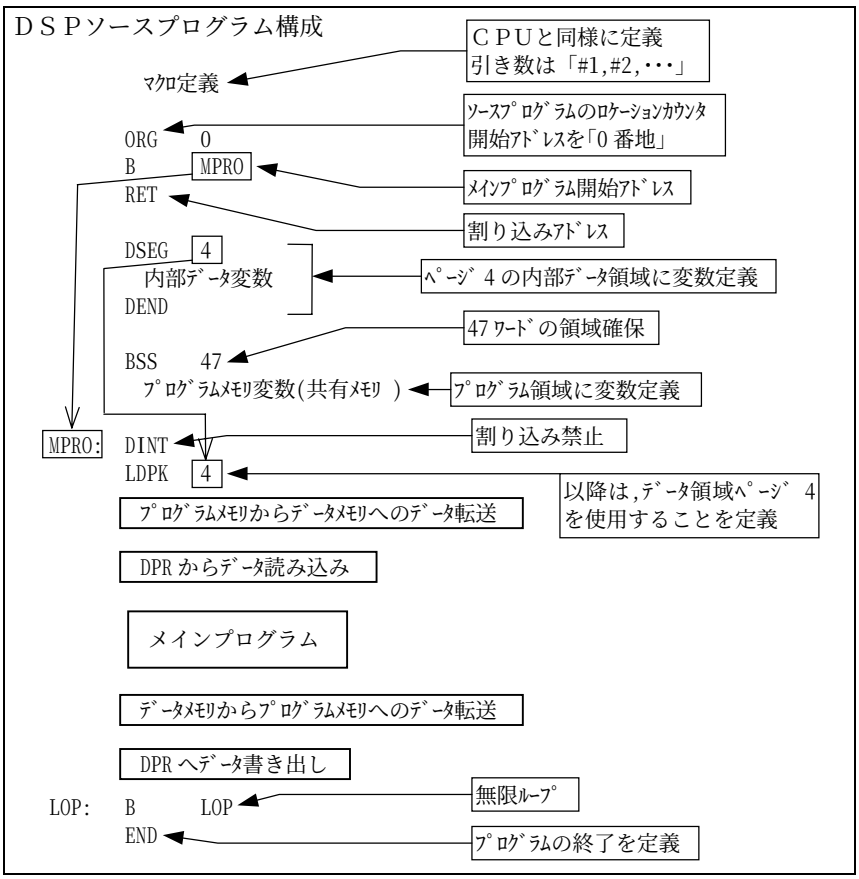
１８．ＤＳＰプログラムと実行

ＤＳＰにおいてもＣＰＵと同様にアセンブラによってソースプログラムを作成する。ＤＳＰの命令はニーモニックやオペランドは異なるが，基本的にＣＰＵアセンブラと同じである。ＤＳＰアセンブラ命令で使用する変数（オペランド）はほとんど「内部データメモリ」に定義した変数を対象としている。ＣＰＵからは「ＤＳＰ内部データメモリ」は直接アクセスできないために，ＤＳＰはＣＰＵからプログラムメモリに書き出されたデータを全て「内部データメモリ」へ転送する必要がある。また，演算結果をＣＰＵが読み出せるように「内部データメモリ」からプログラムメモリへ転送する操作も行わなければならない。これらを含めたＤＳＰソースプログラムの基本構成は図のようである。



まず，必要ならば「マクロ定義」を行う。マクロ定義はＣＰＵの場合と同様に「ﾏｸﾛ名：MACRO 引き数 ～ ENDM」で行うが，「引き数名」は「#1，#2，･･･」である。「ORG 0」はソースプログラムのロケーションカウンタを「０」にし，開始アドレスを「０番地」に定義する。前述したように，ＤＳＰはＣＰＵによってスタート，ストップさせられ，ＤＳＰプログラムは０番地からスタートするが，２～３１番地は割り込みベクタ等に予約されており，ソースプログラムの記述はできない。「０番地」にはＤＳＰスタート時にジャンプする「メインプログラム開始アドレス」への分岐命令を記述する（メインプログラム

の開始アドレスラベルを「MPRO」としている）。２番地は割り込みベクタであり，本プログラムでは使用しないが，一応「RET」を記述する。

「DSEG ～ DEND」は「内部データメモリの変数」を定義する（プログラムメモリではない）。この変数はソースプログラム上では存在するが，最終的な機械語レベルでは存在しない。データメモリのページは「４」を使用し，定義できる最大は１２８ワードである。これを越える変数の定義を行うには別のページ，例えばページ５を定義する。

