のいずれか１文字
7 = データ長 7 bit 、 8 = データ長 8 bit

３文字目： ストップビット長（ 1、2 ）のいずれか１文字
1 = ストップ長 1 bit 、 2 = ストップ長 2 bit

例） rs0_open(9600, "N81");

分類番号：02-02	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.0（ SCI 0 ）のクローズ
この関数の呼び出しにより、RS-232C ch.0 は、停止状態となる。

関数プロトタイプ宣言：

```
void   rs0_close( void );
```

説明：

分類番号：02-03	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.0 に対し １文字（ 1 byte ）送信を行う。

関数プロトタイプ宣言：

```
void    rs0_send( char dt );
```

説明：

引数 1： dt 送信を行う１文字（ 1 byteの値 0 ～ 255 ）

分類番号：02-04	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.0 の受信バッファ内に蓄積されている文字数の取り出しを行う。

関数プロトタイプ宣言：

```
int     rs0_count( void );
```

説明：

関数値： 0 ～ バッファ内最大蓄積個数（ デフォルトバッファサイズ 80 byte ）
0 は 受信文字無しを意味する。

参考：

受信バッファサイズは、define_H8.src 内に定義されているので、これを変更し
Rst_Intを再構築すれば、バッファサイズは変更可能である。
しかし現在、カウンタが byte変数なので　最大　255 byte までである。

分類番号：02-05	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.0 の受信バッファから、1文字 取り出す。

関数プロトタイプ宣言：

```
int     rs0_recv( void );
```

説明：

関数値： 受信文字コード（ 0 ～ 255 ）を返す。
受信バッファが 空の場合、-1（ 0xFFFF ） を 返す。

分類番号：02-06	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.0 へ 文字列を送信する。

関数プロトタイプ宣言：

```
int   rs0_prn( char *txt );
```

説明：

引数１： txt ： 送信する文字列

分類番号：02-07	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.0 へ文字列を送信する。
その後、CrLfコード（改行コード）を送信する。

関数プロトタイプ宣言：

```
int     rs0_print( char *txt );
```

説明：

引数１： txt 送信する文字列

分類番号：02-08	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.0 から１行分の 文字列を取り出す。
エコーバックは行わない。（ デリミッタは、Cr コード ）

関数プロトタイプ宣言：

```
int  rs0_input( char *buf, int lim );
```

説明：

引数１： buf ： １行分の文字列バッファ先頭アドレス

引数２： len ： 最大 文字格納個数（ バッファサイズ ）

参考： Crコードは、Nullコードに置きかえられる。

分類番号：02-09	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.0 から１行分の 文字列を取り出す。
エコーバックを 行う。（ デリミッタは、Cr コード ）
[BackSpace] Keyに 対応している。（ Windows / Hyper Terminalに 対応 ）

関数プロトタイプ宣言：

```
int     rs0_input_echo( char *buf, int lim );
```

説明：

引数１： buf ： １行分の文字列バッファ先頭アドレス

引数２： len ： 最大 文字格納個数（ バッファサイズ ）

参考： Crコードは、Nullコードに置きかえられる。

分類番号：02-10	ソースファイル： Rst_Int.mar

機能：

関数プロトタイプ宣言：

説明：

分類番号：02-11	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.1（ SCI 1 ） のオープン
この関数の呼び出しにより、RS-232C ch.1 が 使用出来るようになる。

関数プロトタイプ宣言：

```
void  rs1_open( unsigned int bps, char *fmt );
```

説明：

引数 1: bps は ボーレイトの値（ 以下の値を取る ）
（ 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 31250, 38400 ）

引数 2: fmt は シリアル通信の語構成を指定する文字列（ ３文字にて構成される ）

１文字目： パリティの指定（ N、E、O ）のどれか１文字
N = Nonパリティ 、E = Even パリティ 、O = Odd パリティ

２文字目： データ語長（ 7、8 ）のいずれか１文字
7 = データ長 7 bit 、 8 = データ長 8 bit

３文字目： ストップビット長（ 1、2 ）のいずれか１文字
1 = ストップ長 1 bit 、 2 = ストップ長 2 bit

例） rs1_open(9600, "N81");

分類番号：02-12	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.1（ SCI 1 ）のクローズ
この関数の呼び出しにより、RS-232C ch.1 は、停止状態となる。

関数プロトタイプ宣言：

```
void   rs1_close( void );
```

説明：

分類番号：02-13	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.1 に対し 1文字（ 1 byte ）送信を行う。

