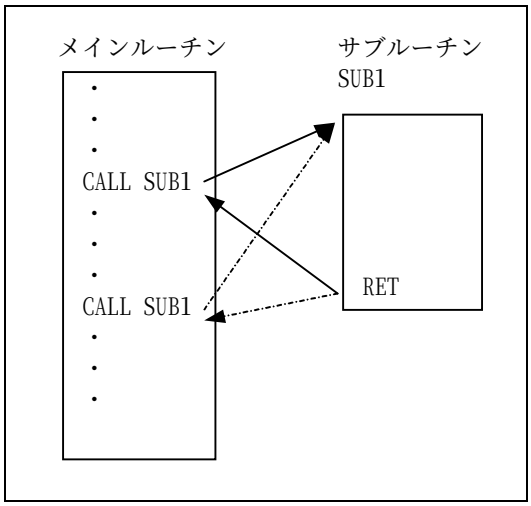
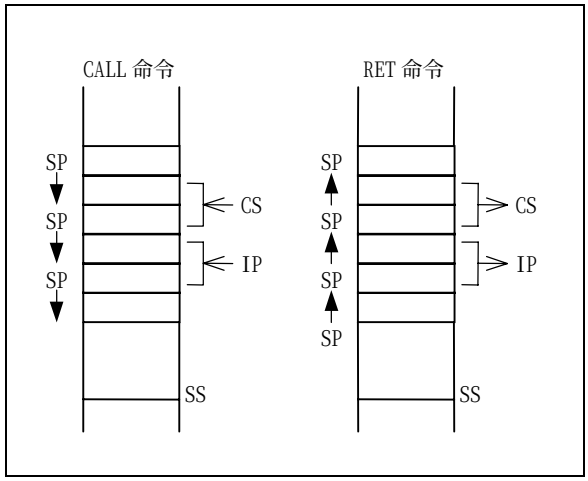
９．サブルーチンとマクロ定義

サブルーチン（Sub Routine）とはメインプログラム（メインルーチン）とは別にブロック化された一連の処理を行うプログラムのことをいう。サブルーチン化するメリットの１つは同じ処理を行うプログラムを何度も記述する必要がなくなること，もう１つはメインプログラムをサブルーチンによってブロック化することにより，プログラムの修正やサブルーチンの入れ替え（結合状態）が容易になることである。



サブルーチンによるプログラムの流れは右図のようであり，サブルーチンへの移行は「CALL」命令で行われ，その終了（メインルーチンへの戻り）は「RET」命令で行われる。

ＣＰＵが行う命令が格納されたメモリのアドレスは「CS（コードセグメント）」と「IP（インストラクションポインタ）」で決定される。「CALL」命令が実行されるとＣＰＵはサブルーチンから戻った時に実行する命令のアドレス CS と IP（CALL 命令の次の命令）を「SS（スタックセグメント）」と「SP（スタックポインタ）」で決まるメモリの物理アドレスへ退避（保存）し，サブルーチンのアドレスを CS と IP にセットする。SP の値は退避されるごとに２づつ減じられる。一方，サブルーチンで「RET」命令が実行されると，先の退避した戻りアドレスをスタックから CS と IP に復帰（ロード）し，次の命令へ移る。なお，サブルーチンがメインルーチンと同一セグメント内に記述される場合（通常はこの記述である）にはスタックには「IP」だけが退避，復帰される。プログラムの実行状態はこのようであるが，ユーザーは特に「SP」の値を考慮しなくてよい（「SS」と「SP」の値はＤＯＳが実行ファイルをローディングする時に決める）。



サブルーチンプログラムを作成する場合には，通常メインプログラムとは別のソースファイルで作成し，それぞれをコンパイルして「LINK」で結合するか，あるいはメインソースプログラム中で「INCLUDE」ディレクティブと呼ばれる関数によって結合して実行ファイルを作成する。両者の方法によって簡単なプログラムを作成してみよう。

０から９までの数字を２列に縦に並べて表示するプログラムを作成するが，まず，「INCLUDE」ディレクティブによって結合する場合を説明する。

```
/* EX91.C */
main()
{
disp1();
}
```

```
;EX91A.ASM
_TEXT   segment byte public 'CODE'
        assume  cs:_TEXT
        public  _disp1
_disp1  proc near
        MOV  AX,0A000H
        MOV  ES,AX
        MOV  AL,30H
        MOV  BX,40
        MOV  CX,10
LP1:    MOV  ES:[BX],AL
        INC  AL
        ADD  BX,00A0H
        LOOP LP1

        MOV  AL,30H
        MOV  BX,60
        MOV  CX,10
LP2:    MOV  ES:[BX],AL
        INC  AL
        ADD  BX,00A0H
        LOOP LP2
        RET
_disp1  endp
_TEXT   ends
        end
```

```
;EX91A1.ASM
_TEXT   segment byte public 'CODE'
        assume  cs:_TEXT
        public  _disp1
_disp1  proc near
        MOV  AX,0A000H
        MOV  ES,AX
        MOV  AL,30H
        MOV  BX,40
        CALL DSUB

        MOV  AL,30H
        MOV  BX,60
        CALL DSUB
        RET
_disp1  endp

DSUB    proc near
        MOV  CX,10
LP:     MOV  ES:[BX],AL
        INC  AL
        ADD  BX,00A0H
        LOOP LP
        RET
DSUB    endp
_TEXT   ends
        end
```