次に，ＣＰＵとの「共有メモリによるデータ転送」のためのデータ領域を定義するが，３１番地まではシステム予約であるため，それ以降に定義しなければならない。ここでは，５０番地からデータ変数を定義するために

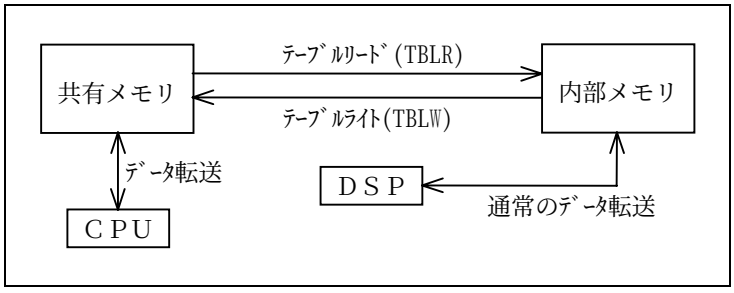
「ＢＳＳ　４７」

を記述し，４７ワードの領域を確保する。これは「B MPRO」が２ワード，「RET」が１ワードの命令であり，これらを合わせて５０ワードとなっている。この定義の次のアドレスが５０番地である。

実行プログラムは「MPRO: DINT」からであり，まず，割り込み禁止を記述する（本プログラムでは割り込みを使用していないので，省略してもよい）。「LDPK 4」を実行し，以降で使用するデータメモリが「ページ４」であることを定義する。

プログラムでは「テーブルリード（Table Read）」命令によってＣＰＵから書き込まれたデータや演算で使用する定数を内部データメモリへ転送する。その後，必要ならばＤＰＲからデータを読み込む。メイン処理プログラム演算終了後，次回の演算に必要なデータやＣＰＵへ転送するデータ（これらは内部データに保存されている）を「テーブルライト（Table Write）」命令によってプログラムメモリの変数領域に転送する。演算が終了したことをＣＰＵに知らせるためにＤＰＲへデータを出力する（ＤＰＲへのデータとして何もなければダミーデータを出力する）。その後は，ＣＰＵがＤＳＰの演算終了を確認してストップするので，ＤＳＰは無限ループを実行して待機する。ソースプログラムの最後は終了を「END」で定義する。

ＤＳＰ「内部データメモリ」と「プログラムメモリのデータ領域」の関係は右図のようであり，内部メモリはＣＰＵから直接アクセスできない。したがって，共有メモリと通して内部メモリへ転送を行う。「テーブルリード」と「テーブルライト」命令はこれを行う命令である。「テーブルリード」命令は次のように記述する。




```
LACK ﾌﾟﾛｸﾞﾗﾑ領域変数ｱﾄﾞﾚｽ
TBLR 内部ﾃﾞｰﾀ変数ｱﾄﾞﾚｽ
```

これは, 内部データ変数アドレスの内容を ACC で示されるプログラム領域の変数アドレスに格納するという操作を行う。それぞれのアドレスはアセンブラではラベルを用いて表現できる。また，「テーブルライト」命令も同様に

```
LACK ﾌﾟﾛｸﾞﾗﾑ領域変数ｱﾄﾞﾚｽ
TBLW 内部ﾃﾞｰﾀ変数ｱﾄﾞﾚｽ
```

