

TOSHIBA

SP-870-102

自己診断テスト サンプルソフトウェア

Rev 1.0

2008年 8月

- 当社は品質、信頼性の向上に努めておりますが、一般に半導体製品は誤作動したり故障することがあります。当社半導体製品をご使用いただく場合は、半導体製品の誤作動や故障により、生命・身体・財産が侵害されることのないように、購入者側の責任において、機器の安全設計を行うことをお願いします。
なお、設計に際しては、最新の製品仕様をご確認の上、製品保証範囲内でご使用いただくと共に、考慮されるべき注意事項や条件について「東芝半導体製品の取り扱い上のご注意とお願い」、「半導体信頼性ハンドブック」などでご確認ください。
- 本資料に掲載されている製品は、一般的電子機器（コンピュータ、パーソナル機器、事務機器、計測機器、産業用ロボット、家電機器など）に使用されることを意図しています。特別に高い品質・信頼性が要求され、その故障や誤作動が直接人命を脅かしたり人体に危害を及ぼす恐れのある機器（原子力制御機器、航空宇宙機器、輸送機器、交通信号機器、燃焼制御、医療機器、各種安全装置など）にこれらの製品を使用すること（以下“特定用途”という）は意図もされていませんし、また保証もされていません。本資料に掲載されている製品を当該特定用途に使用することは、お客様の責任でなされることとなります。
- 本製品の使用または、使用不能により生ずる付随的な損害（事業利益の損失、事業の中断、事業情報の損失、またはその他の金銭的損失を含むがこれらに限定されない）に関して当社は一切の責任を負いかねます。
- 本資料に掲載されている製品は、外国為替および外国貿易法により、輸出または海外への提供が規制されているものです。
- 本資料に掲載されている技術情報は、製品の代表的動作・応用を説明するためのもので、その使用に際して当社および第三者の知的財産権その他の権利に対する保証または実施権の許諾を行うものではありません。
- 本資料に掲載されている製品を、国内外の法令、規則および命令により製造、販売を禁止されている応用製品に使用することはできません。
- 本資料の掲載内容は、技術の進歩などにより予告なしに変更されることがあります。

目次

| | |
|---------------------------------------------|----|
| 1. 概要..... | 3 |
| 2. サンプルプログラムの概要..... | 3 |
| 2. 1 RAMメモリテスト (0x55, 0xAA の R/W テスト) | 3 |
| 2. 2 RAMメモリテスト (March-C テスト) | 3 |
| 2. 3 チェックサム計算..... | 3 |
| 2. 4 CRC 計算 (CRC-CCITT) | 3 |
| 3. プログラム内容..... | 4 |
| 3. 1 RAMメモリテスト (0x55, 0xAA の R/W テスト) | 4 |
| 3. 2 RAMメモリテスト (March-C テスト) | 8 |
| 3. 3 チェックサム計算..... | 16 |
| 3. 4 CRC 計算 (1) | 18 |
| 3. 5 CRC 計算 (2) | 21 |
| 4. 概略フローチャート..... | 25 |
| 5. 補足..... | 28 |
| 5. 1 数値表現..... | 28 |
| 5. 2 用語..... | 28 |

1. 概要

このサンプルプログラムは TLCS-870/C1 シリーズ用です。

このサンプルプログラムは各種の自己診断テスト用です。このサンプルプログラムの一部を取り出して再利用することで、希望するテストを実施することができます。

2. サンプルプログラムの概要

サンプルプログラムは、それぞれのモジュールに対して C 言語、及びアセンブリ言語で記述したものがああります。

・ C 言語 :

tml.c 及び tml.h の 2 つのソースファイルからなります。使用する関数、定義宣言を取り出して使用してください。

メモリテスト関連 (0x55/0xAA RW test、March-c test) に関しては、ご使用になるコンパイラによっては、意図したテストが行なわれない可能性がありますので、コンパイルされたアセンブラリストを確認するようにして下さい。

・ アセンブリ言語 :

tml.asm 及び tml.h の 2 つのソースファイルからなります。使用するモジュール、定義宣言を取り出して使用してください。

アセンブリ言語で記述されたモジュールは C 言語とのインターフェイスが考慮されています。C 言語のソースから、サンプルプログラムのモジュールを呼び出す場合は、関数タイプを” __adcel” とすることで利用することが出来ます。

2. 1 RAM メモリテスト (0x55,0xAA の R/W テスト)

指定された RAM エリア (スタックエリアを除く) に対しての 0x55、0xAA の Read/Write テストを行います。

2. 2 RAM メモリテスト (March-C テスト)

指定された RAM エリア (スタックエリアを除く) に対して、March-C でのテストを行います。March-C テストパターンは以下のように定義されます。

- ・ 指定されたエリアすべてに 0 を書き込む。
- ・ アドレス全体にわたって増分しながら、0 を読み取り 1 を書き込む。
- ・ アドレス全体にわたって増分しながら、1 を読み取り 0 を書き込む。
- ・ アドレス全体にわたって減分しながら、0 を読み取り 1 を書き込む。
- ・ アドレス全体にわたって減分しながら、1 を読み取り 0 を書き込む。
- ・ 指定されたエリアから 0 を読み取る。

2. 3 チェックサム計算

指定されたメモリエリアに対して、1byte 毎に加算したチェックサムを計算を行います。

チェックサムは 16bit で表されます。

RAM/ROM データの整合性確認や、データ送受信における誤り検出などに用いることが出来ます。但しデータの順序が入れ替わるなどした場合は、計算結果が同じになり、誤り検出ができませんので、より検出精度を求める場合は CRC 等他の方法を検討してください。

2. 4 CRC 計算 (CRC-CCITT)

指定された RAM/ROM エリアの巡回冗長検査 (CRC) を計算します。

サンプルプログラムの初期値は” 0xFFFF”、生成多項式は” $x^{16}+x^{12}+x^5+x^0$ ” です

3. プログラム内容

3. 1 RAMメモリテスト (0x55,0xAA の R/W テスト)

指定された RAM エリア (スタックエリアを除く) に対しての 0x55、0xAA の Read/Write テストを行います。

RAM エリアは開始アドレス、及びバイト数を指定します。バイト数は偶数でなければいけません。テストする RAM エリアの内容は保持されますが、割込みなど他の処理系で変更される可能性のあるものについては、この限りではありません。