「EX91.C」「EX91A.ASM」はサブルーチンを使用しないプログラムである。ＡＳＭプログラムにおいて，同じような処理を行うプログラムを２度記述している。これをサブルーチン化し，メインプログラムと同一ソースファイル中に記述したＡＳＭプログラムが「EX91A1.ASM」である。サブルーチン名は「DSUB」としたが，予約変数名以外であれば何でもよい。また，大文字でも小文字でもよい（コンパイルでは判別しない）。サブルーチンは「proc near」で１つのプロシジャーとして定義し，「endp」でその終りを示す。

「EX91A2.ASM」と「EX91AD.INC」はサブルーチン「DSUB」を別のファイルとして作成し，「INCLUDE」ディレクティブで結合す

```
;EX91A2.ASM
_TEXT   segment byte public 'CODE'
        assume  cs:_TEXT
        public  _disp1
_disp1  proc near
        MOV  AX,0A000H
        MOV  ES,AX
        MOV  AL,30H
        MOV  BX,40
        CALL DSUB

        MOV  AL,30H
        MOV  BX,60
        CALL DSUB
        RET
_disp1  endp

INCLUDE  B:\INCLUDE\EX91AD.INC

_TEXT   ends
        end
```

```
;B:\INCLUDE\EX91AD.INC
DSUB    proc near
        MOV  CX,10
LP:     MOV  ES:[BX],AL
        INC  AL
        ADD  BX,00A0H
        LOOP LP
        RET
DSUB    endp
```

る場合のプログラムである。サブルーチン「DSUB」の部分をそのまま別ファイルとし，メインソースプログラム内の同一セグメント内で「INCLUDE」を定義する。なお，ここではインクルードされるファイル「EX91AD.INC」（拡張子は予約名以外であれば何でもよい）をドライブＢのサブディレクトリ「INCLUDE」内に作成した場合である。ＡＳＭファイルのコンパイルでは「EX91A2.ASM」だけ指定することにより，インクルードファイルを結合してコンパイルを行い，１つのオブジェクトファイル「EX91A2.OBJ」を作成する。この「INCLUDE」ディレクティブで結合する方法は，単に１つのソースプログラムを２つに分けただけであるので，このインクルードされるファイルはそれ自身で独立したソースファイルではない。すなわち，インクルードファイルだけをコンパイルすることはできない。

では，次に，サブルーチンも独立したソースファイルで作成し，「LINK」よって結合する場合のプログラムを「EX91A3.ASM」「EX91AS.ASM」に示す。この時にはサブルーチンのソースプログラムもまたＡＳＭファイル（拡張子が「.ASM」）である。メインソースファイルではサブルーチン「DSUB」が別のソースファイルで定義されていることを指示するために「EXTRN（External ディレクティブ）」宣言を行わなければならない。サブルーチン名に続く「:near」は同一セグメント内にサブルーチンを定義することを意味し，サブルーチンソースファイル「EX91AS.ASM」内の「DSUB proc near」に対応する。「EX91AS.ASM」でも，それ自身がＣ言語に対応した形式記述でなければならないので，メインソースプログラムと同様な記述となる。サブルーチン「DSUB」は別のソースファイル（ここでは「EX91A3.ASM」）で参照（呼び出される）されるので，「Public」宣言でその許可を行う。