関数プロトタイプ宣言：

```
void    rs1_send( char dt );
```

説明：

引数 1： dt 送信を行う1文字（ 1 byteの値 0 ～ 255 ）

分類番号：02-14	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.1 の受信バッファ内に蓄積されている文字数の取り出しを行う。

関数プロトタイプ宣言：

```
int     rs1_count( void );
```

説明：

関数値： 0 ～ バッファ内最大蓄積個数（ デフォルトバッファサイズ 80 byte ）
0 は 受信文字無しを意味する。

参考：

受信バッファサイズは、define_H8.src 内に定義されているので、これを変更し
Rst_Intを再構築すれば、バッファサイズは変更可能である。
しかし現在、カウンタが byte変数なので 最大 255 byte までである。

分類番号：02-15	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.1 の受信バッファから、1 文字 取り出す。

関数プロトタイプ宣言：

```
int     rs1_recv( void );
```

説明：

関数値： 受信文字コード（ 0 ～ 255 ）を返す。
受信バッファが 空の場合、-1（ 0xFFFF ） を 返す。

分類番号：02-16	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.1 へ 文字列を送信する。

関数プロトタイプ宣言：

```
int   rs1_prn( char *txt );
```

説明：

引数１： txt ： 送信する文字列

分類番号：02-17	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.0 へ文字列を送信する。
その後、CrLfコード（改行コード）を送信する。

関数プロトタイプ宣言：

```
int     rs1_print( char *txt );
```

説明：

引数１： txt 送信する文字列

分類番号：02-18	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.0 から１行分の 文字列を取り出す。
エコーバックは行わない。（ デリミッタは、Cr コード ）

関数プロトタイプ宣言：

```
int  rs1_input( char *buf, int lim );
```

説明：

引数１： buf ： １行分の文字列バッファ先頭アドレス

引数２： len ： 最大 文字格納個数（ バッファサイズ ）

参考： Crコードは、Nullコードに置きかえられる。

分類番号：02-19	ソースファイル： Rst_Int.mar

機能：

RS-232C ch.0 から１行分の 文字列を取り出す。
エコーバックを 行う。（ デリミッタは、Cr コード ）
[BackSpace] Keyに 対応している。（ Windows / Hyper Terminalに 対応 ）

関数プロトタイプ宣言：

```
int     rs1_input_echo( char *buf, int lim );
```

説明：

引数１： buf ： １行分の文字列バッファ先頭アドレス

引数２： len ： 最大 文字格納個数（ バッファサイズ ）

参考： Crコードは、Nullコードに置きかえられる。

分類番号：02-20	ソースファイル： Rst_Int.mar

機能：

関数プロトタイプ宣言：

説明：

分類番号：03-01	ソースファイル： Rst_Int.mar

機能：

インターバルタイマー（ ITU 0 ）の初期化

関数プロトタイプ宣言：

```
void    init_timer( void );
```

説明：

この関数呼び出しにより、 10[ms]周期のインターバルタイマーが起動し
経過時間の監視用途に
set_timer1() 、 get_timer1()
set_timer2() 、 get_timer2()

LED点滅制御に
set_led_ontime() 、 flash_led()

及び、追加タイマー割込み処理
enter_tm_proc() も利用可能となる。

分類番号：03-02	ソースファイル： Rst_Int.mar

機能：

タイマー処理 減算カウンタ値 設定

関数プロトタイプ宣言：

```
void  set_timer1( unsigned char cnt );
```

説明：

引数１： cnt = 0 ～ 255 の値を設定可能

設定後、インターバルタイマの割り込み処理にて、
10[ms]毎に デクリメントされていく。
0 になったらデクリメント動作は停止する。

そして、このデクリメントされていくカウント値を
読み出すのに、get_timer1() を使用する。

分類番号：03-03	ソースファイル： Rst_Int.mar

機能：

減算カウンタ値 読み出し

関数プロトタイプ宣言：

```
int     get_timer1( void );
```

説明：

set_timer1() にて設定されたカウント値が、10[ms]毎に デクリメントされる事になる。
この値を読み出すことにより、経過時間を確認する事が出来る。
0 になったらデクリメント動作は停止する。

関数値： 現在の減算カウンタ値を読み出す。

分類番号：03-04	ソースファイル： Rst_Int.mar

機能：

タイマー処理 加算カウンタ値 設定

関数プロトタイプ宣言：

```
void    set_timer2( unsigned char cnt );
```

説明：

引数１： cnt = 0 ～ 255 の値を設定可能

設定後、インターバルタイマの割り込み処理にて、
10[ms]毎に インクリメントされていく。
255 になったら、次は 0 に戻る。

そして、このインクリメントされていくカウント値を
読み出すのに、get_timer2() を使用する。

分類番号：03-05	ソースファイル： Rst_Int.mar

機能：

加算カウンタ値 読み出し

関数プロトタイプ宣言：

```
int     get_timer2( void );
```

説明：

set_timer2() にて 設定されたカウント値が、10[ms]毎に インクリメントされる事になる。
この値を読み出すことにより、経過時間を確認する事が出来る。