= C 言語 =

| tml_ram_test_cb | | |
|-----------------|----------------------------------------------------------------------|---------------------------------------------------|
| function call | BOOL tml_ram_test_cb(volatile unsigned char *adr, unsigned int byte) | |
| parameter 1 | volatile unsigned char *adr | テスト開始アドレス |
| parameter 2 | unsigned int byte | テストバイト数 (1~65535) |
| return value | BOOL | TML_NO_ERROR : (1)no erro TML_ERROR : (0)error |

```

1  /*****
2  * NAME : tml_ram_test_cb
3  *
4  * PARAMETER : PARAMETER 1 address pointer on first data
5  *             PARAMETER 2 length of the byte
6  *
7  * RETURN VALUE : error infomation (TML_ERROR:error / TML_NO_ERROR:no error)
8  * DESCRIPTION : The R/W check of 0x55/0xAA in the specified range
9  *
10 *****/
11 #pragma optimize_off tml_ram_test_cb /* optimization restraint */
12 /* note ) specification bytes be an even number! */
13 BOOL tml_ram_test_cb(volatile unsigned char *adr,unsigned int byte)
14 {
15     BOOL err = TML_NO_ERROR; /* set no error */
16     unsigned char data_backup[2];
17
18     for ( ; byte>0; byte-=2) {
19         data_backup[0] = *adr; /* save of the date */
20         data_backup[1] = *(adr+1);
21
22         *adr = 0x55; /* The writing in check of 0x55, 0xaa */
23         *(adr+1) = 0xaa;
24         if ((*adr == 0x55) && (*(adr+1) == 0xaa)) {
25         }else{
26             err = TML_ERROR; /* set error */
27         }
28
29         *adr = 0xaa; /* The writing in check of 0xaa, 0x55 */
30         *(adr+1) = 0x55;
31         if ((*adr == 0xaa) && (*(adr+1) == 0x55)) {
32         }else{
33             err = TML_ERROR; /* set error */
34         }
35
36         *adr = data_backup[0]; /* restore of the data */
37         *(adr+1) = data_backup[1];
38
39         adr+=2;
40     }
41     return (err);

```

使用例

```
1 /*****
2 * Example : tml_ram_test_cb
3 *****/
4 /* 0x100~0x110 check! */
5 error = tml_ram_test_cb((unsigned char *)0x0100, 0x0010);
```

= アセンブリ言語 =

| .tml_ram_test_cb | | |
|------------------|-------------------------|---------------------------------------------------|
| parameter 1 | WA register | テスト開始アドレス |
| parameter 2 | BC register | テストバイト数(1~65535) |
| return value | A register | TML_NO_ERROR : (1)no erro TML_ERROR : (0)error |
| used register | WA, BC, IX, IY register | |
| Code Size | 67 byte | |
| Use Stack Size | 2 byte | |

```

1  ;*****
2  ;* NAME : .tml_ram_test_cb
3  ;*-----
4  ;* PARAMETER : WReg.  address pointer on first data
5  ;*             BReg.  length of the byte
6  ;*
7  ;* RETURN VALUE : Areg. error infomation (TML_ERROR:error / TML_NO_ERROR:no error)
8  ;* USE REGISTER : WA, BC, IX, IY
9  ;* DESCRIPTION : The R/W check of 0x55/0xAA in the specified range
10 ;*
11 ;*****
12
13     PUBLIC  .tml_ram_test_cb
14
15 .tml_ram_test_cb:
16     PUSH   DE                ; DE on Stack
17     LD     IX, WA            ; IX:Address
18     LD     A, TML_NO_ERROR  ; Error Inf. Clear
19     CMP   BC, 0             ; length check
20     J     T, ram_cb_end
21 ram_cb_s1:
22     LD     DE, (IX)         ; Backup Data
23     LD     (IX), 0x55       ; 0x55 Write
24     LD     (IX+1), 0xAA    ; 0xAA Write
25     CMP   (IX), 0x55       ; Read check
26     J     F, ram_cb_err1
27     CMP   (IX+1), 0xAA    ; Read check
28     J     T, ram_cb_s2
29 ram_cb_err1:
30     LD     A, TML_ERROR    ; Set Error
31 ram_cb_s2:
32     LD     (IX), 0xAA      ; 0xAA Write
33     LD     (IX+0x1), 0x55  ; 0x55 Write
34     CMP   (IX), 0xAA      ; Read Check
35     J     F, ram_cb_err2
36     CMP   (IX+0x1), 0x55  ; Read Check
37     J     T, ram_cb_s3
38 ram_cb_err2:
39     LD     A, TML_ERROR    ; Set Error
40 ram_cb_s3:
41     LD     (IX), DE        ; Reatore Data
42     ADD   IX, 2            ; Next Address
43     SUB   BC, 2           ; Counter decrement
44     CMP   BC, 0           ; finish ?
45     J     F, ram_cb_s1    ; else JP
46 ram_cb_end:
47     POP   DE              ; Restore DE
47     RET

```

使用例

```
1 ;*****  
2 ;* Example : .tml_ram_test_cb  
3 ;*****  
4 ; 0x100~0x110 check!!  
5     LD     WA, 0x0100  
6     LD     BC, 0x0010  
7     CALL  .tml_ram_test_cb
```

3. 2 RAMメモリテスト (March-C テスト)

指定されたRAMエリア(スタックエリアを除く)に対してのMarch-Cでのテストを行います。RAMエリアは開始アドレス、及びバイト数を指定します。

テストするRAMエリアの内容は保持されませんので、保持する必要がある場合は、テストRAMエリアのデータの退避、及びテスト終了後のデータ復帰処理を行う必要があります。

= C 言語 =

| tml_ram_test_marchc | | |
|---------------------|--------------------------------------------------------------------------|---------------------------------------------------|
| function call | BOOL tml_ram_test_marchc(volatile unsigned char *adr, unsigned int byte) | |
| parameter 1 | volatile unsigned char *adr | テスト開始アドレス |
| parameter 2 | unsigned int byte | テストバイト数(1~65535) |
| return value | BOOL | TML_NO_ERROR : (1)no erro TML_ERROR : (0)error |

```

1  /*****
2  * NAME : tml_ram_test_marchc
3  *
4  * PARAMETER : PARAMETER 1 address pointer on first data
5  *             PARAMETER 2 length of the byte
6  *
7  * RETURN VALUE : error infomation (TML_ERROR:error / TML_NO_ERROR:no error)
8  * DESCRIPTION : March-C in the specified range The test
9  *
10 *****/
11 #pragma optimize_off tml_ram_test_marchc /* optimization restraint */
12 BOOL tml_ram_test_marchc(volatile unsigned char *adr, unsigned int byte)
13 {
14     BOOL err = TML_NO_ERROR; /* set no error */
15     volatile unsigned char *stop_adr;
16     unsigned int i;
17
18     stop_adr = (volatile unsigned char*)(adr + byte -1); /* set end address */
19
20     /* The RAM clearance by the check range */
21     for (i=0; i<byte; i++){
22         *(adr+i) = 0x00;
23     }
24
25     /* The W/R check of "1" from the address head / the lower rank bit */
26     for (i=0; i<byte; i++){
27         if (TML_ERROR == bitchk_r0_w1(adr+i)) {
28             err = TML_ERROR; /* set error */
29         }
30     }
31
32     /* The W/R check of "0" from the address head / the lower rank bit */
33     for (i=0; i<byte; i++){
34         if (TML_ERROR == bitchk_r1_w0(adr+i)) {
35             err = TML_ERROR; /* set error */
36         }
37     }
38
39     /* The W/R check of "1" from the address lower rank / the lower rank bit */
40     for (i=0; i<byte; i++){
41         if (TML_ERROR == bitchk_r0_w1(stop_adr-i)) {
42             err = TML_ERROR; /* set error */
43         }
44     }