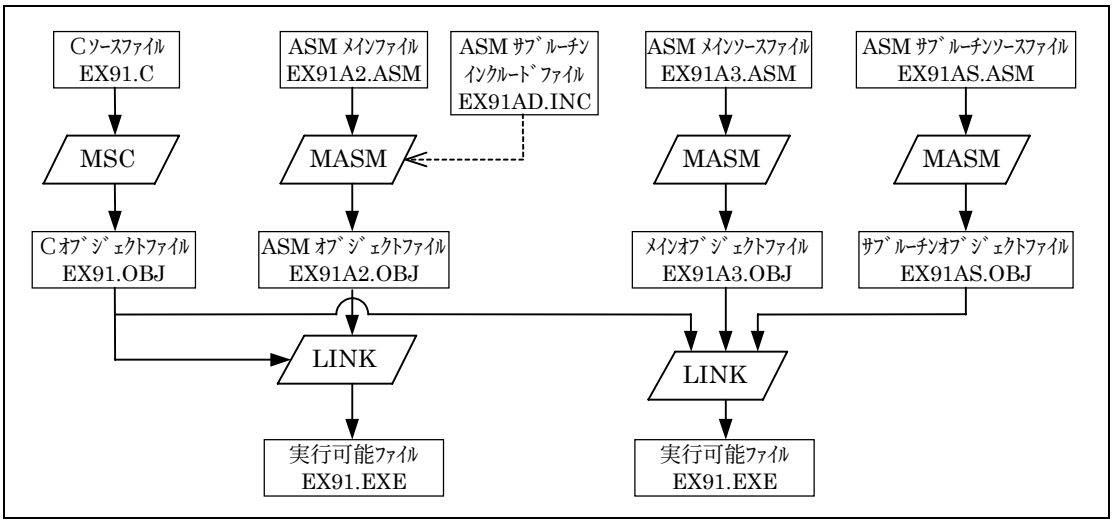
２つのプログラムはそれぞれ独立してコンパイルを行い，リンカーによって結合される。「extrn」と「public」はリンク時にこの２つのオブジェクトファイルを結合するための情報となる。

```
;EX91A3.ASM
_TEXT    segment byte public 'CODE'
         assume  cs:_TEXT
         public  _displ
         extrn DSUB:near
_displ  proc near
         MOV  AX,0A000H
         MOV  ES,AX
         MOV  AL,30H
         MOV  BX,40
         CALL DSUB

         MOV  AL,30H
         MOV  BX,60
         CALL DSUB
         RET
_displ   endp
_TEXT    ends
         end
```

```
;EX91AS.ASM
_TEXT    segment byte public 'CODE'
         assume  cs:_TEXT
         public  DSUB
DSUB     proc near
         MOV  CX,10
LP:      MOV  ES:[BX],AL
         INC  AL
         ADD  BX,00A0H
         LOOP LP
         RET
DSUB     endp
_TEXT    ends
         end
```

「INCLUDE」ディレクティブで結合する方法に比べて，独立したソースファイルを作成しリンカーで結合する方法では，それぞれを別々にコンパイルするので，プログラム記述などのエラーが発生した場合などの処置が容易となる。２つの方法におけるコンパイルとリンクの流れは下図のようである。なお，現在作成している制御プログラムは全て「INCLUDE」ディレクティブで結合する方法としている。



［マクロ定義］

マクロ定義（Macro Define）とはマクロアセンブラで用意されているディレクティブであり，１連のプログラムの記述形式を定義するものである。先の「EX91A.ASM」をマクロ定義を用いて，記述すると「EX92A.ASM」のようになる。

```
;EX92A.ASM
DSUB    MACRO D1,D2
        LOCAL LP
         MOV  AL,D1
         MOV  BX,D2
         MOV  CX,10
LP:      MOV  ES:[BX],AL
         INC  AL
         ADD  BX,00A0H
         LOOP LP
        ENDM

_TEXT    segment byte public 'CODE'
         assume  cs:_TEXT
         public  _disp1
_disp1   proc near
         MOV  AX,0A000H
         MOV  ES,AX
         DSUB 30H,40
         DSUB 30H,60
         RET
_disp1   endp
_TEXT    ends
         end
```

記述定義は

```
マクロ名    MACRO 引数,引数,引数,・・・

                ENDM
```

であり，ソースプログラムの最初のセグメント外で行う。「マクロ名」および「引数」は予約変数以外である。「引数」は数値，変数および文字であるが，文字を引き渡す場合には，例えば

```
MOV AX,OFFSET &D1
```

のように先頭に「&」を付ける。「LOCAL」はマクロ定義中で使用するラベルを定義を行う。

一方，プログラム中では

```
マクロ名    引数,引数,引数,・・・
```

と記述し，コンパイルによってそれぞれの引数がマクロ定義の変数に渡され，プログラム中に展開される。なお，引数は「，」で区切り，ブランクを入れてはいけない。

このマクロ定義は見かけ上，先のサブルーチンのようであるが，コンパイルされた結果のファイルは「EX91A.ASM」と全く同じものとなる。したがって，サブルーチン化したプログラムに比べて使用するプログラムメモリは大きくなるが，「CALL」や「RET」に伴う処理を行わないために，プログラムの実行時間は速くなる。

マクロアセンブラで使用するものとして，「EQU」ディレクティブがある。これは数値を変数で定義するものであり，次の形式でソースプログラムの最初（マクロ定義より先に）に行う。

```
名前　EQU　データ
```

ソースプログラム中では全て名前（予約名以外の英字で始まるもの）で記述できる。プログラム中で数値をそのまま記述する代わりに「EQU」ディレクティブで定義しておけば，簡単にデータを変えることができる。

マクロアセンブラではその他にも多くのディレクティブがあるが，それらについてはマニュアルを参照されたい。