**ＤＹＮＡＸ新人研修(西田　哲)のレポート**

**（１）**

**ＣＰＵの種類**

**Ｖ５３（ＮＥＣ）／８０８６（Ｉｎｔｅｌ）：８０１８６、８０２８６、８０３８６、８０４８６、Ｐｅｎｔｉｎｕｍ、Ｐｅｎｔｉｎｕｍ－Ⅱ、Ｐｅｎｎｔｉｎｕｍ－Ⅲ**

**Ｃｅｒｅｌｏｎ－Ⅱ**

**ＣＰＵ　（ｃｅｎｔｒａｌ　ｐｒｏｃｅｓｓｏｒ　ｕｎｉｔの略）（中央情報処理装置）**

**Ｖ５３、ＳＨとの互換性**

**ＳＨ（日立）は、ドライバー、コントローラに適している。**

**（１）ＡＤコンバータを内臓している。**

**（２）いろいろなＩ／Ｏを内臓している。**

**モータ**

**ドライバー　（モータ）ドライバー（まわす、うごかすもの）**

**コントローラ　（制御）ホスト対応（パソコン）**

**ＲＴ１　（Ｒｏｂｏｔ・ｔｅｒｍｉｎａｌ）**

**Ｆｉｃｓ**

**Ａｔｏｍ**

**Ｆｉｃｓ－Ｔａｒｂｏ**

**（ＡＣ）－Ｔｕｒｂｏ**

【コントローラのＣＰＵ】

動作を記述する言語、コンピュータ・アゼンブラ

・Ｖ５３系 ←――― ＳＨＡＬ
↓ ．Ｓ
・ＳＨ系 ←――― ＳＨ
．ＳＨ

高級言語 Ｆｏｒｔｒａｎ、ｃｏｂｏｌ
Ｃ.
Ｃ＋＋
Ｂａｓｉｃ
（Ｊａｖａ、―――）
ＳＨＡＬ

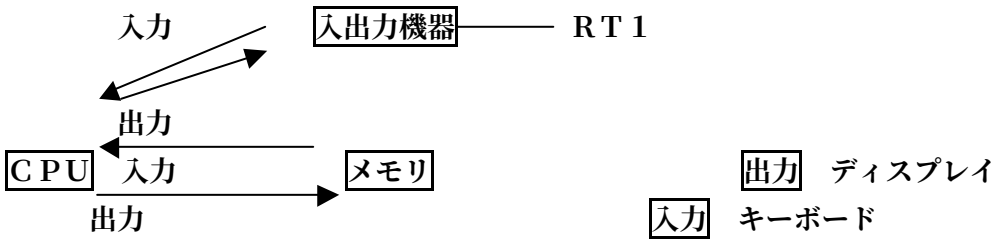
アゼンブラ言語 Ｍｏｖ ＡＸ、１００
１対１対応

マシン（機械）語 ２ＢＨ ３５Ｈ

【構造化】

（1）ｆｏｒ（；；）、Ｉｆ、ｓｗｉｔｃｈ、
（2）ｄｅｃｌａｒｅ 宣言
（3）ｍｏｄｕｌｅ モジュール

【コンピュータ、ＣＰＵは、何をするのか？】




演算
+、-、＊、/
論理 AND、OR、XOR
各判断

【入出力機器】

・デジタルチェッカー

ＤＩ＝ｄｅｇｉｔａｌ　ｉｎｐｕｔ

スイッチ

ＤＯ＝ｄｅｇｉｔａｌ　ｏｕｔｐｕｔ

ランプ　Ｌ．Ｅ．Ｄ

パルス出力

【動作命令】

① 入力

↓

② 演算

↓

③ 出力

↓

④ 制御の流れを支える

ＳＨ７０００で記述する。

ＳＨＡＬ７０００

Ｐｒｏｃ ←―― ｐｒｏｃｅｄｕｒｅの略

手続き　動作記述

【ＣＰＵ　レジスタ】

ＡＸ，ＢＸ、ＣＸ、ＤＸ、ＳＩ、ＤＩ

入力 ←

ＣＰＵ　　内部メモリ

ＡＸ，ＢＸ…　→ 出力

Ｉ／Ｏ

ﾚｼﾞｽﾀ　　ﾒﾓﾘ　　ﾜｰｸ（WORK）変数　　パラメータ

ＳＩ＝ｐａｄｒｓｖ；　　プログラム

入力 ◀──代入文

（言語によっては、代入文は：＝）

【　例　】

①　ｍｄｉｔｐｂ　＝　１４　　入力して出力

ﾒﾓﾘ／変数.　　定数.　◀──▶ 変数

②　@（ｗｏｒｄ）ＳＩ[２]＝ＡＸ

出入力　　メモリ　　ﾚｼﾞｽﾀ

-、*、/

③ＳＩ＋＝１６　　１６を足しなさい。演算命令

（ＳＩ＝ＳＩ+１６）

**【制御の流れ】**

①　ｆｏｒ（；；）　　ｆｏｒ文

（逐次）処理

ｋ＝０；

ｆｏｒ（ｉ＝０　；　ｉ<１０　；　ｉ＋＋）{

終了チェック　ステップ毎処理

ｋ＋＝ｉ；

| ｋ | ｉ | ２８ | ８ |
|---|---|---|---|
| ０ | ０ | ３６ | ９ |
| ０ | １ | ４７ | １０ |
| １ | ２ | | |
| ３ | ３ | | |
| ６ | ４ | | |
| １０ | ５ | | |
| １５ | ６ | | |
| ２１ | ７ | | |

② ｉｆ文

ｉｆ (条件式) ｛ 処理

}

結果として ０か０以外

偽 真

FALSE TRUE

(%TRUE)

③ ｓｗｉｔｃｈ（ＡＸ）{

ｃａｓｅ□:

ｂｒｅａｋ;

ｃａｓｅ□:

ｂｒｅａｋ;

ｄｅｆａｕｌｔ;

}

④

ａｌｌ_ｐｔｌ_ｍｔｘ_ｃｌｒ() ◀⟶A(F)call all_plt_mtx_clr

（関数呼び出し）

サブルーチン ｃａｌｌ

ｃｏｐｙ_ＰＧＭ ()