```



```

45
46 /* The W/R check of "0" from the address lower rank / the lower rank bit */
47 for (i=0;i<byte;i++){
48     if(TML_ERROR ==bitchk_r1_w0(stop_adr-i)){
49         err = TML_ERROR;          /* set error */
50     }
51 }
52
53 /* The check area is the confirmation of ALL"0" */
54 for (i=0;i<byte;i++){
55     if (*adr != 0x00) {
56         err = TML_ERROR;          /* set error */
57     }
58     adr++;
59 }
60
61 return(err);
62 }

```

-内部関数-

```

1 /*----- static function (uses at tml_ram_test_marchc()) -----*/
2 /*
3 *****
4 * NAME : bitchk_r0_w1
5 *-----
6 * PARAMETER : PARAMETER 1 address pointer
7 *
8 * RETURN VALUE : error (TML_ERROR:error / TML_NO_ERROR:no error)
9 * DESCRIPTION : W/R check of "1"
10 *
11 *****
12 */
13 #pragma optimize_off bitchk_r0_w1 /* optimization restraint */
14 static BOOL bitchk_r0_w1(volatile unsigned char *adr)
15 {
16     BOOL err = TML_NO_ERROR; /* set no error */
17
18     if (*adr != 0x00) { /* all "0" check */
19         err = TML_ERROR; /* set error */
20     }
21
22     *adr |= 0x01; /* set of the 0th bit */
23     if (*adr != 0x01) {
24         err = TML_ERROR; /* set error */
25     }
26
27     *adr |= 0x02; /* set of the 1st bit */
28     if (*adr != 0x03) {
29         err = TML_ERROR; /* set error */
30     }
31
32     *adr |= 0x04; /* set of the 2nd bit */
33     if (*adr != 0x07) {
34         err = TML_ERROR; /* set error */
35     }
36
37     *adr |= 0x08; /* set of the 3rd bit */
38     if (*adr != 0x0F) {
39         err = TML_ERROR; /* set error */
40     }
41
42     *adr |= 0x10; /* set of the 4th bit */
43     if (*adr != 0x1F) {
44         err = TML_ERROR; /* set error */
45     }
46
47     *adr |= 0x20; /* set of the 5th bit */
48     if (*adr != 0x3F) {
49         err = TML_ERROR; /* set error */
50     }
51
52     *adr |= 0x40; /* set of the 6th bit */
53     if (*adr != 0x7F) {
54         err = TML_ERROR; /* set error */
55     }
56 }

```

```

47     *adr |= 0x80;                               /* set of the 7th bit */
48     if (*adr != 0xFF) {
49         err = TML_ERROR;                       /* set error */
50     }
51
52     return(err);
53 }
54
55
56 /*
57 ****
58 * NAME : bitchk_r1_w0
59 *-----
60 * PARAMETER : PARAMETER 1 address pointer
61 *
62 * RETURN VALUE : error (TML_ERROR:error / TML_NO_ERROR:no error)
63 * DESCRIPTION : W/R check of "0"
64 *
65 ****
66 */
67 #pragma optimize_off bitchk_r1_w0             /* optimization restraint */
68 static BOOL bitchk_r1_w0(volatile unsigned char *adr)
69 {
70     BOOL err = TML_NO_ERROR;                   /* set no error */
71
72     if (*adr != 0xFF) {                        /* all "1" check */
73         err = TML_ERROR;                       /* set error */
74     }
75
76     *adr &= 0xFE;                               /* clear of the 0th bit */
77     if (*adr != 0xFE) {
78         err = TML_ERROR;                       /* set error */
79     }
80     *adr &= 0xFD;                               /* clear of the 1st bit */
81     if (*adr != 0xFC) {
82         err = TML_ERROR;                       /* set error */
83     }
84     *adr &= 0xFB;                               /* clear of the 2nd bit */
85     if (*adr != 0xF8) {
86         err = TML_ERROR;                       /* set error */
87     }
88     *adr &= 0xF7;                               /* clear of the 3rd bit */
89     if (*adr != 0xF0) {
90         err = TML_ERROR;                       /* set error */
91     }
92     *adr &= 0xEF;                               /* clear of the 4th bit */
93     if (*adr != 0xE0) {
94         err = TML_ERROR;                       /* set error */
95     }
96     *adr &= 0xDF;                               /* clear of the 5th bit */
97     if (*adr != 0xC0) {
98         err = TML_ERROR;                       /* set error */
99     }
100    *adr &= 0xBF;                               /* clear of the 6th bit */
101    if (*adr != 0x80) {
102        err = TML_ERROR;                       /* set error */
103    }
104    *adr &= 0x7F;                               /* clear of the 7th bit */
105    if (*adr != 0x00) {
106        err = TML_ERROR;                       /* set error */
107    }
108
109    return(err);
110 }

```

使用例

```
1 /*****  
2 * Example : tml_ram_test_marchc  
3 *****/  
4 /* 0x100~0x110 check!! */  
5 error = tml_ram_test_marchc ((unsigned char *)0x0100, 0x0010);
```

= アセンブリ言語 =

| . vde_ram_test_marchc | | |
|-----------------------|-------------------------|---------------------------------------------------|
| parameter 1 | WA register | テスト開始アドレス |
| parameter 2 | BC register | テストバイト数(1~65535) |
| return value | A register | TML_NO_ERROR : (1)no erro TML_ERROR : (0)error |
| used register | WA, BC, IX, IY register | |
| code size | 241 byte | |
| stack size | 7 byte | |

```

1  ;*****
2  ;* NAME : .tml_ram_test_marchc
3  ;*-----
4  ;* PARAMETER : WReg.  address pointer on first data
5  ;*             BReg.  length of the byte
6  ;*
7  ;* RETURN VALUE : Areg. error infomation (TML_ERROR:error / TML_NO_ERROR:no error)
8  ;* USE REGISTER : WA, BC, IX, IY
9  ;* DESCRIPTION : March-C in the specified range The test
10 ;*****
11
12     PUBLIC  .tml_ram_test_marchc
13
14 .tml_ram_test_marchc:
15     PUSH   HL           ; Stack on HL
16     PUSH   DE           ; Stack on DE
17
18     LD     HL, WA        ; WA:Start Address
19     LD     DE, HL        ; DE:Start Address
20     ADD   HL, BC         ;
21     DEC   HL             ; HL:End Address
22
23     CMP   BC, 0          ; length check
24     J     F, ram_mc_start
25     LD   B, TML_NO_ERROR
26     J     ram_mc_step_12
27 ;
28 ram_mc_start:
29     LD   B, TML_NO_ERROR ; error inf.
30 ;**** zero clear ****
31 ram_mc_step_0: ; All ZERO set
32     LD   (DE), 0         ; ZERO Store
33     CMP  HL, DE          ; Clear End
34     J    T, ram_mc_step_1 ; then JP
35     INC  DE              ; Address increment
36     J    ram_mc_step_0
37
38 ;**** 1-write from lower address ****
39 ram_mc_step_1: ;
40     LD   DE, WA          ; DE:Start Address
41
42 ram_mc_step_2: ;
43     CAL  .bitchk_r0_w1   ; 0-Read / 1-Write : error output is B
44     CMP  HL, WA          ; End
45     J    T, ram_mc_step_3 ; then jp
46     INC  WA              ; Address increment
47     J    ram_mc_step_2
48
49 ;**** 0-write from lower address ****
50 ram_mc_step_3: ;
51     LD   WA, DE          ; WA:Start Address

