７．アセンブラ言語

ＣＰＵが実行する命令は２進数であるが，この命令と１対１に対応し，プログラマにとってわかり易く表した命令がアセンブラ言語（Assembler Language，以下ＡＳＭという）である。ＡＳＭ命令はＣＰＵの１つの命令を表すので，その操作は非常に簡単であるが，種々の処理を行うには１連の命令の記述，すなわちプログラミングテクニックが必要となる。作成する制御プログラムでは，後述するＤＳＰが種々の演算を行うが，ＣＰＵはＤＳＰを含む外部装置とのデータ入出力や目的とする信号の出力あるいはデータのディスプレイ表示などを行い，制御のタイミングをコントロールしている。これらを行うにはＣＰＵ周辺のハードウェアの知識が必要であり，ＣＰＵの動作を理解しなければならない。ＣＰＵの命令の詳細や外部装置は後節で説明することにし，ここではＡＳＭの基本プログラム構成および実行ファイル作成について説明する。

ＡＳＭプログラムはそれ自信だけで実行ファイルを作成することもできる。しかし，制御に直接関係しないパラメータの設定やディスプレイ初期表示などをＡＳＭで作成することは複雑であるので，これらはＣプログラムで作成する。したがって，ＡＳＭプログラムはＣプログラムの関数（厳密にはサブルーチン）として作成する。ＡＳＭプログラムはＣ言語の一部となるので，そのプログラム形式はＣ言語に合わせる必要がある。その基本構成は右のようである。ソースプログラムは大きく分けて，「データセグメント領域」と「コードセグメント領域」であり，前者では変数の定義を行い，後者でプログラミングを行う。「DGROUP」はＣプログラム内でデータ領域に割り当てられた名称である。「ｸﾞﾙｰﾌﾟ名　segment」でデータ領域を定義し，「ｸﾞﾙｰﾌﾟ名　ends」でこのセグメントの終りを示す。「word」は１ワード単位で領域を確保することを示し，「public ‘data’」はＣプログラム内で共通に使用するための名称である。「assume ds:DGROUP」はこのデータセグメント（ds=DS）名を「DGROUP」と仮定する。一方，「_TEXT segment」はプログラム領域であることを定義，「_TEXT ends」はその終りを示し，「_TEXT」はＣ言語で決められた名前である。「assume」によってコードセグメント（cs=CS）名を「_TEXT」，データセグメント（ds=DS）名を「DGROUP」と仮定している。「byte」は１バイト単位で領域を確保することを示し，「public ‘CODE’」はＣプログラム内で共通に使用するための名称である。「関数名　proc near」でユーザー関数を定義し，「関数名　endp」てその終りを定義するが，この関数はＣソースプログラ

```
;ASM 基本構成
DGROUP    group  ｸﾞﾙｰﾌﾟ名
ｸﾞﾙｰﾌﾟ名  segment word public ‘data’
          assume  ds:DGROUP
             ・
          （変数定義）
             ・
ｸﾞﾙｰﾌﾟ名  ends

_TEXT     segment byte public ‘CODE’
          assume  cs:_TEXT,ds:DGROUP
          public  関数名
関数名    proc near
             ・
          （プログラム）
             ・
関数名    endp
_TEXT     ends
          end
```

ムから呼び出される関数（Ｃプログラム内でも「関数名」を記述する）となるので，「public」によって定義しておく。「proc」は「procedure（手続き）」の意味であり，「near」は同一セグメント内にあることを示している。すべてのプログラムの終了は「end」で定義する。なお，ＡＳＭプログラムでは１行に１命令であり，「；」はそれ以降がコメントであることを示す。

実際にプログラムを作り，実行ファイルを作成するが，ここでは６章（Ｃ言語）で行った「キーボードから２つの値を入力し，その和を表示する（EX63.C）」プログラムを作成する。ただし，演算はＡＳＭで行うようにする。下のソースプログラム「EX71.C」を各自ディスク（ドライブＢ）のサブディレクトリ「CFILE」内に作成する。また，ＡＳＭソースプログラム保存用のサブディレクトリ「AFILE」を作り，その中に「EX71A.ASM」を作成する（２つのソースファイルが共通であることがわかるように，ＡＳＭファイル名はＣファイル名に「A」を付けたものとする）。

```
;EX71A.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  _summ
_summ    proc near
         MOV  AX,_a
         MOV  BX,_b
         ADD  AX,BX
         MOV  _c,AX
         RET
_summ    endp
_TEXT    ends
         end
```

```
/* EX71.C */
extern int a,b,c;
main()
{
printf("a=");scanf("%d",&a);
printf("b=");scanf("%d",&b);
summ();
printf("c=%d\n",c);
}
```