【ｂｉｔ、ｂｙｔｅ、ｗｏｒｄ、ｌｏｎｇ】

ｂｉｔ：情報量の最小単位　0か1か

（例）

（1）スイッチ　1個　ON／OFF　　DI

（2）L．E．D　1個　つくか、消えているか

【2進数、10進数】

2進数　1桁

10進数　0、1、2、3、…8、9、10、11…99、100

| 2進法 | 10進法 |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 10 | 2 |
| 11 | 3 |
| 100 | 4 |
| 101 | 5 |
| 110 | 6 |
| 111 | 7 |
| 1000 | 8 |
| 1001 | 9 |
| 1010 | 10 |
| 1011 | 11 |
| 1100 | 12 |
| 1101 | 13 |
| 1110 | 14 |
| 1111 | 15 |
| 10000 | 16 |

1ビット

```
   1
 +1
----
  10

   0     0     1     1
 +0    +1    +0    +1
----  ----  ----  ----
   0     1     1    10
```

ｂｉｔｅ＝８ｂｉｔ
０００．０００　　０
０００．０００１　　１
０００．００１０　　２
１０００．０００　１２８
１１１１．１１１１　２５５
**2進数1桁**　　0か1の2通り
**2進数2桁**　　００
　　０１　　2＊2＝4通り
　　１０　　0から3まで
　　１１
**3進数3桁**　　０００
　　００１　　2＊2＊2＝8通り
　　０１０　　0から7まで
　　１０１
　　１１０
　　１１１
**2進数4桁**　　００００
　　・　　2＊2＊2＊2＝１６通り
　　・　　2から１５まで
　　・
　　・
　　１１１１

---

１ｂｙｔｅ　　2進数8桁＝８ｂｉｔ　　１０進数でいうと0から２５５の２５６通り
　　－１２８から１２７の２５６通り
　　2進数の符号の問題

ｗｏｒｄ＝２ｂｙｔｅ＝１６ｂｉｔ
**10進数**　0から６５５３５　２５６＊２５６＝６５５３６
（－３２７６８から３２７６７の６５５３６通り）
ｌｏｎｇ（ｗｏｒｄ）＝４ｂｙｔｅ＝３２ｂｉｔ
－２１４７４８３６４８から２１４７４８３６４７
６５５３６＊６５５３６＝４２９４９６７２９６　/　２＝２１４７４８３６４８

【１６進数】

１６進数（ＨＥＸＡＤＥＣＩＭＡＬ　ＮＵＭＢＥＲ）　　ＨＥＸ（ヘキサ）

２進数　０か１

１０進数　０から９

１６　　１６進数　０、１…、８、９、Ａ，Ｂ、Ｃ、Ｄ、Ｅ、Ｆ

10　11　12　13　14　15

| 10 進数 | 16 進数 | 2 進数 |
|---|---|---|
| ０ | ０ | ０００００ |
| １ | １ | ０００１ |
| ２ | ２ | １０ |
| ３ | ３ | １１ |
| ４ | ４ | １００ |
| ５ | ５ | １０１ |
| ６ | ６ | １１０ |
| ７ | ７ | １１１ |
| ８ | ８ | １０００ |
| ９ | ９ | １００１ |
| １０ | Ａ | １０１０ |
| １１ | Ｂ | １０１１ |
| １２ | Ｃ | １１００ |
| １３ | Ｄ | １１０１ |
| １４ | Ｅ | １１１０ |
| １５ | Ｆ | １１１１ |

| | | | |
|---|---|---|---|
| １６ | １０ | １．０００ | Ｂ |
| １７ | １１ | １．０００１ | Ｂ |
| １８ | １２ | １．００１０ | |
| １９ | １３ | １．００１１ | |

１２Ｈは、Ｖ５３　ＳＨＡＬ　　表記法

０ｘ１２は、　ＳＨ

０ｘＡＢＣＤ ⟶ ４３９８１

| | |
|---|---|
| Ｄ | １３ |
| Ｃ＊１６ | １２＊１６ |
| Ｂ＊２５６ | １１＊２５６ |
| Ａ＊４０９６ | １０＊４０９６ |
| | ４３９８１ |

| | １６進数 | １０進数 |
|---|---|---|
| | ０ | ０ |
| | ・ | ・ |
| | ・ | ・ |
| | Ｆ | １５ |
| | １０ | １６ |
| byte　８ビット | ＦＦ | ２５５ |
| | １００ | ２５６ |
| | １２ビットＦＦＦ | ４０９５ |
| | １０００ | ４０９６ |
| word　１６ビットＦＦＦＦ | | ６５５３５ |
| | １００００ | ６５５３６ |

正とみると

ＦＦＦＦ．ＦＦＦＦ　　４２９４９６７２９５（－１）

１６進数１桁＝４ｂｉｔ　２＊２＊２＊２＝１６通り

（０から１５、０からＦ、０から９、ＡからＦ）

１バイト　１６進数　２桁　８ビット　０ｘ００から０ｘＦＦ（０から２５５）２５６通り

ｗｏｒｄ＝２ｂｙｔｅ１６進数　４桁　１６ビット　０ｘ０００００から０ｘＦＦＦＦ（０から６５５３５）６５５３６通り

ｌｏｎｇ（ｗｏｒｄ）＝４ｂｙｔｅ　８桁　３２ビット０ｘ０００００．０００００から０ｘＦＦＦＦ（０から４２９４９６７２９５）約４０億通り

| | ２進数（bit） | １６進数（Hex） | バイト | １０進数 |
|---|---|---|---|---|
| ｂｉｔ | １桁 | | | 0 か 1 |
| ｂｙｔｅ | ８桁 | ２桁 | １バイト | 0 から 255 |
| ｗｏｒｄ | １６桁 | ４桁（0 から 0xFFFF） | 2 バイト | 0 から 65535 |
| ｌｏｎｇ | ３２桁 | ８桁 | 4 バイト | 0 か 4294967295 |