関数値： 現在の加算カウンタ値を読み出す。

分類番号：03-06	ソースファイル： Rst_Int.mar

機能：

[ms]単位の Wait（ 時間待ち ）を作り出す。
ソフト的に空ループを回しているだけなので時間精度は、大雑把なものである。

関数プロトタイプ宣言：

```
void    wait_milli( unsigned int cnt );
```

説明：

引数１： cnt 0 ～ 65,535 の値（ 単位 [ms] ）をとる。

呼び出したら、その時間が経過しないと戻ってこない。

分類番号：03-07	ソースファイル： Rst_Int.mar

機能：

追加タイマー割込み処理の登録

関数プロトタイプ宣言：

```
void    enter_tm_proc( void* );
```

説明：

1/100[秒] タイマー割込み処理にて、呼び出してほしい処理（ 関数 ）を登録する。
登録する関数は、Cの関数でも、アセンブラの関数でもかまわない。
制限としては、必ず処理が 5[ms]以内に終了する事。
登録するタイミングは、タイマーを初期化する 前でも 後でもよい。
このルーチンを使用するプログラマは、割込み処理を熟知しているものとします。

引数１： 本来は、void* ではない。 関数の実行開始アドレスを渡す。
しかし、Evaluation software製 Cコンパイラの場合は このような宣言をするしかなかった。
test() という関数を登録する場合は

exter_tm_proc(*test); とする。

分類番号：03-08	ソースファイル： Rst_Int.mar

機能：

追加タイマー割込み処理を停止させる。

関数プロトタイプ宣言：

```
off_tm_proc( void );
```

説明：

enter_tm_proc() にて登録したタイマー割り込み処理を停止させます。
再度、追加タイマー割込み処理を 走らせる場合は、
enter_tm_proc()を再度呼び出す事で対応して下さい。