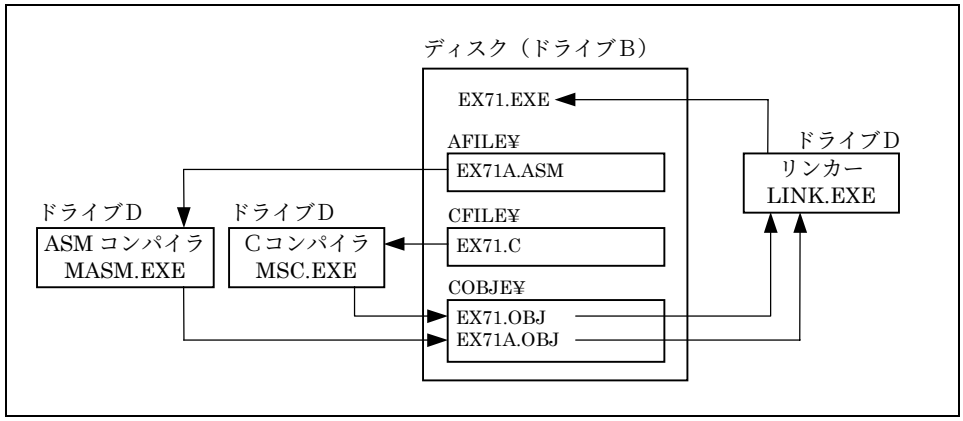
それぞれのプログラムについて簡単に説明する。Ｃソースプログラムでは使用する変数を定義するが，これらはＡＳＭ内でも使用するので，「外部関数でも参照される」ことを示す「extern（external 文）」を定義する。加算を行うＡＳＭの関数を「summ」としているので「summ( );」でその関数を呼び出す。一方，ＡＳＭではＣで指定された変数をデータ領域（ここでは名前を「mdata」とするが，予約変数以外であれば何でもよい）で定義する。ただし，変数名はＣで定義した変数名に「__（アンダーバー）」を付ける決まりになっている。また，これらの変数はＣプログラム内で「int」定義すなわち１６ビット変数としているので，ＡＳＭでは「DW（Define Word）：１ワード定義」とする。コードセグメント内では，まず，関数「summ」がＣから呼び出されることを示す「public」を宣言する。変数と同様にＡＳＭ内では「__」を付ける。「proc」内にプログラムを記述するが，ＡＳＭ命令は，全て

［ニーモニック］　［第１オペランド］，［第２オペランド］

という形式である。「ニーモニック」は命令を表し，続く「オペランド」と呼ばれるもの

がその命令に必要なレジスタやデータ等を示す。詳細な命令の説明は後述するが，「MOV」命令は「第２オペランド」から「第１オペランド」へデータを移動する，「ADD」命令は２つのオペランドの加算を行い，その結果を「第１オペランド」に保存するものである。また，「RET」はその proc から抜け出し，元に戻ることを表し，「Return（リターン）」と呼ぶ。

ＡＳＭソースプログラムもまたＣ言語の場合と同様にコンパイルという作業を行いオブジェクトファイルを作成し，Ｃのオブジェクトファイルと結合（リンク）しなければならない。ＡＳＭプログラムのコンパイラは「マクロアセンブラ（MASM）」を使用する。



カレントドライブをＤとしてＭＡＳＭコンパイラを次の手順で起動する。

```
D:\>MASM
Microsoft MACRO Assembler  Version 3.00
(C)Copyright Microsoft Corp 1981, 1983, 1984

Source filename [.ASM]: B:\AFILE\EX71A
Object filename [EX71A.OBJ]: B:\COBJE\
Source listing  [NUL.LST]:B:\EX71A
Cross reference [NUL.CRF]:

XXXXX Bytes free

Warning Severe
Errors  Errors
0       0

D:\>
```