**【2 進数、16 進数の符号】**

**整数**　　**1の倍数**

**実数**　　**－１２３．５６（１．２３E５６）**　　**Ｈｏａｔ、ｄｏｕｂｌｅ**
**活動小数点（実数）**

**（複素数）**　　**固定小数点（実数）**

**Ｆｉｃｓにおいて**
**X＝１２．３４mm　　内部的には、１２３４として扱っている.**
**見かけだけ**
**０．０１mm を１とする数**
DECIMAL＝**１０進数**
HEXADECIMAL　NUMBER＝16 **進数**

| 10 **進数** | | 16 **進数** |
|---|---|---|
| **－１** | ◀――――▶ | FFFF．FFFF |

**「１６bit（word）を符号付き整数とみなす」場合**
**関数電卓は、16 進数を３２bit（long）の符号付き整数としてあつかっている。**
MSB：Most　significant　bit　**最上位ビット**　**符号をあらわす　０：正　１：負**
**関数電卓は、16 進数を３２bit(long)の符号付き整数として扱っている。**

| | |
|---|---|
| **５** | **００００．０１０１** |
| ＋　**９** | **００００．１００１** |
| **１４** | **００００　１１１０** |

| | | |
|---|---|---|
| **－１** | （１１１１） | **１１１１** |
| **０** | | **００００** |
| **２** | | **０００１** |
| **３** | | **００１０** |
| **４** | | **０１００** |

| | |
|---|---|
| | **１１１１** |
| ＋ | **００００** |
| **１** | **０００００** |

（curry、over　flow）

**【１６進数に関して】**

**（１）ＭＳＢが符号を表わす。　　０：　正**

**１：　負**

**（２）正・負の加算（足し算）は、正・負の区別をせずに、正正、負負、負正、負負の４つの符号を全くおなじように扱うことができる。**

**ｗｏｒｄ（１６bit＝２byte）を符号つき整数とみなすと**

**０ｘ８０００から０ｘ７FFFF の数**

**－３２７６８から＋３２７６７**

**符号なし整数とみなすと**

**０から０ｘFFFF の数**

**０から６５５３５**

**符号つきのとき　０ｘFFFF＝－１**

**ｌｏｎｇ　３２ｂｉｔを符号つき整数とみなすと**

**０ｘ８０００．０００００から０ｘ７ＦＦＦＦ．ＦＦＦＦの数**

**－２１４７４８．３６４８から２１４７４８．３６４７**

**０ｘＦＦＦＦ．ＦＦＦＦ　＝－１**

**－２　　０ｘＦＦＦＦ．ＦＦＦＥ**

**－１６　　０ｘＦＦＦＦ．ＦＦＦ０**

**３２ｂｉｔの場合**

**ＭＳＢが、符号を表わす。０：正　　１：負**

| | | | |
|---|---|---|---|
| | 8 | ０００．００００ | |
| | 8 | ｘｘｘ．ｘｘｘｘ | |
| | | ・ | |
| | | ・ | |
| | | ・ | |
| | 9 | | |
| | A | | |
| | ・ | B | －５ |
| | ・ | C | －４ |
| | E | D | －３ |
| | F | ｘｘｘ．ｘｘｘE | －２ |
| | F | ＦＦＦ．ＦＦＦＦ | －１ |
| | 0 | ０００．０００ | 0 |
| | 0 | ０００．０００１ | 1 |
| | | ・ | 2 |
| | | ・ | |
| | 7 | ＦＦＦ．ＦＦＦＦ | |

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | ０００ | 8 | 1 | ０００ |
| 1 | 0 | ００１ | 9 | 1 | ００１ |
| 2 | 0 | ０１０ | A | 1 | ０１０ |
| 3 | 0 | ０１１ | B | 1 | ０１１ |
| 4 | 0 | １００ | C | 1 | １００ |
| 5 | 0 | １０１ | D | 1 | １０１ |
| 6 | 0 | １０１ | E | 1 | １１０ |
| 7 | 0 | １１０ | F | 1 | １１１ |

**１６進数は、ＭＳＢを符号とすることにより**

**足し算／引き算**

**正負が、統一的に（同じように）扱うことができる。**

**１０進数の場合**

**－１＋１ ⟶ ０**

**３２ｂｉｔの場合**

```
      FFFF. FFFF   －1
  +   0000. 0001     1
-------------------------
    1 0000. 0000     0
```

**【論理演算】**

**・論理演算**

**（メモリ／アドレス）**

| | | |
|---|---|---|
| **かつ** | **ＡＮＤ** | **論理積** |
| **または** | **ＯＲ** | **論理和** |
| | **ＸＯＲ** | **排他的論理和** |
| | **e**xclusive or | |