```

```

52
53 ram_mc_step_4:
54     CAL    .bitchk_r1_w0      ; 1-Read / 0-Write : error output is B
55     CMP    HL, WA             ; End
56     J      T, ram_mc_step_5   ;      then jp
57     INC    WA                 ; Address increment
58     J      ram_mc_step_4
59
60 ;***** 1-write from upper address *****
61 ram_mc_step_5:
62     PUSH   DE
63     LD     DE, HL              ; DE:End Address
64     POP    HL                  ; HL:Start Address
65     LD     WA, DE              ; WA:End Address
66 ram_mc_step_6:
67     CAL    .bitchk_r0_w1      ; 0-Read / 1-Write : error output is B
68
69     CMP    WA, HL              ; End
70     J      T, ram_mc_step_7   ;      then JP
71     DEC    WA                  ; Address decrement
72     J      ram_mc_step_6
73
74 ;***** 0-write from lower upper *****
75 ram_mc_step_7:
76     LD     WA, DE              ; WA:End Address
77
78 ram_mc_step_8:
79     CAL    .bitchk_r1_w0      ; 1-Read / 0-Write : error output is B
80     CMP    WA, HL              ; End
81     J      T, ram_mc_step_9   ;      then JP
82     DEC    WA                  ; Address decrement
83     J      ram_mc_step_8
84
85 ;***** All 0 check *****
86 ram_mc_step_9:
87
88     LD     WA, DE              ; WA:End Address
89 ram_mc_step_10:
90     CMP    (HL), 0             ; data == 0
91     J      T, ram_mc_step_11 ; then JP
92     LD     B, TML_ERROR        ; Error
93 ram_mc_step_11:
94     CMP    WA, HL              ; finish ?
95     J      T, ram_mc_step_12 ; then JP
96     INC    HL                  ; Address increment
97     J      ram_mc_step_10
98
99 ram_mc_step_12:
100    LD     A, B                 ; Areg:return
102
102    POP    DE                   ; Restore DE
103    POP    HL                   ; Restore HL
104    RET

```

-内部関数-

```

1  ;/*----- static function (uses at tml_ram_test_marchc()) -----*/
2  ;*****
3  ;* NAME : bitchk_r0_w1
4  ;*-----
5  ;* PARAMETER : WReg. address pointer
6  ;* RETURN VALUE : Breg. error infomation(TML_ERROR:error / TML_NO_ERROR:no error)
7  ;*****
8  .bitchk_r0_w1:
9     LD     IY, WA
10    CMP    (IY), 0y00000000    ;all "0" check
11    J      T, W1_1
12    LD     B, TML_ERROR        ;set error
13 W1_1:
14    SET    (IY), 0             ;set of the 0th bit

```

```

15      CMP      (IY), 0y00000001
16      J        T, W1_2
17      LD       B, TML_ERROR          ;set error
18  W1_2:
19      SET      (IY).1                ;set of the 1st bit
20      CMP      (IY), 0y00000011
21      J        T, W1_3
22      LD       B, TML_ERROR          ;set error
23  W1_3:
24      SET      (IY).2                ;set of the 2nd bit
25      CMP      (IY), 0y00000111
26      J        T, W1_4
27      LD       B, TML_ERROR          ;set error
28  W1_4:
29      SET      (IY).3                ;set of the 3rd bit
30      CMP      (IY), 0y00001111
31      J        T, W1_5
32      LD       B, TML_ERROR          ;set error
33  W1_5:
34      SET      (IY).4                ;set of the 4th bit
35      CMP      (IY), 0y00011111
36      J        T, W1_6
37      LD       B, TML_ERROR          ;set error
38  W1_6:
39      SET      (IY).5                ;set of the 5th bit
40      CMP      (IY), 0y00111111
41      J        T, W1_7
42      LD       B, TML_ERROR          ;set error
43  W1_7:
44      SET      (IY).6                ;set of the 6th bit
45      CMP      (IY), 0y01111111
46      J        T, W1_8
47      LD       B, TML_ERROR          ;set error
48  W1_8:
49      SET      (IY).7                ;set of the 7th bit
50      CMP      (IY), 0y11111111
51      J        T, W1_END
52      LD       B, TML_ERROR          ;set error
53  W1_END:
54      RET
55
56
57
58
59  ;*****
60  ;* NAME   : bitchk_r1_w0
61  ;*-----
62  ;* PARAMETER   : WReg. address pointer
63  ;* RETURN VALUE : Breg. error (TML_ERROR:error / TML_NO_ERROR:no error)
64  ;*****
65  .bitchk_r1_w0:
66      LD       IY, WA
67      CMP      (IY), 0y11111111      ;all "1" check
68      J        T, WO_1
69      LD       B, TML_ERROR          ;set error
70  WO_1:
71      CLR      (IY).0                ;clear of the 0th bit
72      CMP      (IY), 0y11111110
73      J        T, WO_2
74      LD       B, TML_ERROR          ;set error
75  WO_2:
76      CLR      (IY).1                ;clear of the 1st bit
77      CMP      (IY), 0y11111100
78      J        T, WO_3
79      LD       B, TML_ERROR          ;set error
80  WO_3:
81      CLR      (IY).2                ;clear of the 2nd bit
82      CMP      (IY), 0y11111000