「Source filename」はＡＳＭプログラムのソースファイル名をディレクトリを含めて指定する。なお，拡張子「.ASM」は省略可能である。

「Object filename」はソースファイル名と同一名で作成するので，ここでは保存するディレクトリ名を指定する。指定せずにリターンキーを押すと現在のカレントに作成される。なお，ここではオブジェクトファイルはＣのそれと同じディレクトリに保存する。

「Source listing」は通常作成しないが，ここでは実際の１６進数によるマシン語記述とアドレスなどを確認するためにドライブＢの親ディレクトリに作成する（あとで消去してよい）。

「Cross reference」のクロスリファレンスは通常作成しないので，そのままリターンキーを押す。コンパイルを開始し，エラーがなければ「free Bytes」と「Warning Errors 0（警告エラー）」および「Severe Errors 0（致命的エラー）」を表示して終了する。起動時に全てのパラメータを設定する場合には，次のように入力する。

```
D:\>MASM B:\AFILE\EX71A,B:\COBJE\,B:\EX71A;
```

次にＣソースファイルも先と同様にコンパイルを行い，ドライブＢのサブディレクトリ「COBJE」に作成し，その中に「EX71.OBJ」および「EX71A.OBJ」があることを確認する。この二つのオブジェクトファイルを「LINK」よって結合し，実行可能ファイルを次のように作成する。

```
D:\>LINK
Microsoft (R) Overlay Linker  Version 3.51
Copyright (C) Microsoft Corp 1983, 1984, 1985, 1986. All rights reserved.

Object Modules [.OBJ]: B:\COBJE\EX71+B:\COBJE\EX71A
Run File [D:EX71.OBJ]: B:
List File [NUL.MAP]:
Libraries [.LIB]:

D:\>
```

「Object Modules」はそれぞれのオブジェクトファイル名をディレクトリを含めて指定し，「＋」で結合する。

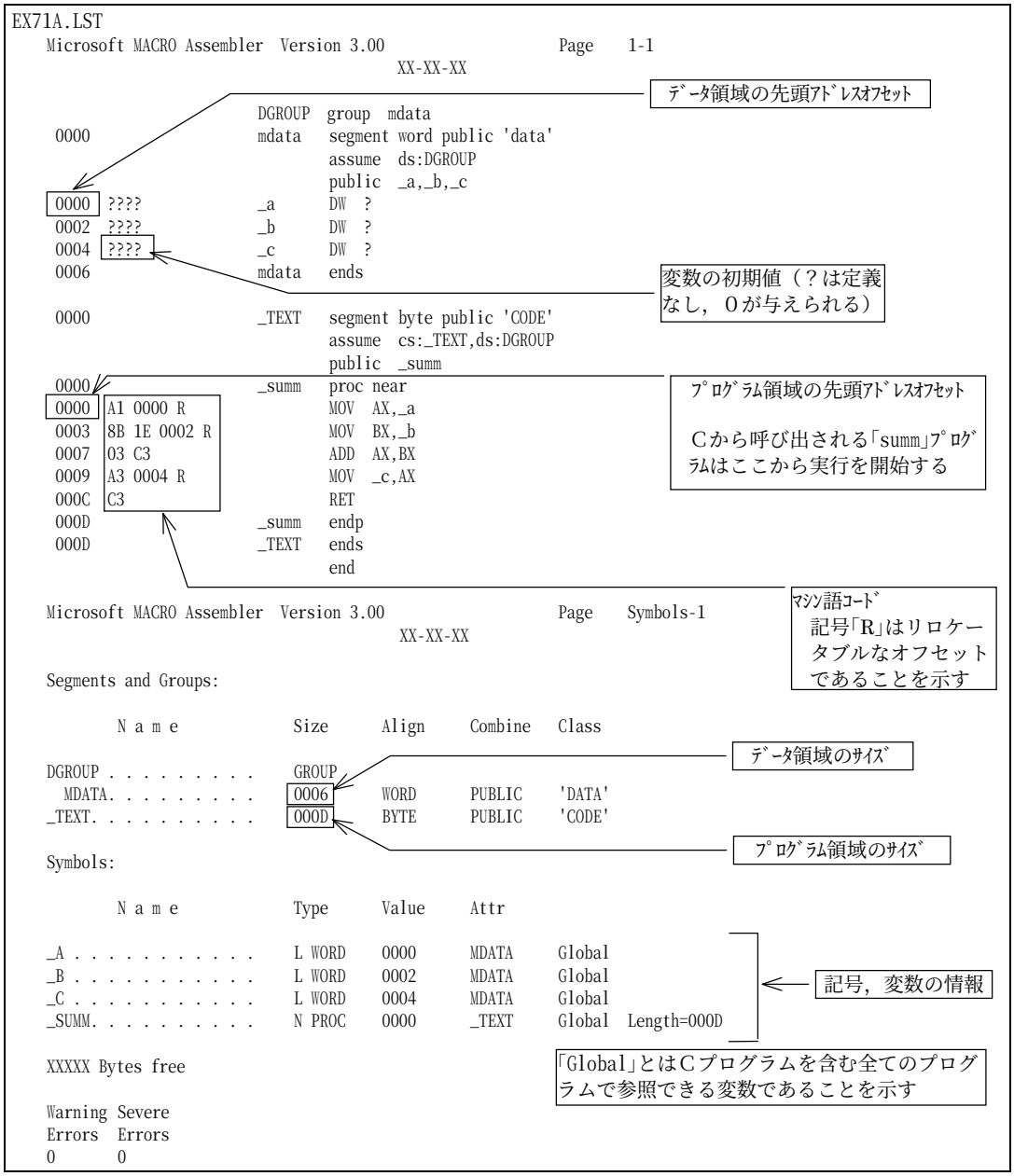
「Run File」はカレントドライブ上に最初に記述したオブジェクトファイル名と同一名で作成するので，ここでは保存するディレクトリ名を指定する。