**【ＡＮＤ】**

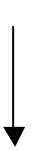
AND **かつ** **論理積**

**２つの入力⟶ １つの出力**

**０か１　　０か１**

| A | B | C＝A＆（＆＆）B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | |

**１ｂｉｔでいうと**　　(8bit、16bit、32bit)



**１＝ＴＲＵＥ　　真　（０でない）　　条件を満たす**

**０＝ＦＡＬＳＥ　　偽　（０である）　　条件を満たさない**

**ＡもＢも成り立っているときのみ（Ａ＆Ｂ）も成り立つ。（両方とも成り立ったとき）**

【ＯＲ】

ＯＲ　　「または」.「論理和」

| A | B | C＝A｜B |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |

どちらか１つでも１の時　1

AかB　　A｜B

両方のときだけ　　0

【ＸＯＲ】

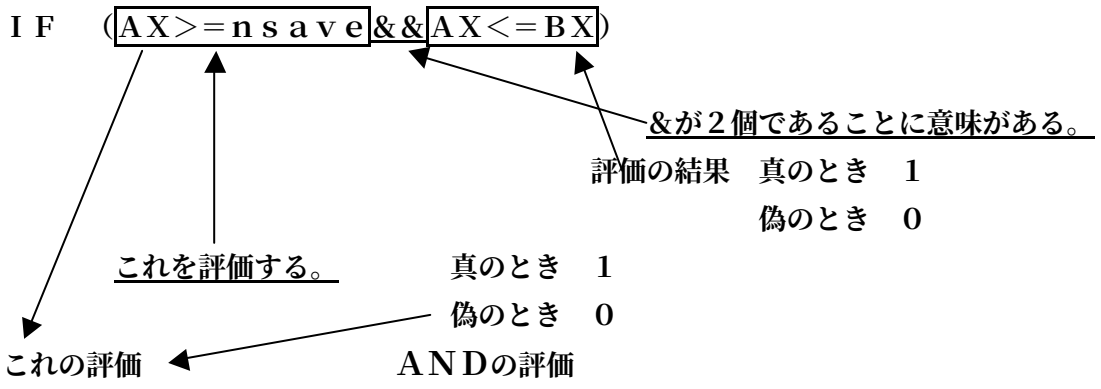
exclusive　or

| A | B | A＾B |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

AがBどちらかだけか１のとき１となる。

両方０、両方１のとき１は０となる。

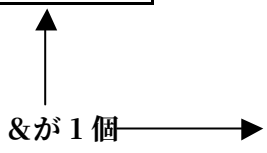
【ＩＦ（　　）条件式】



ｉｆ（ ＡＸ ＆＆ ＢＸ ）

ＡＸ ＆＝ＢＸ　　ＡＸ ＋／－ ＝ ＢＸ

ＡＸ　　ＡＸ ＆ ＢＸ



＆が１個 → ｂｉｔごとの論理積

ｉｆ（ ＡＸ ＆ ＢＸ ）

３２ｂｉｔ

ｂｉｔごとにＡＮＤ ｉｆ（ ）の（）の中身の結果が、０か、＜＞０（０でない）

【ＡＸ ＆ ＢＸ】

ＡＸ ＆ ＢＸについて

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ＡＸ | 0000. | 0100. | 0000. | 0000. | 0000. | 0000. | 1111. | 0000 |
| ＆ | | | | | | | | |
| ＢＸ | 1101. | 1100. | 0000. | 1111. | 1010. | 0000. | 1010. | 1011 |
| | 0000. | 0100. | 0000. | 0000. | 0000. | 0000. | 1010. | 0000 |

ＡＸに、格納（セーブ）しない。

マスクする

ＡＸ ＝ ０ｘ１２３４５６７８

下位１６ｂｉｔの値だけが欲しい場合

０ｘ０００００ＦＦＦＦでＡＮＤをとる。

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| | 0001. | 0010. | 0011. | 0100. | 0101. | 0110. | 0111. | 1000 |
| ＡＮＤ） | 0000. | 0000. | 0000. | 0000. | 1111. | 1111. | 1111. | 1111 |
| | 0～0 | 0～0 | 0～0 | 0～0 | 0101 | 0110 | 0111 | 1000 |

| ０ｘ | ０ | ０ | ０ | ０ | ５ | ６ | ７ | ８ |
|---|---|---|---|---|---|---|---|---|

ｉｆ（ ＡＸ ＆ ０ｘ００００ＦＦＦＦ）

ｂｉｔごとの結果が、０（すべてのｂｉｔが０）のとき ｆａｌｓｅ 非実行

どれか１ｂｉｔでも１のとき ｔｒｕｅ ＩＦ文実行

［OR］　AX ｜ ＝ BX ｜ BX

例）　AX＝0x1234　0000
OR　BX＝0x0000　5678
0x1234　5678

AX　0001. 0010. 0011. 0100. 0000. 0000. 0000. 0000
BX　0000. 0000. 0000. 0000. 0000. 0101. 0110. 1000
0000 0000 0000 0000 0101 0110 0111 1000
0x 1 2 3 4 5 6 7 8
これだったら　AX＋＝BX、　AX＝AX＋BXと同じ

［XOR］　論理の反転に使う。
AX＝0x1234　5678
BX＝0xFFFF　FFFF
A^＝B

0001. 0010. 0011. 0100. 0101. 0110. 0111. 1000
XOR　1111. 1111. 1111. 1111. 1111. 1111. 1111. 1111
1110 1101 1100 1011 1010 1001 1000 0111

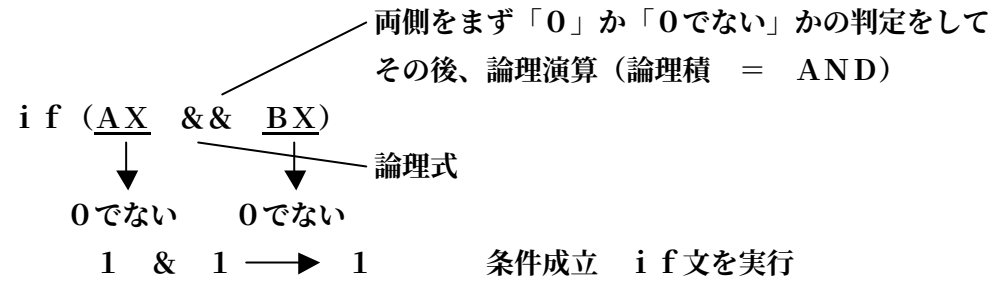
AXの各ｂｉｔを反転（0⟶1）、（1⟶0）したものができあがる。

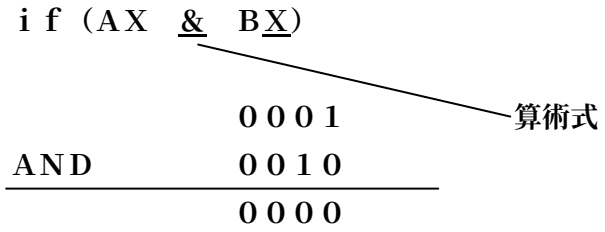
よく使う例
ＤＩの反転
反転したい　ビットを１
反転したくないビットを０
８ビットで考える。