分類番号：04-01	ソースファイル： hex_bin.mar

機能：

1byte（ 下位 4 bit ）バイナリ値を 2桁 16進数文字列に変換し、
与えられた文字列バッファ内文字列の後ろに 追加する。

関数プロトタイプ宣言：

```
char*  b_hex1_cat( char b, char *buf );
```

説明：

引数１（入力）： b : 1 byteバイナリ値（ 下位 4 bit が 対象 ）

引数２（出力）： buf : 格納文字列（ 格納済みの文字列の後ろに追加される。 ）

上位4bit、下位4bitの順に '0'～'9'、'A'～'F'の文字にて表現される
16進文字列２文字として格納される。

関数値： 文字列の終端（ Null位置 ）のポインタ

分類番号：04-02	ソースファイル： hex_bin.mar

機能：

1byte（ 下位 4 bit ）バイナリ値を 2桁 16進数文字列に変換し
文字列バッファに格納する。

関数プロトタイプ宣言：

```
char*  b_hex1( char b, char *buf );
```

説明：

引数１（入力）： b : 1 byteバイナリ値（ 下位 4 bit が 対象 ）

引数２（出力）： buf : 格納文字列

上位4bit、下位4bitの順に '0'～'9'、'A'～'F'の文字にて表現される
16進文字列２文字として格納される。

関数値： 文字列の終端（ Null位置 ）のポインタ

分類番号：04-03	ソースファイル： hex_bin.mar

機能：

１byte バイナリ値を ２桁 16進数文字列に変換し、
与えられた文字列バッファ内文字列の後ろに 追加する。

関数プロトタイプ宣言：

```
char*   b_hex2_cat( char b, char *buf );
```

説明：

引数１（入力）：　b ： 1 byteバイナリ値

引数２（出力）：　buf ： 格納文字列（ 格納済みの文字列の後ろに追加される。 ）

上位4bit、下位4bitの順に '0'～'9'、'A'～'F'の文字にて表現される
16進文字列２文字として格納される。

関数値：　文字列の終端（ Null位置 ）のポインタ

分類番号：04-04	ソースファイル： hex_bin.mar

機能：

１byte バイナリ値を ２桁 16進数文字列に変換し
文字列バッファに格納する。

関数プロトタイプ宣言：

```
char*   b_hex2( char b, char *buf );
```

説明：

引数１（入力）：　b ： 1 byteバイナリ値

引数２（出力）：　buf ： 格納文字列

上位4bit、下位4bitの順に '0'～'9'、'A'～'F'の文字にて表現される
16進文字列２文字として格納される。

関数値：　文字列の終端（ Null位置 ）のポインタ

分類番号：04-05	ソースファイル： hex_bin.mar

機能：

１word バイナリ値を ４桁 16進数文字列に変換し、
与えられた文字列バッファ内文字列の後ろに 追加する。

関数プロトタイプ宣言：

```
char*  w_hex4_cat( int w, char *buf );
```

説明：

引数１（入力）： w ： 1 Word（ 16bit ）バイナリ値

引数２（出力）： buf ： 格納文字列（ 格納済みの文字列の後ろに追加される。 ）

最上位から最下位に向い、4bitずつ '0'～'9'、'A'～'F'の文字にて表現される
16進文字列４文字として格納される。

関数値： 文字列の終端（ Null位置 ）のポインタ

分類番号：04-06	ソースファイル： hex_bin.mar

機能：

１word バイナリ値を ４桁 16進数文字列に変換し
文字列バッファに格納する。

関数プロトタイプ宣言：

```
char*  w_hex4( int w, char *buf );
```

説明：

引数１（入力）： w ： 1 Word（ 16bit ）バイナリ値

引数２（出力）： buf ： 格納文字列

最上位から最下位に向い、4bitずつ '0'～'9'、'A'～'F'の文字にて表現される
16進文字列４文字として格納される。

関数値： 文字列の終端（ Null位置 ）のポインタ

分類番号：04-07	ソースファイル： hex_bin.mar

機能：

１dword バイナリ値を ６桁 16進数文字列に変換し、
与えられた文字列バッファ内文字列の後ろに 追加する。

関数プロトタイプ宣言：

```
char*   dw_hex6_cat( long l, char *buf );
```

説明：

引数１（入力）： w : 1 DoubleWord（ 32bit ）バイナリ値
変換範囲は、下位 24bitが対象となる。（ 上位 8bitは無視 ）

引数２（出力）： buf : 格納文字列（ 格納済みの文字列の後ろに追加される。 ）

最上位から最下位に向い、4bitずつ '0'～'9'、'A'～'F'の文字にて表現される
16進文字列６文字として格納される。

関数値： 文字列の終端（ Null位置 ）のポインタ

分類番号：04-08	ソースファイル： hex_bin.mar

機能：

１dword バイナリ値を ６桁 16進数文字列に変換し
文字列バッファに格納する。

関数プロトタイプ宣言：

```
char*   dw_hex6( long l, char *buf );
```

説明：

引数１（入力）： w : 1 DoubleWord（ 32bit ）バイナリ値
変換範囲は、下位 24bitが対象となる。（ 上位 8bitは無視 ）

引数２（出力）： buf : 格納文字列

最上位から最下位に向い、4bitずつ '0'～'9'、'A'～'F'の文字にて表現される
16進文字列６文字として格納される。

関数値： 文字列の終端（ Null位置 ）のポインタ

分類番号：04-09	ソースファイル： hex_bin.mar

機能：

１dword バイナリ値を ８桁 16進数文字列に変換し、
与えられた文字列バッファ内文字列の後ろに 追加する。

関数プロトタイプ宣言：

```