```

| | | | |
|-----|---------|------------------|-----------------------|
| 83 | J | T, WO_4 | |
| 84 | LD | B, TML_ERROR | ;set error |
| 85 | WO_4: | | |
| 86 | CLR | (IY).3 | ;clear of the 3rd bit |
| 87 | CMP | (IY), 0y11110000 | |
| 88 | J | T, WO_5 | |
| 89 | LD | B, TML_ERROR | ;set error |
| 90 | WO_5: | | |
| 91 | CLR | (IY).4 | ;clear of the 4th bit |
| 92 | CMP | (IY), 0y11100000 | |
| 93 | J | T, WO_6 | |
| 94 | LD | B, TML_ERROR | ;set error |
| 95 | WO_6: | | |
| 96 | CLR | (IY).5 | ;clear of the 5th bit |
| 97 | CMP | (IY), 0y11000000 | |
| 98 | J | T, WO_7 | |
| 99 | LD | B, TML_ERROR | ;set error |
| 100 | WO_7: | | |
| 101 | CLR | (IY).6 | ;clear of the 6th bit |
| 102 | CMP | (IY), 0y10000000 | |
| 103 | J | T, WO_8 | |
| 104 | LD | B, TML_ERROR | ;set error |
| 105 | WO_8: | | |
| 106 | CLR | (IY).7 | ;clear of the 7th bit |
| 107 | CMP | (IY), 0y00000000 | |
| 108 | J | T, WO_END | |
| 109 | LD | B, TML_ERROR | ;set error |
| 110 | WO_END: | | |
| 111 | RET | | |

使用例

| | | | |
|---|-----------------------------------|----------------------|-------------------|
| 1 | ;***** | | |
| 2 | ;* Example : .tml_ram_test_marchc | | |
| 3 | ;***** | | |
| 4 | ; 0x100~0x110 check!! | | |
| 5 | LD | WA, 0x0100 | |
| | LD | BC, 0x0010 | |
| | CALL | .tml_ram_test_marchc | ; Areg=0 then OK: |

3. 3 チェックサム計算

指定されたメモリエリアのチェックサム計算（1バイト毎の加算）を行いません。
メモリエリアの指定は開始アドレス、及びバイト数を指定します。計算中はメモリエリアのデータが割込みなど他の処理系で変更されないようにして下さい。

= C 言語 =

| tml_chksum | | |
|---------------|--------------------------------------------------------------|------------------|
| function call | BOOL tml_chksum(const unsigned char *adr, unsigned int byte) | |
| parameter 1 | const unsigned char *adr | テスト開始アドレス |
| parameter 2 | unsigned int byte | テストバイト数(1~65535) |
| return value | unsigned int | 計算結果 |

```

1  /*****
2  * NAME : tml_chksum
3  *
4  * PARAMETER : PARAMETER 1 address pointer on first data
5  *              PARAMETER 2 length of the byte
6  *
7  * RETURN VALUE : checksum data
8  * DESCRIPTION : The computation of the checksum in the specified range
9  *
10 *****/
11 unsigned int tml_chksum(const unsigned char *adr, unsigned int size)
12 {
13     unsigned int cnt;
14     unsigned int data=0;
15
16     for(cnt=0; cnt<size; cnt++){                /* adds a specified range */
17         data = data + *adr;
18         adr++;
19     }
20     return (data);
21 }

```

使用例

```

1  /*****
2  * Example : tml_chksum
3  *****/
4  /* 0x100~0x110 calc. !! */
5  error = tml_chksum((unsigned char *)0x0100, 0x0010);

```


= アセンブリ言語 =

| . tml_chksum | | |
|---------------|---------------------|------------------|
| parameter 1 | WA register | テスト開始アドレス |
| parameter 2 | BC register | テストバイト数(1~65535) |
| return value | WA register | 計算結果 |
| used register | WA, IX, IY register | |
| code size | 29 byte | |
| stack size | 2 byte | |

```

1  ;*****
2  ;* NAME : .tml_chksum
3  ;*-----
4  ;* PARAMETER : WReg.  address pointer on first data
5  ;*             BReg.  length of the byte
6  ;*
7  ;* RETURN VALUE : WReg.  checksum data
8  ;* USE REGISTER : WA, IX, IY
9  ;* DESCRIPTION : The computation of the checksum in the specified range
10 ;*
11 ;*****
12
13     PUBLIC  .tml_chksum
14
15 .tml_chksum:
16     PUSH   DE                ; DE on Stack
17     LD     IX, WA            ; IX:Address
18     LD     WA, 0x0000        ; clear chksum data
19     LD     IY, 0x0000        ; clear counter
20     CMP    BC, 0x0           ; length check
21     J     LE, checksum_end
22 checksum_n:
23     LD     E, (IX)           ; read memory
24     LD     D, 0x0            ;
25     ADD    WA, DE            ; store chksum
26     INC    IX                ; address increment
27     INC    IY                ; counter increment
28     CMP    IY, BC            ; finish ?
29     J     LT, checksum_n     ; else JP
30 checksum_end:
31     POP    DE                ; Restore DE
32     RET

```

使用例

```

1  ;*****
2  ;* Example : .tml_chksum
3  ;*****
4  ; 0x100~0x110 calc. !!
5     LD     WA, 0x0100
6     LD     BC, 0x0010
7     CALL  .tml_chksum

```

3. 4 CRC計算 (1)

指定されたメモリエリアのCRC計算を行いません。

メモリエリアの指定は開始アドレス、及びバイト数を指定します。計算中はメモリエリアのデータが割込みなど他の処理系で変更されないようにして下さい。

サンプルプログラムの生成多項式は $x^{16}+x^{12}+x^5+x^0$ (CRC-CCITT)、初期値は“0xffff”となっています。

= C言語 =

| tml_crc | | |
|---------------|--------------------------------------------------------------------|---------------|
| function call | unsigned int tml_crc(const unsigned char *adr , unsigned int byte) | |
| parameter 1 | const unsigned char *adr | 開始アドレス |
| parameter 2 | unsigned int byte | バイト数(1~65535) |
| return value | unsigned int | 計算結果 |

```

1  *****/
2  * NAME : tml_crc
3  *-----
4  * PARAMETER : PARAMETER 1 address pointer on first data
5  *              PARAMETER 2 length of the byte
6  *
7  * RETURN VALUE : CRC data
8  * DESCRIPTION : The CRC-CCITT computation(general version)
9  *
10 *****/
11 unsigned int tml_crc(const unsigned char *adr , unsigned int byte)
12 {
13     unsigned int crc=0xffff;    /* first value */
14     unsigned int i;
15     unsigned char j;
16
17     for (i=0; i<byte; i+=1) {
18         crc ^= adr[i] << 8;
19         for (j=0; j<8; j+=1) {
20             if(crc & 0x8000) {
21                 crc = (crc << 1) ^ 0x1021;
22             }else{
23                 crc = crc << 1;
24             }
25         }
26     }
27     return crc;
28 }

```

*) 初期値、生成多項式を変更したい場合は、13行目、21行目の値を変更してください。

使用例

```

1  *****/
2  * Example : .tml_crc
3  *****/
4  /* 0x100~0x110 */
5  crc = tml_crc((unsigned char *)0x0100, 0x0010);