のように記述し，ACC で示されるプログラム領域の変数アドレスの内容を内部データ変数アドレスに格納する。

ＤＳＰでは演算開始前および終了後にＣＰＵとの転送データ，保存しておく必要のある全てのデータに対して，この処理を行わなければならない。通常，内部データメモリに定義したものと同じ変数をプログラムメモリにも定義するので，プログラムメモリの変数名は内部データメモリ変数名の先頭に「Ｘ」を付けた名前としている。

```
/* EX181.C */
#include <process.h>
#include “b:\cfile\crt.h”
extern int a,b,c;
main()
{
clstxt();
/* DSP Pro.Loading */
outp(0x02d0,0);
outp(0x02d2,(inp(0x02d2) & 0xc0));
outp(0x02d2,(inp(0x02d2) | 0x40));
spawnlp(P_WAIT,“d:ld25”,“d:ld25”,
        “b:\\dfile\\ex181.oj”,“-m”,“c000”,NULL);
/* data input */
printf("a=");scanf("%d",&a);
printf("b=");scanf("%d",&b);
mult();
printf("c=%d\n",c);
}
```

```
;EX181.A32
        ORG  0
        B    MPRO
        RET

        DSEG 4
         VA:  BSS 1
         VB:  BSS 1
         VC:  BSS 1
        DEND

        BSS   47
         XVA: BSS 1
         XVB: BSS 1
         XVC: BSS 1

MPRO:   DINT
        LDPK 4
        LACK XVA
        TBLR VA
        LACK XVB
        TBLR VB

        LT   VA
        MPY  VB
        PAC
        SACL VC

        LACK XVC
        TBLW VC

        OUT  VC,PA1
LOP:    B    LOP
        END
```

```
;EX181A.ASM
DGROUP  group  mdata
mdata   segment word public 'data'
        assume  ds:DGROUP
        public  _a,_b,_c
_a      DW  ?
_b      DW  ?
_c      DW  ?
mdata   ends

_TEXT   segment byte public 'CODE'
        assume  cs:_TEXT,ds:DGROUP
        public  _mult
_mult   proc near
; DATA Trans.
        MOV  AX,0C000H
        MOV  ES,AX
        MOV  AX,_a
        MOV  BX,100
        MOV  ES:[BX],AX
        MOV  AX,_b
        ADD  BX,2
        MOV  ES:[BX],AX
; DSP Start
        MOV  DX,02D2H
        IN   AL,DX
        AND  AL,0BFH
        OR   AL,01H
        OUT  DX,AL
; DPR Check
        MOV  DX,02D2H
DWAIT:  IN   AL,DX
        AND  AL,10H
        JZ   DWAIT
        MOV  DX,02D4H
        IN   AX,DX
; DSP Stop
        MOV  DX,02D2H
        IN   AL,DX
        OR   AL,40H
        AND  AL,0FEH
        OUT  DX,AL
; DATA Trans.
        MOV  AX,0C000H
        MOV  ES,AX
        MOV  BX,100
        MOV  AX,ES:[BX]
        RET
_mult   endp
_TEXT   ends
        end
```

サンプルとして共有メモリを使用し，２つの数値を入力してその積を求めるプログラムを示す。入力はＣ言語「EX181.C」で行い，演算はＤＳＰで行う。ＣＰＵアセンブラプログラムは「EX181A.ASM」，ＤＳＰソースプログラム名は「EX181.A32」（ＤＳＰソースプログラムの拡張子は「.A32」である）とする。

ＣプログラムではＤＳＰボードの設定を行い，６章で説明した「LD25.EXE」を子プロセスとして実行し，後述するＤＳＰのマシン語プログラムをロードする。なお，ＤＳＰのプログラムはドライブＢのサブディレクトリ「DFILE」に作成してあるものとする。ＣおよびＣＰＵアセンブラソースプログラムをコンパイル，リンクし実行可能ファイル「EX181.EXE」を作成する。

一方，ＤＳＰソースプログラムも次の手順でコンパイルおよびリンクを行い，実行可能ファイル（ローディング可能ファイル）を作成する。まず，ＤＳＰコンパイラ「ASM25.EXE」はドライブＤにあるので，次のように入力する。

カレントドライブＡの時

```
A:\>D:ASM25 B:\DFILE\EX181[.A32] [-l] [-s]
```

カレントドライブＤの時

```
A:\>ASM25 B:\DFILE\EX181[.A32] [-l] [-s]
```

[ ]は省略可能である。入力後

```
TMS320C25 DSP Cross Assembler Version 1.11
(C)Copyright 1989,1990,1991 AE Corporation

 Total error : 0

D:\>
```

を表示してコンパイルを終了する。エラーがある場合にはその内容を表示する。これによりドライブＢのサブディレクトリ「DFILE」内にリロケータブルファイル「EX181.RL」が作成される。なお,