char*   dw_hex8_cat( long l, char *buf );
```

説明：

引数１（入力）：　w : 1 DoubleWord（ 32bit ）バイナリ値

引数２（出力）：　buf : 格納文字列（ 格納済みの文字列の後ろに追加される。 ）

最上位から最下位に向い、4bitずつ '0'～'9'、'A'～'F'の文字にて表現される
16進文字列８文字として格納される。

関数値：　文字列の終端（ Null位置 ）のポインタ

分類番号：04-10	ソースファイル： hex_bin.mar

機能：

１dword バイナリ値を ８桁 16進数文字列に変換し
文字列バッファに格納する。

関数プロトタイプ宣言：

```
char*   dw_hex8( long l, char *buf );
```

説明：

引数１（入力）：　w : 1 DoubleWord（ 32bit ）バイナリ値

引数２（出力）：　buf : 格納文字列

最上位から最下位に向い、4bitずつ '0'～'9'、'A'～'F'の文字にて表現される
16進文字列８文字として格納される。

関数値：　文字列の終端（ Null位置 ）のポインタ

分類番号：04-11	ソースファイル： hex_bin.mar

機能：

ポインタで指しているバイナリデータを、指定 byte数、16進文字列データに変換し
文字列バッファに 16進文字列を格納する。

関数プロトタイプ宣言：

```
char*   bin_hex_block( char *bbuf, char *hbuf, int len );
```

説明：

引数１（入力）：　bbuf　16進変換したい、バイナリデータが入った byte 配列

引数２（出力）：　hbuf　変換後の16進文字列を格納するバッファ

引数３（入力）：　len　変換したいバイナリデータの byte数
（ 変換した 16進文字列の文字数は２倍となる。 ）

関数値：　文字列の終端（ Null位置 ）のポインタ

分類番号：04-12	ソースファイル： hex_bin.mar

機能：

文字列バッファの内容が 16進文字列として正しいものかどうか確認して 判定値を返す。

関数プロトタイプ宣言：

```
int     check_hex_str( char *txt, int len );
```

説明：

引数１（入力）：　txt　16進文字列バッファ

引数２（入力）：　len　16進文字列 文字数

関数値：　　0 = 正常 、 1 = 不正16進文字列

チェック内容：

16進文字列を構成する各文字が、'0'～'9' 及び 'A'～'F' で有る事を確認する。

分類番号：04-13	ソースファイル： hex_bin.mar

機能：

16進１桁の文字列を 1 byte整数値（ 下位 4 bitが有効 ）に変換して返す。

関数プロトタイプ宣言：

```
char    hex1_b( char *txt );
```

説明：

引数１（入力）： txt 16進文字列を格納した文字配列（ 先頭２文字が変換対象となる。 ）
必ず、２文字は 格納しておく事、３文字目以降がある場合 無視する。

関数値（出力）： 1 byte のバイナリデータ

分類番号：04-14	ソースファイル： hex_bin.mar

機能：

16進２桁の文字列を 1 byte整数値に変換して返す。

関数プロトタイプ宣言：

```
char    hex2_b( char *txt );
```

説明：

引数１（入力）： txt 16進文字列を格納した文字配列（ 先頭２文字が変換対象となる。 ）
必ず、２文字は 格納しておく事、３文字目以降がある場合 無視する。

関数値（出力）： 1 byte のバイナリデータ

分類番号：04-15	ソースファイル： hex_bin.mar

機能：

16進４桁の文字列を 2 byte整数値に変換して返す。

関数プロトタイプ宣言：

```
int     hex4_w( char *txt );
```

説明：

引数１（入力）： txt 16進文字列を格納した文字配列（ 先頭４文字が変換対象となる。 ）
必ず、４文字は 格納しておく事、５文字目以降がある場合 無視する。

関数値（出力）： 1 Word（ 16 bit ） のバイナリデータ

分類番号：04-16	ソースファイル： hex_bin.mar

機能：

16進６桁の文字列を 4 byte整数値に変換して返す。

関数プロトタイプ宣言：

```
long    hex6_dw( char *txt );
```

説明：

引数１（入力）： txt 16進文字列を格納した文字配列（ 先頭６文字が変換対象となる。 ）
必ず、６文字は 格納しておく事、７文字目以降がある場合 無視する。

関数値（出力）： 1 Dword（ 32 bit ）の バイナリデータ
この場合、 最上位 byte は 0 固定となる。

分類番号：04-17	ソースファイル： hex_bin.mar

機能：

16進８桁の文字列を 4 byte整数値に変換して返す。

関数プロトタイプ宣言：

```
long    hex8_dw( char *txt );
```

説明：

引数１（入力）：　txt　16進文字列を格納した文字配列（ 先頭８文字が変換対象となる。 ）
必ず、８文字は 格納しておく事、９文字目以降がある場合 無視する。

関数値（出力）：　1 Dword（ 32 bit ）の バイナリデータ

分類番号：04-18	ソースファイル： hex_bin.mar

機能：

連続する16進文字列を、指定 byte数 バイナリデータに変換して格納する。

関数プロトタイプ宣言：

```
char*   hex_bin_block( char *hbuf, char *bbuf, len );
```

説明：

引数１（入力）：　hbuf　16進文字列を格納したバッファ

引数２（出力）：　bbuf　変換後のバイナリデータを格納するバッファ

引数３（入力）：　len　変換するバイナリデータの byte数
（ 16進文字列長は、２倍となる。）

16進文字列２文字を、順次 1 byteバイナリ値に変換し格納していく。

分類番号：05-01	ソースファイル： str_func.mar