```

= アセンブリ言語 =

| . tml_crc | | |
|---------------|-------------------------|------------------|
| parameter 1 | WA register | テスト開始アドレス |
| parameter 2 | BC register | テストバイト数(1~65535) |
| return value | WA register | 計算結果 |
| used register | WA, BC, IX, IY register | |
| code size | 66 byte | |
| stack size | 4 byte | |

```

1  ;*****
2  ;* NAME : tml_crc
3  ;*-----
4  ;* PARAMETER : WReg.  address pointer on first data
5  ;*             BReg.  length of the byte
6  ;* RETURN VALUE : Areg.  CRC data
7  ;*
8  ;* USE REGISTER : WA, BC, IX, IY
9  ;* DESCRIPTION : The CRC-CCITT computation(general version)
10 ;*
11 ;*****
12
13     PUBLIC  . tml_crc
14
15 . tml_crc:
16     PUSH    HL                ; Stack on HL
17     PUSH    DE                ; Stack on DE
18     LD      IX, BC            ; IX: length
19     LD      DE, 0xffff        ; 1st data
20     LD      IY, 0x0           ;
21     CMP     IX, 0x0           ; check length
22     J       LE, tml_crc_end   ;
23 tml_crc_1:
24     LD      HL, IY            ;
25     ADD     HL, WA            ;
26     LD      C, (HL)          ; load memory data
27     LD      B, 0x0           ;
28     LD      HL, BC           ;
29     LD      B, 0x8           ;
30 tml_crc_2:
31     SHLCA   HL                ; 8bit shift
32     DEC     B                 ;
33     J       NE, tml_crc_2     ;
34     XOR     DE, HL            ;
35     LD      C, 0x0           ;
36 tml_crc_3:
37     TEST    D, 7              ;
38     J       T, tml_crc_4     ;
39     SHLCA   DE                ;
40     XOR     DE, 0x1021        ; Polynomial: 0x1021 (x^16 + x^12 + x^5 + 1)
41     J       tml_crc_5        ;
42 tml_crc_4:
43     SHLCA   DE                ;
44 tml_crc_5:
45     INC     C                 ;
46     CMP     C, 0x8           ;
47     J       LT, tml_crc_3    ;
48     INC     IY                ; decrement length
49     CMP     IY, IX           ; finish?
50     J       LT, tml_crc_1    ; else JP
51 tml_crc_end:
52     LD      WA, DE            ; WA: return data

```

| | | | |
|----|-----|----|--------------|
| 53 | POP | DE | ; Restore DE |
| 54 | POP | HL | ; Restore HL |
| 55 | RET | | |

*) 初期値、生成多項式を変更したい場合は、19 目、40 行目の値を変更してください。

使用例

| | | | |
|---|-----------------------|------------|--|
| 1 | ;***** | | |
| 2 | ;* Example : .tml_crc | | |
| 3 | ;***** | | |
| 4 | ; 0x100~0x110 | | |
| 5 | LD | WA, 0x0100 | |
| 6 | LD | BC, 0x0010 | |
| 7 | CALL | .tml_crc | |

3. 5 CRC計算 (2)

3. 4で紹介したサンプルプログラムを参照テーブルを用いることで、処理の高速化を図ったプログラムです。

= C 言語 =

| tml_crc_fast | | |
|---------------|------------------------------------------------------------------------|---------------|
| function call | unsigned int tml_crc_fast(const unsigned char *adr, unsigned int byte) | |
| parameter 1 | const unsigned char *adr | 開始アドレス |
| parameter 2 | unsigned int byte | バイト数(1~65535) |
| return value | unsigned int | 計算結果 |

```

1  /*****
2  /* CRC-CCITT TABLE
3                                     */
4  /*****
5  const unsigned int crc_table[256] = {
6      0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
7      0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
8      0X1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
9      0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
10     0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
11     0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
12     0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
13     0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
14     0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
15     0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
16     0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
17     0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
18     0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
19     0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
20     0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
21     0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,
22     0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,
23     0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,
24     0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,
25     0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,
26     0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
27     0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
28     0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
29     0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
30     0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
31     0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
32     0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
33     0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,
34     0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,
35     0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,
36     0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,
37     0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0
38  };
39
40  /*
41  *****/
42  * NAME : tml_crc_fast
43  *-----
44  * PARAMETER : PARAMETER 1 address pointer on first data
45  *             PARAMETER 2 length of the byte
46  *
47  * RETURN VALUE : CRC data
48  * DESCRIPTION : The CRC-CCITT computation (high-speed version)

```

```
49  *
50  ****
51  */
52  unsigned int tml_crc_fast(const unsigned char *s,unsigned int byte)
53  {
54      unsigned int crc=0xffff;          /* first value */
55      while(byte-- > 0) {
56          crc = crc_table[((crc>>8) ^ *s++) & 0xff] ^ (crc<<8);
57      }
58      return crc;
59  }
```

*) 初期値を変更したい場合は、54 行目の値を変更してください。

使用例

```
1  /*****
2  * Example : tml_crc_fast
3  *****/
4      /* 0x100~0x110 */
5      crc = tml_crc_fast((unsigned char *)0x0100, 0x0010);
```

= アセンブリ言語 =

| . tml_crc_fast | | |
|----------------|-------------------------|------------------|
| parameter 1 | WA register | テスト開始アドレス |
| parameter 2 | BC register | テストバイト数(1~65535) |
| return value | WA register | 計算結果 |
| used register | WA, BC, IX, IY register | |
| code size | 576 byte | |
| stack size | 4 byte | |

```

1  ;*****
2  ;* CRC-CCITT TABLE                               */
3  ;*****
4  crc_table:
5      DW      0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7
6      DW      0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF
7      DW      0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6
8      DW      0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE
9      DW      0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485
10     DW      0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D
11     DW      0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4
12     DW      0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC
13     DW      0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823
14     DW      0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B
15     DW      0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12
16     DW      0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A
17     DW      0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41
18     DW      0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49
19     DW      0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70
20     DW      0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78
21     DW      0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F
22     DW      0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067
23     DW      0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E
24     DW      0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256
25     DW      0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D
26     DW      0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405
27     DW      0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C
28     DW      0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634
29     DW      0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB
30     DW      0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3
31     DW      0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A
32     DW      0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92
33     DW      0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9
34     DW      0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1
35     DW      0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8
36     DW      0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0
37
38
39  ;*****
40  ;* NAME : tml_crc_fast
41  ;*-----
42  ;* PARAMETER : WArege. address pointer on first data
43  ;*             BCrege. length of the byte
44  ;* RETURN VALUE : Arege. CRC data
45  ;*
46  ;* USE REGISTER : WA, BC, IX, IY
47  ;* DESCRIPTION : The CRC-CCITT computation((high-speed version))
48  ;*
49  ;*****
50      PUBLIC .tml_crc_fast
51
52 .tml_crc_fast:

```

```

53      PUSH    HL                ; Stack on HL
54      PUSH    DE                ; Stack on DE
55      LD      IX, WA            ; IXreg. = Address
56      LD      WA, 0xffff        ; first data
57      CMP     BC, 0x0           ; if length=0
58      J      T, tml_crc_f4      ; then JP
59      DEC     BC                ; length decrement
60      LD      IY, BC            ;
61      tml_crc_f1:
62      LD      E, (IX)
63      LD      D, 0
64      INC     IX                ; Address increment.
65      LD      HL, WA            ; HLreg = CRC
66      LD      B, 8              ; shift 8 times.
67      tml_crc_f2:
68      SHRC    H                ; right shift
69      RORC    L                ;
70      DEC     B                ; shift end ?
71      J      ne, tml_crc_f2     ; else JP
72      XOR     HL, DE
73      AND     HL, 0x00FF
74      SHLCA   HL                ; Table data size adj.
75      ADD     HL, crc_table
76
77      LD      B, 0x8            ; shift 8 times.
78      tml_crc_f3:
79      SHLCA   WA                ; reft shift
80      DEC     B                ; shift end ?
81      J      ne, tml_crc_f3     ; else JP
82
83      XOR     WA, (HL)          ; WArege = CRC
84
85      CMP     IY, 0             ; finish ?
86      J      T, tml_crc_f4      ; then JP
87      DEC     IY                ; length decrement
88      J      tml_crc_f1
89      tml_crc_f4:
90      POP     DE                ; Restore DE
91      POP     HL                ; Restore HL
92      RET