「-l」指定時は「リストファイル　EX181.LST」

「-s」指定時は「シンボルファイル　EX181.SYM」

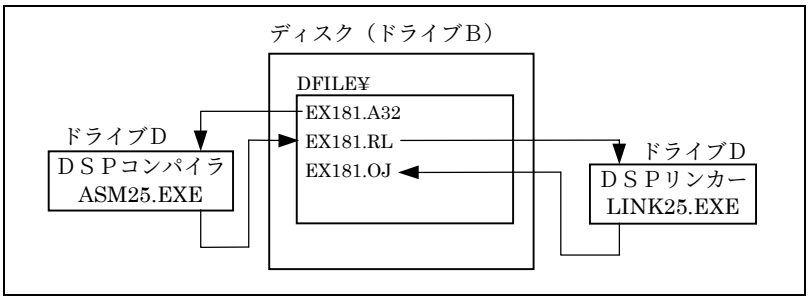
を同一のディレクトリ内に作成する。

次にリンクを行う（カレントドライブＤの場合）。

```
D:\>LINK25 B:\DFILE\EX181[.RL] [他のファイル[.RL] ・・・・]
TMS320CXX DSP Linker Version 1.06
(C)Copyright 1989,1991 AE Corporation

D:\>
```

[ ]は省略可能であり，他のリロケータブルファイルとリンクする時は並べて入力する。ローディング可能な「オブジェクトコードファイル「EX181.OJ」」が同一ディレクトリに作成される。これらの作業状態は下図のようである。



リストファイル「EX181.LST」を作成した時のその内容は下のようであり，プログラムとマシン語およびアドレスの関係が理解できるであろう。アセンブラでの変数はマシン語では，そのアドレスに置換されている。内部データメモリでの変数アドレスは，プログラムメモリのアドレスとな異なって作成されている。

```
EX181.LST                                    ｿｰｽﾌﾟﾛｸﾞﾗﾑの行番号
                  ｱﾄﾞﾚｽ
                                  ﾏｼﾝ語
0001  0000  0000          ORG   0
0002  0000  ff80          B     MPRO
      0001  0035
0003  0002  ce26          RET
0004
0005                      DSEG  4
0006  0200                 VA:  BSS  1
0007  0201                 VB:  BSS  1
0008  0202                 VC:  BSS  1
0009                      DEND
0010
0011  0003                BSS   47
0012  0032                 XVA:  BSS  1
0013  0033                 XVB:  BSS  1
0014  0034                 XVC:  BSS  1
0015
0016  0035  ce01    MPRO:DINT
0017  0036  c804          LDPK  4
0018  0037  ca32          LACK  XVA
0019  0038  5800          TBLR  VA
0020  0039  ca33          LACK  XVB
0021  003a  5801          TBLR  VB
0022
0023  003b  3c00          LT    VA
0024  003c  3801          MPY   VB
0025  003d  ce14          PAC
0026  003e  6002          SACL  VC
0027
0028  003f  ca34          LACK  XVC
0029  0040  5902          TBLW  VC
0030
0031  0041  e102          OUT   VC,PA1
0032  0042  ff80    LOP:  B     LOP
      0043  0042
0033                      END
```