機能：

文字列の英小文字を、大文字に変換する。

関数プロトタイプ宣言：

```
void    oomoji( char *txt );
```

説明：

引数１（入出力）：　txt　文字列を格納したバッファ

文字列内の　'a'～'z' を　'A'～'Z' に 置きかえる。

分類番号：05-02	ソースファイル： str_func.mar

機能：

文字列の英大文字を、小文字に変換する

関数プロトタイプ宣言：

```
void    komoji( char *txt );
```

説明：

引数１（入出力）：　txt　文字列を格納したバッファ

文字列内の　'A'～'Z' を　'a'～'z' に 置きかえる。

分類番号：05-03	ソースファイル： str_func.mar

機能：

指定 byte数、メモリ間コピーを行う。

関数プロトタイプ宣言：

```
void    mem_copy( void *dst, void *src, int len );
```

説明：

引数１（出力）：　dst　送り先バッファ

引数２（入力）：　src　送り元バッファ

引数３（入力）：　len　転送 バイト数

分類番号：05-04	ソースファイル： str_func.mar

機能：

文字列の文字数（ byte数 ）を返す。

関数プロトタイプ宣言：

```
int     str_len( char *txt );
```

説明：

引数１（入力）：　txt　文字列バッファ

関数値：　格納されている文字列の長さを返す。

分類番号：05-05	ソースファイル： str_func.mar

機能：

文字列のコピーを行う。

関数プロトタイプ宣言：

```
void    str_copy( char *dst, char *src );
```

説明：

引数１（出力）：　dst　送り先バッファ

引数２（入力）：　src　送り元バッファ

送り元文字列の Nullコードまでを 送り先へ転送する。

分類番号：05-06	ソースファイル： str_func.mar

機能：

文字数の制限付きで、文字列コピーを行う。

関数プロトタイプ宣言：

```
void    strn_copy( char *dst, char *src, int len );
```

説明：

引数１（出力）：　dst　送り先バッファ

引数２（入力）：　src　送り元バッファ

引数３（入力）：　len　文字列転送の制限バイト数
送り元が、制限バイト数より長ければ、制限バイト数まで転送し
送り先の終端に Nullを格納して終了する。

分類番号：05-07	ソースファイル： str_func.mar

機能：

文字列の連結を行う。

関数プロトタイプ宣言：

```
void    str_cat( char *dst, char *src );
```

説明：

引数１（入出力）：　dst　先頭文字列を格納したバッファ、このバッファの終端に
src の 文字列が 追加される。

引数２（入力）　：　src　追加する文字列

分類番号：05-08	ソースファイル： str_func.mar

機能：

文字列バッファを指定 byte数 Nullコードで埋める。（ バッファの初期化に用いる。）

関数プロトタイプ宣言：

```
void    nulls( char *buf, int len );
```

説明：

引数１（出力）：　buf　初期化するバッファ

引数２（入力）：　len　初期化する、バイト数

分類番号：05-09	ソースファイル： str_func.mar

機能：

文字列バッファ内に指定個数のスペースを入れ Null終端を格納する。

関数プロトタイプ宣言：

```
void    spaces( char *buf, int len );
```

説明：

引数１（出力）：　buf　スペースで初期化するバッファ。

引数２（入力）：　len　スペースコードの個数
格納されたスペースの終端に　Nullが追加される。

分類番号：05-10	ソースファイル： str_func.mar

機能：

文字列 先頭部分のスペースを取り去る。

関数プロトタイプ宣言：

```
void    fspdel( char txt );
```

説明：

引数１（入出力）：　txt　格納された文字列の先頭部分のスペースを取り除く。
取り除かれた分、その後の文字列は前詰めとなる。

分類番号：05-11	ソースファイル： str_func.mar

機能：

文字列、後方部分のスペースを取り去る。

関数プロトタイプ宣言：

```
void    bspdel( char *txt );
```

説明：

引数１（入出力）：　txt　文字列 後ろ部分のスペースコードを取り除く
スペースコード以外の最後の文字の後ろに Nullが格納される。

分類番号：05-12	ソースファイル： str_func.mar

機能：

文字列の比較を行う。

関数プロトタイプ宣言：

```
int     str_cmp( char *txta, char *txtb );
```

説明：

引数１（入力）：　txta　文字列A

引数２（入力）：　txtb　文字列B

関数値：

文字列Aと 文字列Bが 全く同じなら 0 が 戻る。

文字列A ＞ 文字列B なら　1 が 戻る。
例（ "0124", "0123" ）、("0123", "012")

文字列A ＜ 文字列B なら -1 が 戻る。
例（ "0123", "0124" ）、("012", "0123")

分類番号：05-13	ソースファイル： str_func.mar

機能：

長さ制限付きで、文字列の比較を行う。

関数プロトタイプ宣言：

```
int     strn_cmp( char *txta, txtb, int len );
```

説明：

引数１（入力）：　txta　文字列A

引数２（入力）：　txtb　文字列B

引数３（入力）：　len　　文字列 比較バイト数 制限値

関数値：

文字列Aと 文字列Bが 全く同じなら 0 が 戻る。

文字列A ＞ 文字列B なら　1 が 戻る。
例（ "0124", "0123" ）、("0123", "012")