ＤＩ入力　xxxx xxxx
反転ビット　0100 1100
xxxx xxxx
反転します

ＡＸ＝０ｘ０００１

ＢＸ＝０ｘ００１０



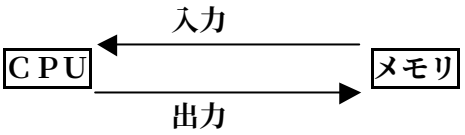




０　偽（ｆａｌｓｅ）　条件が成立しない　ｉｆ文を実行しない。

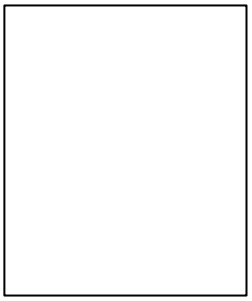
ビットごとに論理積（ＡＮＤ）をとって、その結果が０か０でないかとみる

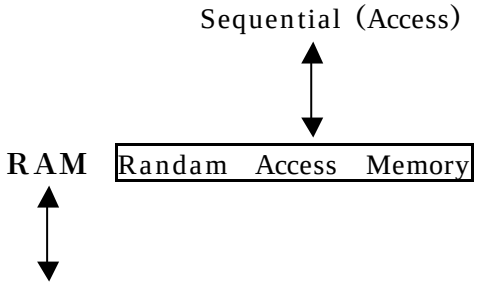
**【メモリ／アドレス】**

**メモリ／アドレス（　＝　番地　）**




**内部メモリ（ＩＣ）**






**本来は対立する用語ではなく** ROM **も言葉の上の意味では、**

Randam Access Memory

ROM　　RWM／ Read. write. Memory

**現実には** RAM **を** RWM **の意味で使っている。**

ROM•…　PROM、FLASH **メモリ**　　**（ロジック、コード、プログラム）**

RAM•…　RAMchip　　**（パラメータ、**Fics **のプログラム、ワーク、変数）**

【ROM】

PROM　　**Ｖ５３で使われている。**

**書くときは、ＲＯＭ　ｗｒｉｔｅｒで書く。**

**消すときは、Ｅｒｓｅｒで消す。**

**ＣＰＵから使うときは、読みだけ**　Read only Memory

**フラッシュメモリ**　　SH **で、**PROM **の代わりに使っている**

**（携帯電話にも使われている。）**

**書くときに、（ＣＰＵを経由して）パソコンから直接書きこめる。**

**ダウンロードする。　ｔｏｏｌ（ｄｘｌｏａｄｅｒ）**

**ＣＰＵにとって特殊な場合 ⟶ ダウンロード中は他のことはできない**

**通常の使用は、読むだけ　Ｒｅａｄ　ｏｎｌｙ　Ｍｅｍｏｒｙ**

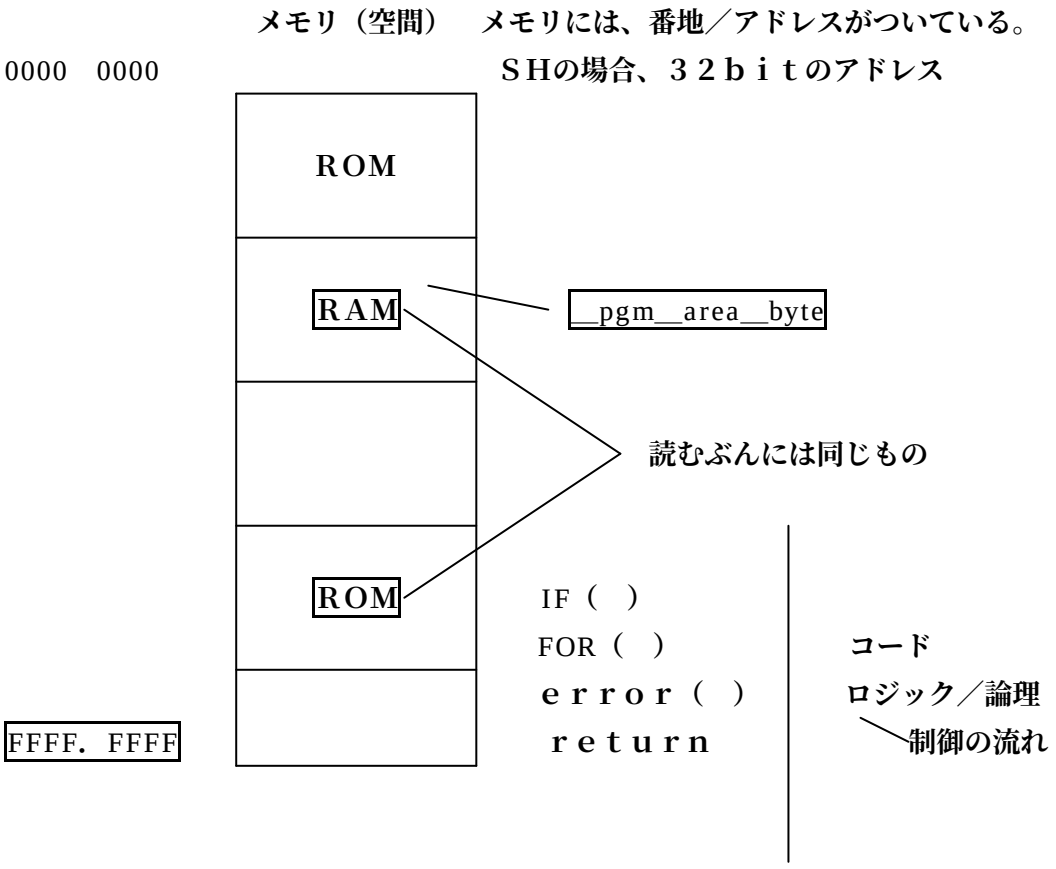
**（コード、ロジック、プログラム）　ｉｆ（　）、ｆｏｒ（；；）**

ＦＬＡＳＨの場合　ＥＥＰＲＯＭとして、Ｆｉｃｓのプログラム（パラメータ）のバックアップ（セーブ）にも使われている。

ＲＡＭ
ＣＰＵからみて、読み書きできるもの

ＣＸ　＝　＿ｐｇｍ＿ａｒｅａ＿ｂｙｔｅ

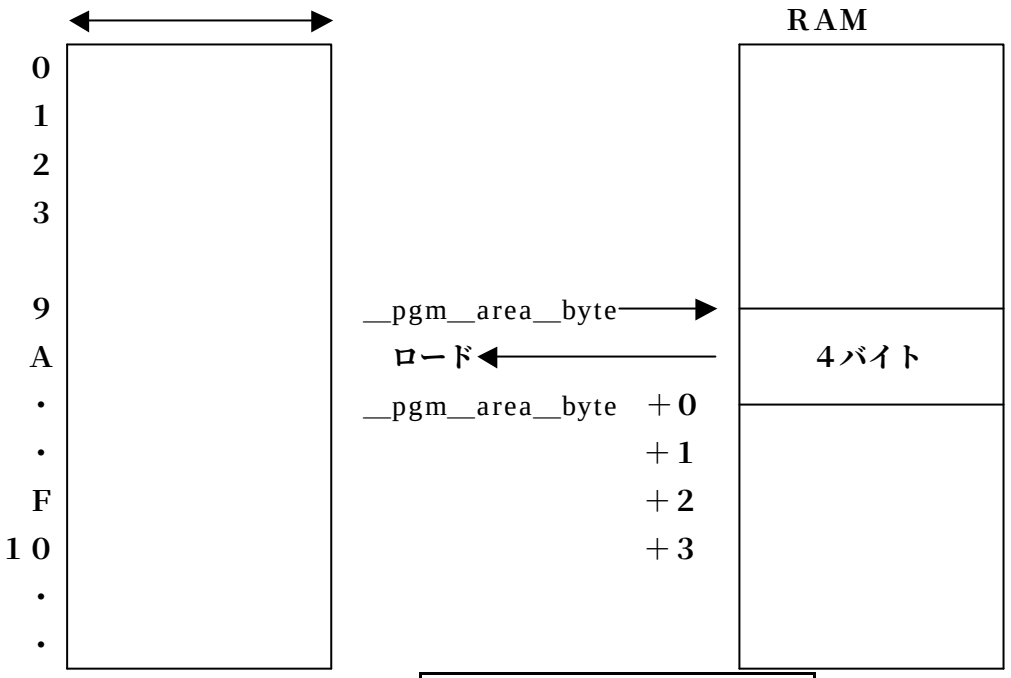
＿ｐｇｍ＿ａｒｅａ＿ｂｙｔｅという名前のついたメモリから
レジスタＣＸにそのメモリ内容をもってくる。
「変数」という言葉で、でてくる。