```

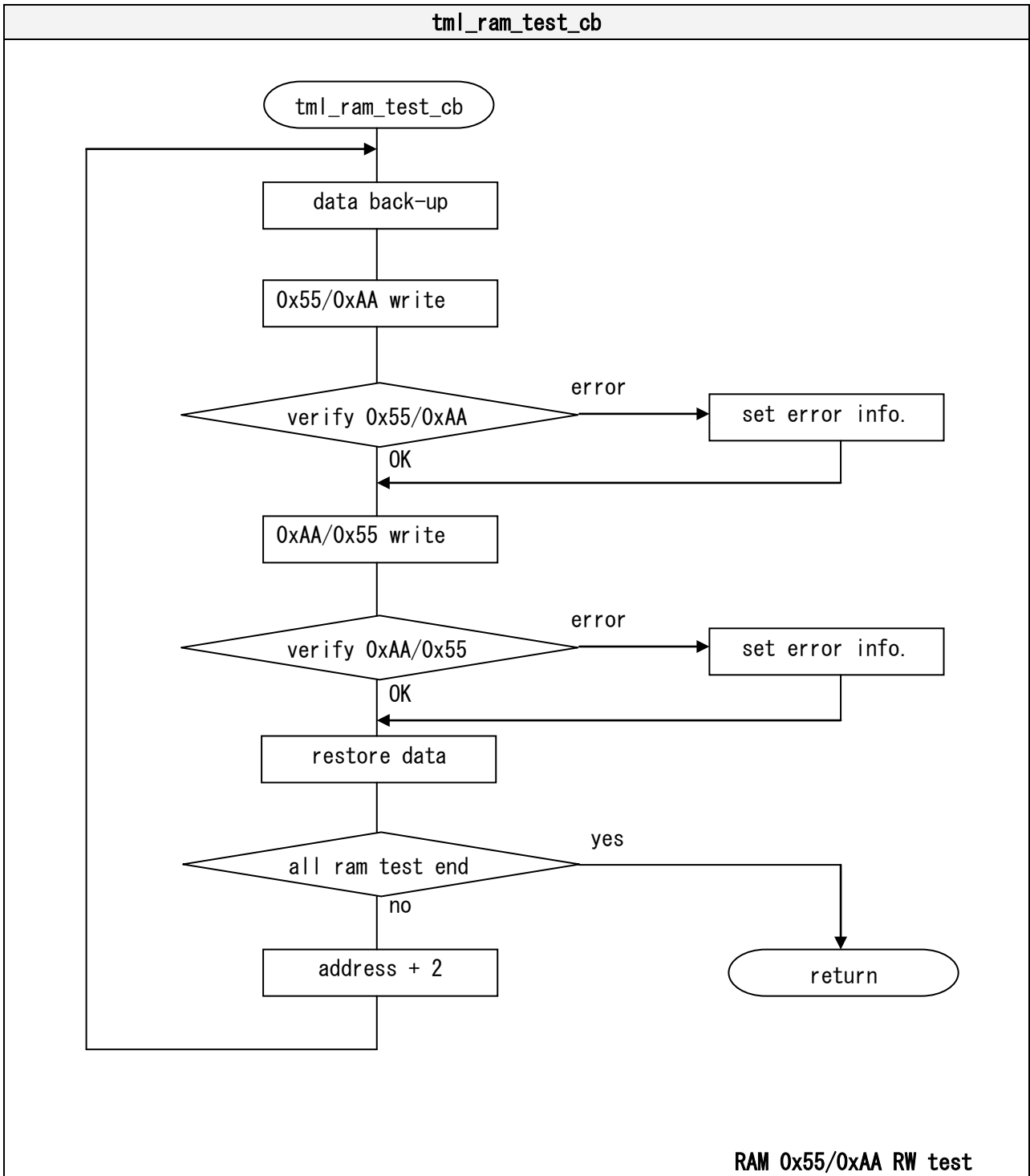
使用例

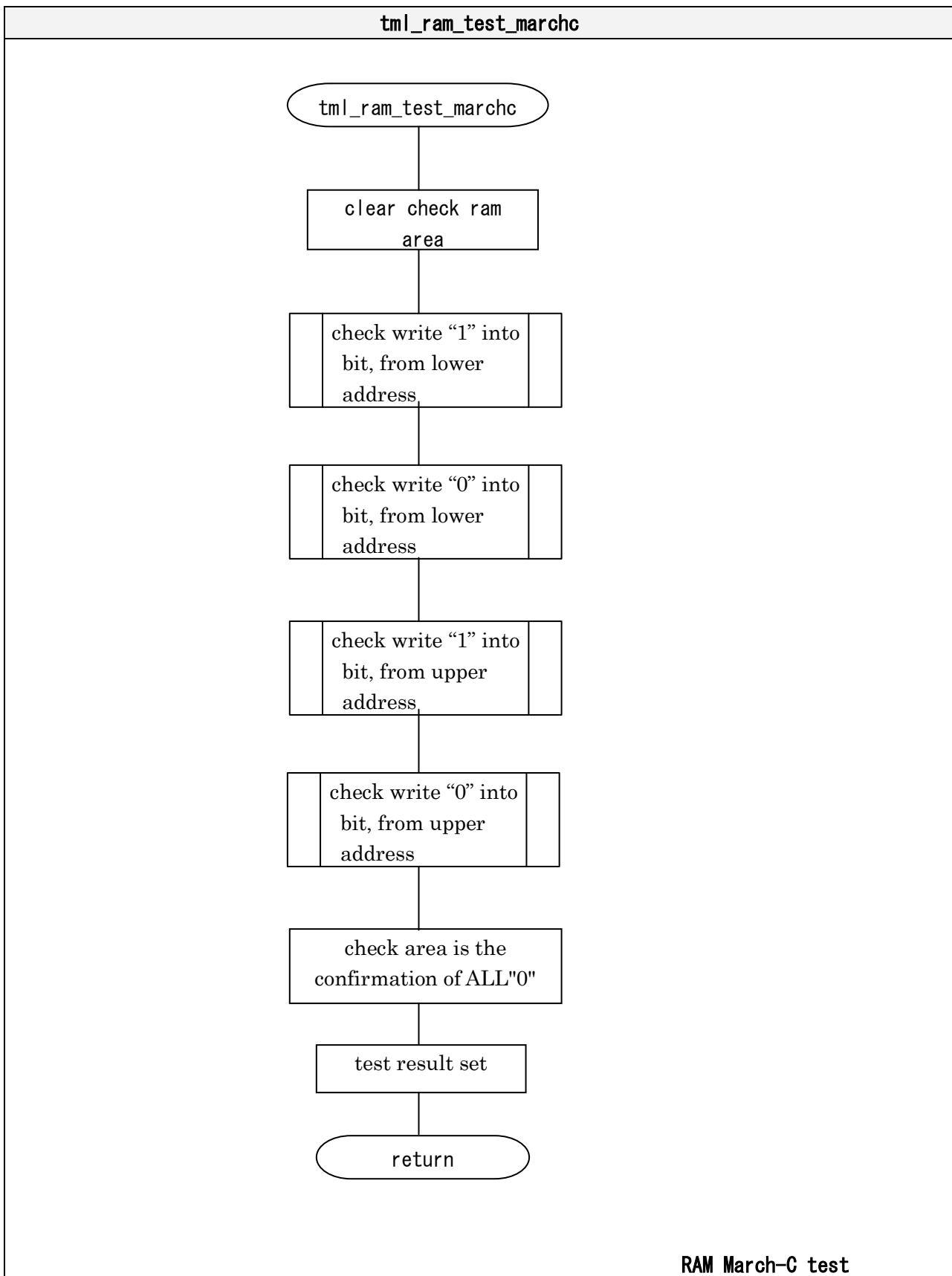
```

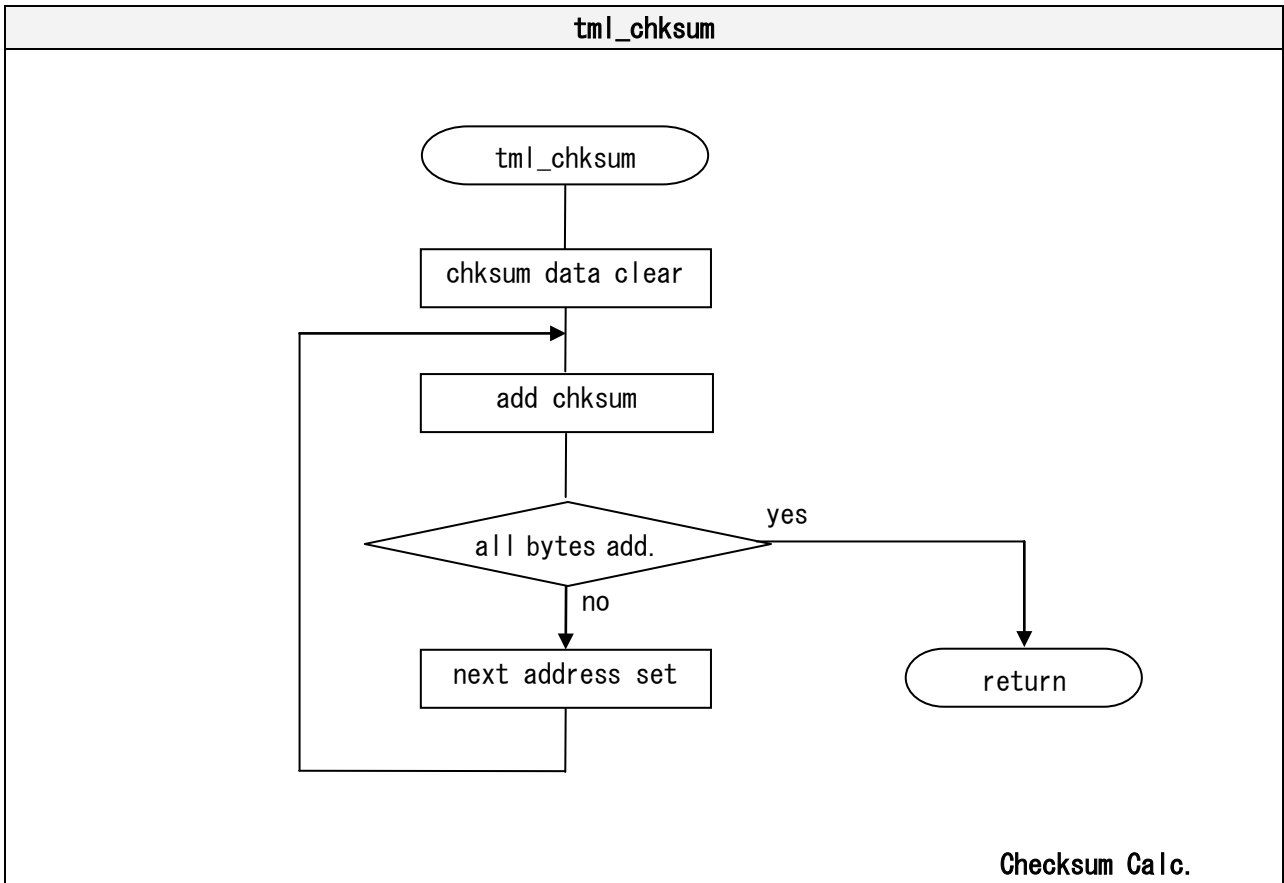
1      ;*****
2      ;* Example : .tml_crc_fast
3      ;*****
4      ; 0x100~0x110
5      LD      WA, 0x0100
6      LD      BC, 0x0010
7      CALL   .tml_crc_fast

```


4. 概略フローチャート







5. 補足

5. 1 数値表現

本文中の数値表現は以下のように、Unix表記で記してあります。

- ・ 2進数の場合、数値の前に0yまたは0Yで始まる0または1の数字列で表します。
- ・ 16進数の場合、数値の前に0xまたは0Xで始まる0～9, A～F, a～fの英数字列で表します。
- ・ 10進数の場合、0, 0x, 0X, 0y, 0Y以外で始まる0～9の数字列で表します。

5. 2 用語

CRC : 「巡回冗長検査」(Cyclic Redundancy Check)の略。連続して出現する誤り(バースト誤り)の検出が可能な誤り検出方式。CRC(8)、CRC(16)、CRC(32)などの方式がある。本文中のサンプルプログラムはCRC-CITTに沿ったものである。

チェックサム : データを送受信する際の誤り検出方法の一つ。送信前にデータを分割し、それぞれのブロック内のデータを数値とみなして合計を取ったもの。メモリのデータ整合性確認にも用いられる。