内部ﾃﾞｰﾀﾒﾓﾘのｱﾄﾞﾚｽ
ﾍﾟｰｼﾞ 4

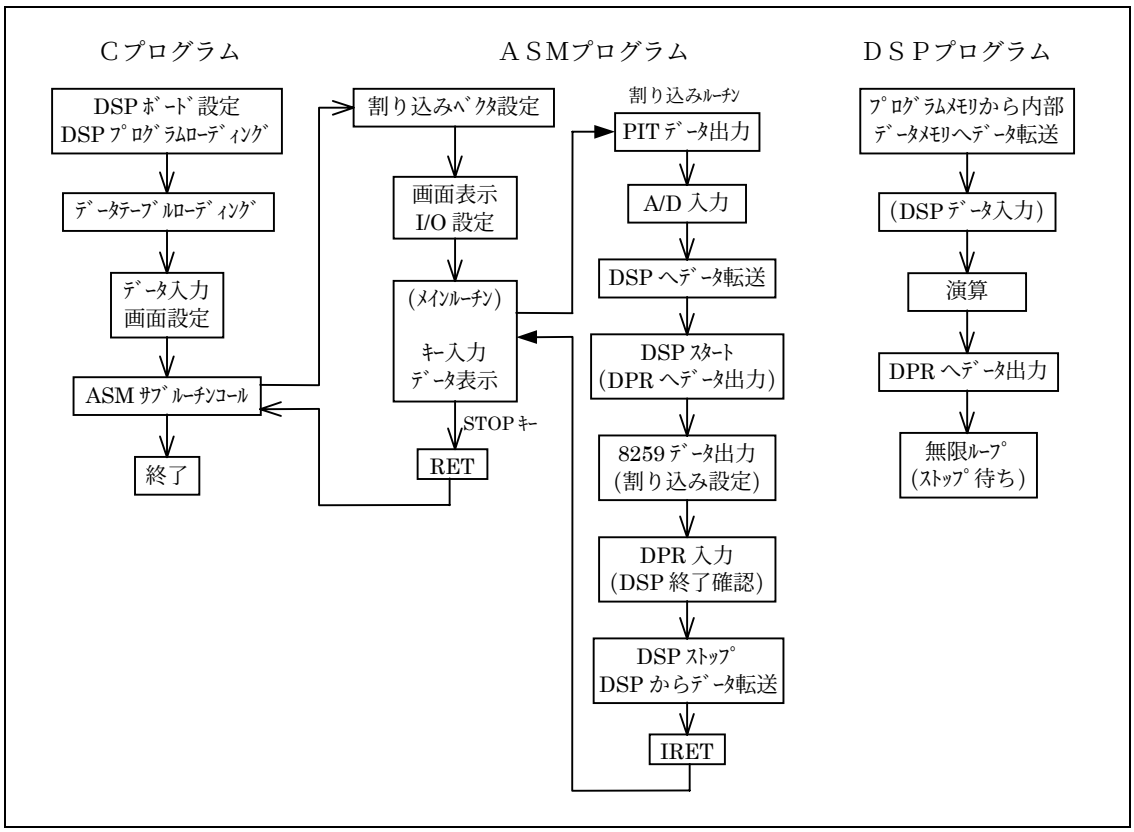
ｱﾄﾞﾚｽ 100 番地

ﾏｼﾝ語 ca32 の 32 は変数 XVA アドレスの下位８ビットを示す

ﾏｼﾝ語 5800 の 00 は変数 VA アドレスの下位８ビットを示す

0042 は分岐命令のジャンプ先アドレスすなわち，１つ前のアドレスを示す

実際の制御プログラムでは割り込み処理を行い，割り込みプログラムでＤＳＰのスタート，ストップおよびデータ転送を行う。割り込みルーチンではＣＰＵとＤＳＰが同時に実行を行っており，一種のマルチコンピュータシステムとなっている。割り込みを行う制御プログラムのそれぞれの処理の流れは下図のようである。



以下，ＤＳＰ（TMS320C25）の命令は本テキストの最後に全てを記載しているが，ここでは現在使用している命令を説明する。なお，アセンブラ命令は全て直接アドレッシングを使用しており，補助レジスタは使用していないので，これについては説明を省略する。命令の詳細はマニュアルを参照されたい。

ＤＳＰ命令の直接アドレッシングでは

［ニーモニック］　［第１オペランド］

あるいは

［ニーモニック］　［第１オペランド］，［第２オペランド］

で表される。「第一オペランド」には内部データメモリアドレス「dma」あるいはプログラムメモリアドレス「pma」を指定するが，アセンブラでは変数名を記述する。また，「第２オペランド」には「第１オペランド」に対する「左シフトのビット数（Ｓ）」を指定し，デフォルト（省略時）は「０」である。すなわち，第２オペランドで示されるデータをＳビット左シフトを行い，ニーモニックの操作を行う。ＤＳＰの演算では符号拡張モードに

初期設定されているので，例えば，１６ビット値をアキュムレータに対して処理を行う時には，まず，１６ビット値を３２ビットに符号拡張して演算を行う。

［ＡＣＣの加算，減算命令］

ADD <dma> [,<S>]	SUB <dma> [,<S>]
ADDH <dma>	SUBH <dma>
ADDS <dma>	SUBS <dma>
ADDK <C8>	SUBK <C8>
ADLK <C16> [,<S>]	SBLK <C16> [,<S>]

ADDH,SUBH: ACC の上位１６ビットに対して行う（ACC の下位１６ビットは不変）
ADDS,SUBS: ACC の下位１６ビットに対して行い，<dma>のデータは１６ビット正数として扱う（ACC の上位１６ビットは不変）
ADDK,SUBK: ８ビット正数
ADLK,SBLK: １６ビットの数
ビットシフト数：０≦Ｓ≦１５