【バイト・アドレス】

０番地　　１ｂｙｔｅのメモリ

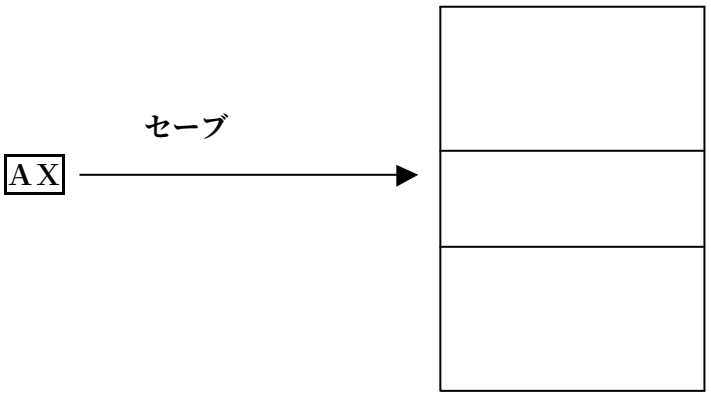
８ﾋﾞｯﾄ＝１ﾊﾞｲﾄ




ＡＸ＝__ｐｇｍ__ａｒｅａ__ｂｙｔｅ

の内容をもってくる

__ｐｇｍ__ａｒｅａ__ｂｙｔｅ＝ＡＸ



ＢＸ　＝　＆　Ｄ１４＿ｃｕｒ　　＆が付くことで、アドレスを意味する。

これが付くとアドレスを持ってくる。

ＲＷＫ１…　レジスタ

ＲＷＫ１　＝　＆　ｉｃｆｌｇ［０］

アドレスをつくる

@（ｂｙｔｅ）　ＲＷＫ１［０］=@（ｂｙｔｅ）ＳＩ［１］

ここが指すメモリの０番地目の１バイトに代入する

@（ｂｙｔｅ）ｉｃｆｌｇ［０］＝　@（ｂｙｔｅ）ＳＩ［１］と同じ意味



ＳＩ＝ｐａｄｒｓｖ

（アドレス）

ＳＨＡＬ（Ｖ５３）でアドレスをもってくる

ｂｘ　＝　ｏｆｆｓｅｔ　Ｄ１４＿ｃｕｒｓ

アドレスである

ＳＨでアドレスを持ってくる

ＢＸ　=＆　Ｄ１４＿ｃｕｒ

これがつかないと中身（内容）をもつ

【　フィールド定義テーブルの構造（C_ＴＢL）　】

C_PGM/C_PRM　画面ごとに 1 つ存在し、C_TBL の前に置かれる。

＜定義例＞

macro　C_PGM（）{

　　0X00

]