全てのパラメータを設定して起動する時には，次のように入力する。

```
D:\>LINK B:\COBJE\EX71+B:\COBJE\EX71A,B:;
```

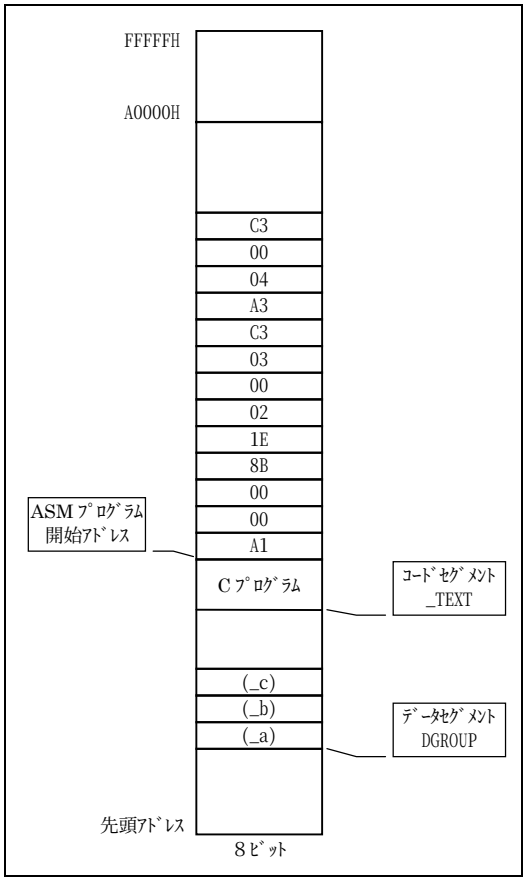
「EX71.EXE」を実行すると，先と全く同じ結果が得られる。色々な値を入力し，「c」が 32768 以上となるような値を入力し，その結果を調べてみよう。