文字列A ＜ 文字列B なら -1 が 戻る。
例（ "0123", "0124" ）、("012", "0123")

分類番号：05-14	ソースファイル： str_func.mar

機能：

文字列の中に、別の指定文字列が 部分文字列として含まれてないか調べる。

関数プロトタイプ宣言：

```
int     instr( char *txta, char txtb );
```

説明：

引数１（入力）：　txta　調べられる文字列

引数２（入力）：　txtb　調べる部分文字列

関数値：　txta内の txtb文字列が含まれるバイト位置（先頭 0 ）
含まれない場合、-1 を返す。

例）

```
 0 = instr( "ABCDEFG", "ABC" );

 2 = instr( "ABCDEFG", "CDE" );

-1 = instr( "ABCDEFG", "ACC" );
```

分類番号：05-15	ソースファイル： str_func.mar

機能：

文字列の先頭部分に、別の文字列が、部分文字列として含まれてないか調べる。

関数プロトタイプ宣言：

```
int     left_instr( char *txta, char *txtb );
```

説明：

引数１（入力）：　txta　調べられる文字列

引数２（入力）：　txtb　調べる部分文字列

関数値：　　部分文字列 txtb が 文字列 txta の先頭に含まれる場合 0 を返す。
　　含まれない場合。-1 を返す。

例）

```
 0 = instr( "ABCDEFG", "ABC" );

-1 = instr( "ABCDEFG", "CDE" );

-1 = instr( "ABCDEFG", "ACC" );
```

分類番号：05-16	ソースファイル： str_func.mar

機能：

符号なし整数（ 2 byte ）を その値を表す 10進文字列に変換する。

関数プロトタイプ宣言：

```
void    w_ascii( char *buf, int w );
```

説明：

引数１（出力）：　buf　10進文字列を格納するバッファ。

引数２（入力）：　w　 2 byte整数値（ 符号なし整数として扱う ）

分類番号：05-17	ソースファイル： str_func.mar

機能：

符号なし整数（ 2 byte ）を その値を表す 10進文字列に変換する。
その際、スペースによる文字数調整を行う機能を有する。

関数プロトタイプ宣言：

```
void    w_asciif( char *buf, int w, int cnt );
```

説明：

引数１（出力）： buf 10進文字列を格納するバッファ。

引数２（入力）： w 2 byte整数値（ 符号なし整数として扱う ）

引数３（入力）： 10進文字列変換時のフォーマット調整パラメータ
先頭にスペースを加えて、全体の文字数を 指定したい場合に使用。
値が指定された文字数より大きい場合は、値が優先する。

w = 2 で cnt = 3 の場合 bufの内容は " 2" となる。
w = 12345 で cnt = 3 の場合 bufの内容は "12345" となる。

分類番号：05-18	ソースファイル： str_func.mar

機能：

関数プロトタイプ宣言：

説明：

分類番号：08-01	ソースファイル： aki_mbio.mar or aki_usbio.mar

機能：

AKI_H8_マザーボード、及び AKI_H8_USB基板の DIP-SWポート、LED、LCDポートの
初期化を行う。

関数プロトタイプ宣言：

```
void    init_aki_mb( void );
```

説明：

分類番号：08-02	ソースファイル： aki_mbio.mar or aki_usbio.mar

機能：

AKI_H8マザーボード、AKI_H8_USB基板に接続される
16桁 2行のLCD（ 液晶表示器 ）の初期化を行う。

関数プロトタイプ宣言：

```
void    init_mb_lcd( void );
```

説明：

分類番号：08-03	ソースファイル： aki_mbio.mar or aki_usbio.mar

機能：

マザーボード上の DIP-SWの値を 取り出す。
（ init_aki()を呼び出した時の値を変数に保存しており、現在の DIP-SWの値ではない。）

関数プロトタイプ宣言：

```
int     get_mb_dipsw( void );
```

説明：

DIP-SWの bit数が、AKI_H8マザーは、 8bit（ Wordデータの b7 ～ b0 ）であるのに対し、
AKI_H8_USB基板は、4bit（ Wordデータの b3 ～ b0 ）である事に注意する事。
DIP-SWのビットが無い、上位ビットは 通常 0 になっている。