　　/*　PARAMETER 等ｃ_top にアドレスが必要なもの

＜マクロ定義＞

C_PGM/C_PRM/C_TBL　macro（マクロ定義の最初）

　　endm（マクロ定義の最後）

＜マクロ展開＞（Ｖ５３）

DB　　？（ﾃﾞﾌｧｲﾝﾊﾞｲﾄ）：１バイト確保する。場所だけ確保

　　０　ｂ７ｂ６ｂ５ｂ４ｂ３ｂ２ｂ１ｂ０変数指定ビット情報

　　<b0>　C_off　　指定　0:[C_OFF]＝Offsetb　icflg

　　1:続く　２バイトで指定

　　<b1>　CRT_OFF　指定　0:別途　独立に指定　　CRT 以外では強制的に０

　　1:続く　２バイトで指定

　　<b2>　V_FLAG　　指定　0:横方向指定　　CRT 以外では強制的に横方向

　　1:縦方向指定

　　<b3>　scale　　指定　0:SCALE＝１　　CRT 以外では強制的に SCALE＝１

　　1:続く　１バイトで指定

DW　？（ﾃﾞﾌｧｲﾝﾜｰﾄﾞ）：１ワード＝２バイトの確保

DW　？　　１　　変数指定ビット情報の<b0>=1 のとき、C_OFF

　　<b0>=0 のとき、実態なし

DW　？　　３　　変数指定ビット情報の<b1>=1 のとき CRT_OFF

　　<b1>=0 のとき、実態なし

DB　？　　５　　変数指定ビット情報の<b3>=１のとき、CRT_SCALE

　　<b3>=0 のとき、実態なし

このすぐ後に、C__ＴＢＬが連続する。

＊上記情報により fld_ｃ_top_set（）で各種フィールド変数を設定する。

＜定義例＞

V53（MINO．Ｖ２￥USRLOGIC.S）

1618　SPCYL_cur:　C_PGM　　０，@yoko、@scale1

のすぐ後に C_TBL が連続する。

**ＳＨ**（mino004￥Minoopt.sh）

```
48  Byte                MINO_AIAO_cur[＊]
49  = {
50      C_PGM()
```

**のすぐ後に** C_TBL **が連続する。**

**フィールド入力定義テーブルの全てのタイプ**

**ＲＴ：　ＲＴ１用**

**ＬＣＤ：　旧機器（ＨＣ３）のために残っているが、実際は使わない。**

**ＣＲＴ：　ＳＨにおいては、ホスト対応ＴＸオプション用に使われる。**

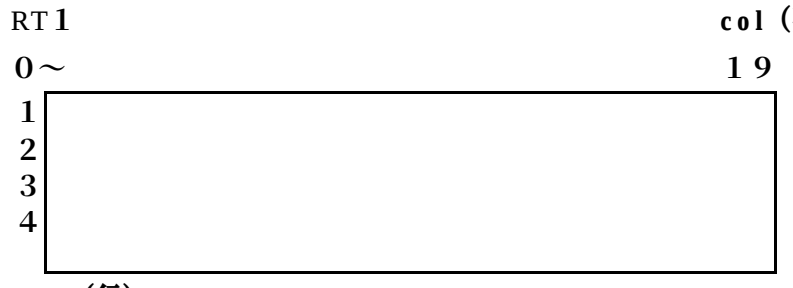
**Ｃ１ＴＸ**

**を共通化する。**

C_TBL**(crt_row,crt_col,lcd_row,lcd_col,rt_row,rt_col,Tno,Off,TextNo)**

**として左から右に順番に当てはめていく。**　（macro，endm）

**＜マクロ展開＞としての構造を図で表わしてみると**




**となる。**

**＜ＬＣＤ 行／　カラム＞：ｌｃｄ_ｒｏｗ／ｌｃｄ_ｃｏｌ**　LCD，RT **のときにのみ有効**

LCD，RT　**画面上の入力位置**

**行**

**カラム**

**＜ＣＲＴ 行／　カラム＞：ｃｒｔ_ｒｏｗ／ｃｒｔ_ｃｏｌ**　CRT，ﾊﾞｯﾌｧ**上のｵﾌｾｯﾄ位置**

**横方向入力時**

**＜行＝ｒｏｗ＞**

**＜列＝ｃｏｌ＞**

＜Tno＞　入力制御情報テーブル（FLDTBL）のレコード番号である。

定義例としてSHにおける

C_TBL　　（1，0，１，0，3，13，_Fn_var，4，０）

の[TNO]の場所にある_Fn_var の内容は、
ファイル名 filed.inc を開き

```
    /* field_ctrl_table の番号
   /* ﾌｨｰﾙﾄﾞ No.
#define   _Fn_VAR          49
```

を探し、更にファイル名 std_text.inc 中の、

(V53)

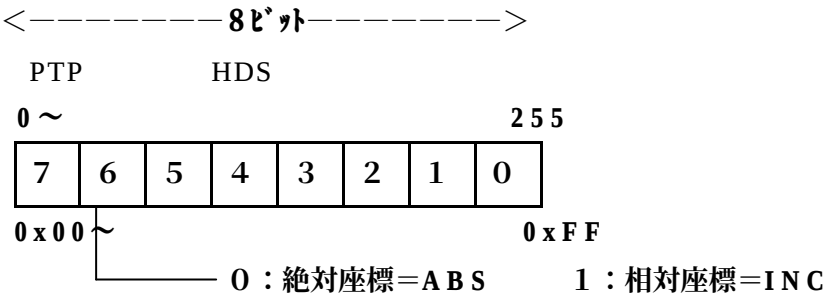
, FLDDEFN(_DN, 0, 0, 0,0, 0, 2, 11, 1)　/* 49 @Fn_Vn 10 進 2 桁(1～96)変数

SH では Fn_Vn

を探し　_Fn_VAR は　の 49@Fn_Vn 10 進 2 桁(1～96)変数とわかる。

*/入力制御情報テーブル（図示）

```
+->変換 @DN:10 進数
/*              |       @BN: 2 進数
/*              |        @HN:16 進数
/*              |
/*              |   +->符号付き数
/*              |   |   +->ﾌﾟﾗｽ符号表示
/*              |   |   |   +->小数点入力　0:不可　1:可
/*              |   |   |   |   +->前ｾﾞﾛ
/*              |   |   |   |   |   +->小数点以下桁数
/*              |   |   |   |   |   | +->有効桁数
/*              |   |   |   |   |   | |         +->検査ﾙｰﾁﾝ
/*              |   |   |   |   |   | |         |   +->ﾃﾞｰﾀ長
/*             +---+--+--+--+--+--+-------+--+---------------;
/* FLDDEFN @DN, 0, 0, 0, 0, 0,2,       11, 1                ;数値型
```