［ＡＣＣの絶対値，論理命令，補数］

ABS

ACC 内容の絶対値

ZAC

ACC 内容の０クリア（LACK 0 と同じ）

NEG

ACC 内容の２の補数をとる（正負反転）

AND <dma>	OR <dma>	XOR <dma>
ANDK <C16> [,<S>]	ORK <C16> [,<S>]	XORK <dma> [,<S>]

ANDK,ORK,XORK:１６ビット定数（空いたビットは０づめ）
ビットシフト数：０≦Ｓ≦１５

［ＡＣＣのロード］

LAC <dma>[,<S>]	LACK <C8>	LALK <C16>[,<S>]

LACK は８ビット正数，LALK は１６ビット定数
ビットシフト数：０≦Ｓ≦１５

ZALH <dma>	ZALS <dma>

ZALH は ACC 上位１６ビットにロードし，下位１６ビットを０クリア
ZALS は ACC 下位１６ビットにロード（正数）し，上位１６ビットを０クリア

［ＡＣＣのストア］

SACH <dma>[,<S>]	SACL <dma>[,<S>]

SACH は ACC 上位１６ビットをストア

SACL はACC 下位１６ビットをストア
ビットシフト数：０≦Ｓ≦７（ACC の内容は不変）

［ＡＣＣのシフト］

SFL	SFR

SFL はACC を１ビット左シフト，SFR はACC を１ビット右シフト

［Ｔレジスタ，Ｐレジスタおよび乗算命令］

Ｔレジスタ（１６ビット）は乗算に対する被乗数レジスタで，乗算はＴレジスタと指定の<dma>（１６ビット）で行われ，結果はＰレジスタ（３２ビット）に保存される。

SPH <dma>	SPL <dma>

SPH はＰレジスタの上位１６ビットをストア，SPL はＰレジスタの下位１６ビットをストア

PAC	APAC	SPAC

PAC はＰレジスタの内容をそのままACC へストア
APAC はＰレジスタの内容をACC へ加算
SPAC はＰレジスタの内容をACC から減算

LT <dma>		
LTA <dma>	LTP <dma>	LTS <dma>

LT は<dma>内容をＴレジスタへロード
LTA は<dma>内容をＴレジスタへロードし，Ｐレジスタの内容をACC へ加算
LTP は<dma>内容をＴレジスタへロードし，Ｐレジスタの内容をそのままACC へロード
LTS は<dma>内容をＴレジスタへロードし，Ｐレジスタの内容をACC から減算

MPY <dma>	
MPYK <C13>	MPYU <dma>
MPYA <dma>	MPYS <dma>

MPY はＴレジスタと<dma>内容を乗算し，Ｐレジスタへロード
MPYK はＴレジスタと１３ビット定数（符号付き）を乗算し，Ｐレジスタへロード
MPYU はＴレジスタ（正数）と<dma>内容（正数）を乗算し，Ｐレジスタへロード
MPYA，MPYS は今までのＰレジスタの内容をACC へ加算，減算し，MPY を行う

SQRA <dma>	SQRS <dma>

今までのＰレジスタの内容をACC へ加算，減算し，<dma>内容をＴレジスタにロードし，２乗された結果をＰレジスタにストアする

［分岐命令］

B <pma>

指定された<pma>へ無条件ジャンプする

BGZ <pma>	BGEZ <pma>
BZ <pma>	BNZ <pma>
BLEZ <pma>	BLZ <pma>

ACC 内容により，下表に従って指定された<pma>へジャンプする

命令	ｼﾞｬﾝﾌﾟ条件	命令	ｼﾞｬﾝﾌﾟ条件
BGZ	ACC＞０	BGEZ	ACC≧０
BZ	ACC＝０	BNZ	ACC≠０
BLEZ	ACC≦０	BLZ	ACC＜０

［Ｉ／Ｏおよびデータ，メモリ関連命令］

IN <dma>,<PA>	OUT <dma>,<PA>

指定されたポートアドレス<PA>に対して１６ビットデータの入力，出力を行う

TBLR <dma>	TBLW <dma>

TBLR はACC の下位１６ビットで指定されたプログラムメモリアドレス<pma>から，指定する内部データメモリアドレス<dma>へデータを転送する
TBLW は指定する内部データメモリアドレス<dma>のデータをACC の下位１６ビットで指定されたプログラムメモリアドレス<pma>へ転送する