分類番号：08-04	ソースファイル： aki_mbio.mar or aki_usbio.mar

機能：

DIP-SWの 下位4bitにて、設定される、アドレス文字 "@" ～ "O" を取り出す。

関数プロトタイプ宣言：

```
int     get_mb_myadr( void );
```

説明：

関数値： DIP-SWの下位 4bitと 40h を組み合せて構成される ASCII文字を アドレス文字として
返す。 アドレス文字の組み合せは以下の通り

"@" : 40h	、	"H" : 48h
"A" : 41h	、	"I" : 49h
"B" : 42h	、	"J" : 4Ah
"C" : 43h	、	"K" : 4Bh
"D" : 44h	、	"L" : 4Ch
"E" : 45h	、	"M" : 4Dh
"F" : 46h	、	"N" : 4Eh
"G" : 47h	、	"O" : 4Fh

分類番号：08-05	ソースファイル： aki_mbio.mar or aki_usbio.mar

機能：

AKI_H8_MBまたは、AKI_H8_USB基板上の LED に データを出力します。
（ 該当するビットが 1 の時、LED が点灯する。）

関数プロトタイプ宣言：

```
void    put_mb_led( unsigned char dt );
```

説明：

引数１： dt LEDのビット並びに対応するデータ

AKI_H8_MB （ ２個 ）	AKI_H8_USB （ ４個 ）
	b3
	b2
b1	b1
b0	b0

なお、（ set_mb_led_ontime()と flash_mb_led() ）の組み合せと、
put_mb_ledは 同じ LED資源をアクセスするので 排他 扱いになります。
同じプログラム内にて混在して使用しないで下さい。

分類番号：08-06	ソースファイル： aki_mbio.mar or aki_usbio.mar

機能：

LEDの 点滅処理を行う際の 点灯開始、及び点灯時間（ 10[ms]単位 ）の設定。

関数プロトタイプ宣言：

```
void    set_mb_led_ontime( unsigned char dt );
```

説明：

LEDの接続されるポートの bit0のLEDを点滅処理に使用します。

引数： dt = 点灯時間（ 10[ms]単位 ）100と 設定したら１秒点灯することになる。

何らかの LEDを点灯させたいイベントが発生した場合に、set_mb_led_ontime()を
呼び出し、その後、無限ループ内にて flash_mb_led()を都度 読み出す事により
表示の更新を行う。

例えば、コマンドを受け付けたら １秒間 LEDを 点灯させるという場合などに使用する。

分類番号：08-07	ソースファイル： aki_mbio.mar or aki_usbio.mar

機能：

set_mb_led_ontime()で設定した時間 LED点灯、時間経過後 LED消灯を行うための
LED表示 更新処理。

関数プロトタイプ宣言：

```
void    flash_mb_led( void );
```

説明：

通常、無限ループ内にて、毎回呼び出す。
内部処理は、アセンブラで数ステップなので時間は殆どかからない。

なお、時間の経過はインターバルタイマの割り込み処理にてカウントしているので
init_timer()を呼び出してないと正常に動作しない。

分類番号：08-08	ソースファイル： aki_mbio.mar or aki_usbio.mar

機能：

AKIマザーボード上の LCDの 出力処理を実行させるか、停止させるかを設定する。

関数プロトタイプ宣言：

```
void    set_mb_lcdsw( unsigned char sw );
```

説明：

LCDの表示処理は、LCDとのタイミングをとるため書き込みに時間が、ややかかる。
LCDが接続されてない場合は、明らかに無駄な時間となるため、このLCD出力処理を
スキップする機能を実現する関数である。

引数１： sw = 0　LCD に 出力しない。
sw = 1　LCD に 出力する。

分類番号：08-09	ソースファイル： aki_mbio.mar or aki_usbio.mar

機能：

AKIマザーボード上の LCD の 表示内容を 消去する。

関数プロトタイプ宣言：

```
void    erase_mb_lcd( void );
```

説明：

スペースコードで ＬＣＤ表示を埋める。

分類番号：08-10	ソースファイル： aki_mbio.mar or aki_usbio.mar

機能：

AKIマザーボード上の LCD に文字列を 出力する。

関数プロトタイプ宣言：

```
void    print_mb_lcd( int ln, char* txt );
```

説明：

引数１： ln 表示行位置（ 0 か 1 ）上側から 0 行目、1 行目

引数２： txt 表示文字列（ 16文字を越えていたら 17文字目以降は表示されない。）

分類番号：08-	ソースファイル： aki_mbio.mar or aki_usbio.mar

機能：

関数プロトタイプ宣言：

説明：

分類番号：08-	ソースファイル： aki_mbio.mar or aki_usbio.mar

機能：

関数プロトタイプ宣言：

説明：