**0x00**[0]（_Fn_DMY）時、カーソルはそこにあるが filed ルーチンとして表示及び入力はしない。

**0xFF**[255]（_Fn_FFh）の時、外部ルーチン　Tno_ff_text_get（特別な処理をする）を呼び出す。このとき、入力の種類を[Off]の項目に設定して置く。

例　**ABS/INC** のときは**_Tf_ptp_a**

＜**Off**＞

[c_off]からのｵﾌｾｯﾄ位置を設定する。

**16** バイトデータの１ステップデータの何番目かを示す。

V53 の書き方　[c_off]には **Offset**　icflg

**Offset**　X_Parameter 等が設定されている。

SH では　[c_off]には**&**icflg

**&**X_Parameter

となる。

パラメータは軸ごとに構造体で定義し、その名前を設定する。

0ff＝255（offth）の時、結果を nobuf（文字型時）

または Lacc（数値型）に返す。

定義例として＜**Off**＞（太字）が 16 バイトデータの１ステップデータの何番目に入るか

C_TBL　1, 0, **1**, 0, 3, 13, @Fn_var, **4**, **0**

C_TBL　1, 0, **1**, 0, 4, 12, @Fn_var, **10**, **0**

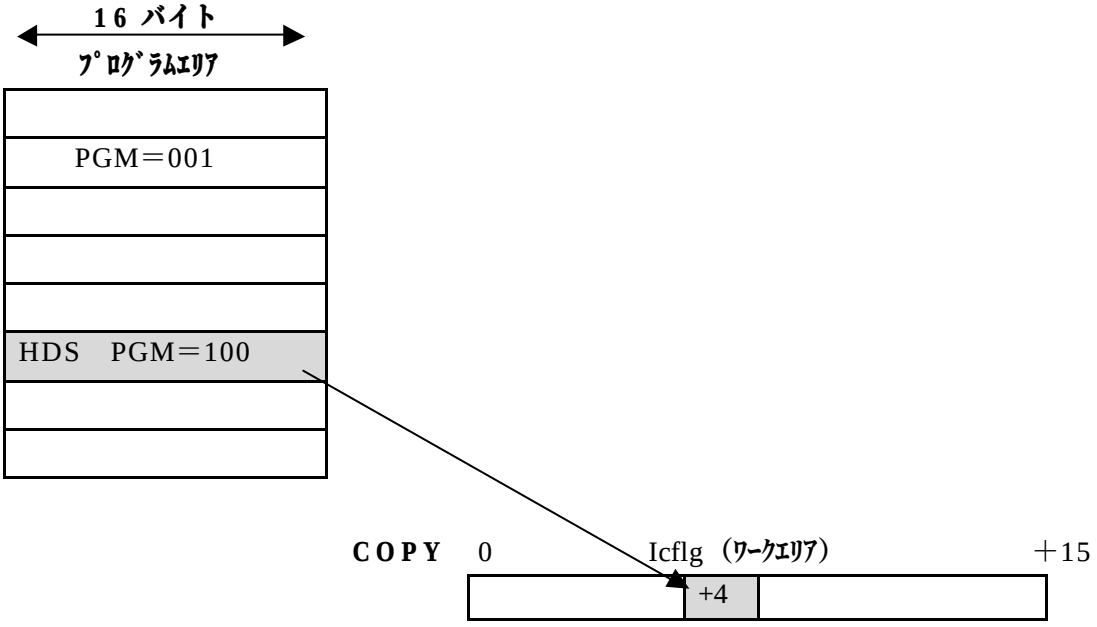
C_TBL　1, 0, **1**, 0, 4, 18, @Fn_var, **12**, **0**

＜16 バイトデータ＞

| +0 | +1 | +2 | +3 | +4 | +5 | +6 | +7 | +8 | +9 | +10 | +11 | +12 | +13 | +14 | +15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HDS | | | | X 軸移動ﾊﾟﾙｽ数変数番号 | | | | | | X 軸ﾎﾟｲﾝﾄ 1 補間ﾊﾟﾙｽ数変数番号 | | X 軸ﾎﾟｲﾝﾄ 1 補間ﾊﾟﾙｽ数変数番号 | | | PTP ﾌﾗｸﾞ |
| | | | | | | | | | | | | | | | |

図で見ると 16 バイトデータの+4，+10，+12 にそれぞれ当てはまり TNo が Off のメモリに確保される。

図で表わすと




となる。

**TextType(TextNo)**

入力フィールドの前に出力するﾃｷｽﾄの番号を指定する。

bit7 の時には CRT（プリンタ）出力時には表示しない。

1:X＝　　2:Y＝　　3:VAR　　4:＝VAR　　5:DI:

6:FLAG　　7:....　　8:TIMER　　＝...

fld_disp 呼び出し時に、フィールドの直前に出力される。

外部ルーチン　TextPointerGet（）を呼び出す。

**定義例として、ファイル名：MINO.V２¥USRLOGIC.S（V５３）とmino004¥Minoopt.sh（SH）を例とする。**

**V５３**（SPCYL_cur：）**の場合**

```
:
  SPCYL_cur:   C_PGM     0, @yoko, @scale1
        C_TBL     1, 0, 1, 0, 3, 13, @Fn_var, 4, 0          X:MAX
        C_TBL     1, 0, 1, 0, 4, 12, @Fn_var, 10, 0         X:DDA①
        C_TBL     1, 0, 1, 0, 4, 18, @Fn_var, 12, 0         X:DDA②
        C_TBL     1, 0, 1, 0, 4,  0, @Fn_FFh, @Tf_ptp_a, 0
        db      0FFh
        $CRTEMPTY
        $LCDEMPTY
        $RT_TEXT       2, 0, < 8, ‘ STPCYL ’,          000h  >
        $RT_TEXT       3, 8, < ‘ ァ=MAX ’,             000h  >
        $RT_TEXT       4, 4, < ‘ (1), DDA00-DDA ’      000h  >
```

**SH**（MINO_AIAO_cur[＊]）**の場合**

```
code {
        byte     MINO_AIAO_cur[＊]
    = {
    C_PGM()
    , C_TBL(1,25, 0,0, 3,18, _Fn_D2  , 2              , 0       )
    , C_TBL(1,32, 0,0, 4,13, _Fn_D1_4, 4              , 0       )
    , C_TBL(1,39, 0,0, 4,18, _Fn_VAR , 6              , _Tp_Var )
    , C_TBL(1,11, 0,0, 2,13, _Fn_FFh , _Tf_MFB, 0             )
    , 0xFF
    , $CRTTEXT(1, 6, “ATOM-DI      STATION=00  CH=0”, 0xFF)
    , $RT_TEXT(2, 8, “ATOM-DI”          0x00)
    , $RT_TEXT(3, 8, “    STATION=00”   0x00)
    , $RT_TEXT(4, 10, “CH=”             0x00)
    };
```

**下線部は画面表示内容のテキスト（固定されている。）**

$CRTTEXT の ‘ TEXT ’ のﾃﾞﾘﾐﾀは 00h：**テキストの区切り**

**0FFh：テキストの終了**

**$CRTEMPTY　表示文字列が無い時の指定**

**$LCDEMPTY　表示文字列が無い時の指定**

**$RT_EMPTY　表示文字列が無い時の指定**

row，col **で指定された場所から表示する。**

CRT **用の時には、**$LCDTEXT **は何も展開されない。（RT1 用）**

C_TBL　**（crt_row，crt_col，lcd_row，lcd_col，rt_row，rt_col，Tno，Off，TextNo）を V53　（**SPCYL_cur:C_TBL**）の構造に対して左から（　）内の（,）ごとに、それぞれに当てはめてみると（太字部分）**

SPCYL_cur:　C_PGM　　**0**，@yoko，@scale **1**

　　C_TBL　**1，0，1，0，3，13，@Fn_var，4，0**　　;X:MAX

**に当てはまる。**

**SH**（MINO_AIAO_cur[＊]）**で当てはめてみると（太字部分）**

code｛

　byte　　MINO_AIAO_cur[＊]

　={

，C_TBL（**1，25，0，0，3、18，_Fn_D2　，2　　，0　　**）

**にそれぞれ当てはまる。**

**備考：**