このようにマクロアセンブラではソースプログラム上には全くメモリのアドレス（物理アドレス）が現れないので（だたし，ＶＲＡＭやＤＳＰメモリ等ユーザーが特別なメモリをアクセスする場合にはプログラムにそのアドレスを直接記述する），ユーザーはＣ言語のように全てメモリを変数としてプログラミングできる。コンパイルではアセンブラ記述を全てマシン語に変換し，ＣＰＵが解読できるようにする。実際にコンパイルされたデータがどのようになっているかを見てみよう。先ほどの MASM コンパイルによって作成したソースリスティングファイル「EX71A.LST」をエディタでオープンすると，つぎのようにソースプログラムとマシン語のデータや各種変数の定義などの情報が記述されている。



全てのアドレスは「オフセットアドレス」で表現されている。実際にはこれがＣソースプログラムで記述したオブジェクトファイルと結合され，最終的な実行ファイルが作られる。ＣＰＵがメモリに対してアクセスする時には「物理アドレス」である。Ｃ言語による実行ファイルではコードセグメントは「_TEXT」，データセグメントは「DGROUP」であり，これらのセグメントの値は確定していないが，これらのセグメントがいかなる値をとってもプログラムは実行できる，すなわち，実行ファイルがメモリの何処に配置されても実行できるようになっている。実行する場合にメモリの何処に配置されるかは，ＤＯＳが管理し，その時のシステム状態（メモリ使用状態）を調べて，ローディングを行う。こ

のような実行ファイルを再配置可能ファイル「リロケータブル（Relocatable）」という。リロケータブルファイルは拡張子「EXE」を持つものである。これに対し，拡張子「COM」も実行ファイルであるが,リロケータブルではないファイルである。



右図はＡＳＭプログラムのメモリ配置状態の例を示す。実際は，このプログラムにＣによるプログラムも配置され,その構成はリンカーで決まっており,ＡＳＭの前か後かわからないが，変数はデータ領域，プログラムはコードセグメント領域である。データセグメント「DGROUP」およびコードセグメント「_TEXT」はＤＯＳが実行ファイルをローディングする時に決まり,それぞれ別のセグメントである。したがって，それぞれのセグメントは６４ＫＢの領域をもっている。

下のプログラム（EX72.C と EX72A.ASM）は４章 BASIC の最初に行った VRAM に直接データを書き込んで画面の真ん中付近に「A」の文字を表示するＡＳＭプログラムである。プログラム中にデータを与える場合には，１６進数では終りに「H」を付け，先頭文字がアルファベット（A～F）の時には変数と区別するために「0」を付加する。また，データは数式で記述する（160*12*80）こともできる。キャラクタコードを与える場合には

```
;EX72A.ASM
_TEXT    segment byte public 'CODE'
         assume  cs:_TEXT
         public  _disp1
_disp1   proc near
         MOV   AX,0A000H
         MOV   ES,AX
         MOV   BX,160*12+80
         MOV   AL,41H
         MOV   ES:[BX],AL
         RET
_disp1   endp
_TEXT    ends
         end
```

```
/* EX72.C */
main()
{
disp1();
}
```

```
MOV  AL,41H
```

の部分は

```
MOV  AL,'A'
```

のように，そのまま文字を「‘（シングルクウォーテーション）」で囲んで記述しても，

コンパイルによってキャラクタコードに変換する。

次のプログラム（EX73.C と EX73A.ASM）は６章Ｃ言語の最初で行った「POWER ELECTRONICS」表示（「EX61.C」）をＡＳＭで行うプログラムである（表示は画面中央付近としている，正確な位置はテキストＶＲＡＭの構成から求める）。

```
/* EX73.C */
main()
{
disp2();
}
```

なお，プログラム内容については，後章の命令説明を参考にして各自解読されたい。

```
;EX73A.ASM
DGROUP   group  mdata
mdata    segment word public 'data'
         assume  ds:DGROUP
MOJI     DB 'POWER ELECTRONICS'
mdata    ends

_TEXT    segment byte public 'CODE'
         assume  cs:_TEXT,ds:DGROUP
         public  _disp2
_disp2   proc near
         MOV  AX,0A000H
         MOV  ES,AX
         MOV  BX,160*12+60
         MOV  CX,17
         MOV  SI,OFFSET DGROUP:MOJI
LP1:     MOV  AL,[SI]
         MOV  ES:[BX],AL
         INC  SI
         ADD  BX,2
         LOOP LP1
         RET
_disp2   endp
_TEXT    ends
         end
```