5. SUBROUTINE HANDLING

5.1 Introduction

Subroutines are usually small sections of a program for performing a single function, be it initialising data items, displaying items on a visual display screen, or carrying out a modulus eleven check, for example.

Writing programs as a series of subroutines can reduce development, testing and maintenance time compared with a 'monolithic' approach. As smaller units are easier to understand, testing can be carried out on each subroutine in turn.

If a CREDIT subroutine is located in a different module to the routine that is to call it, then the calling module must contain an external directive (EXT) giving the subroutine name, and the module containing the subroutine itself must contain an entry directive (ENTRY) to match.

Example:

```
Main module                Subroutine module

EXT  SUB1                  SUB1      PROC

PERF SUB1
```

Subroutines in CREDIT are enclosed in directives starting with the procedure directive (PROC) and ending with the procedure end directive (PEND).

Example:

```
XCH       PROC
           <
           >        CREDIT statements forming
           <        the subroutine called XCH
          PEND
```

The subroutine name is located in the label field of the PROC directive; in the above example the subroutine name is XCH, and this is the name that must appear in the ENTRY and EXT directives, if performed from another module.

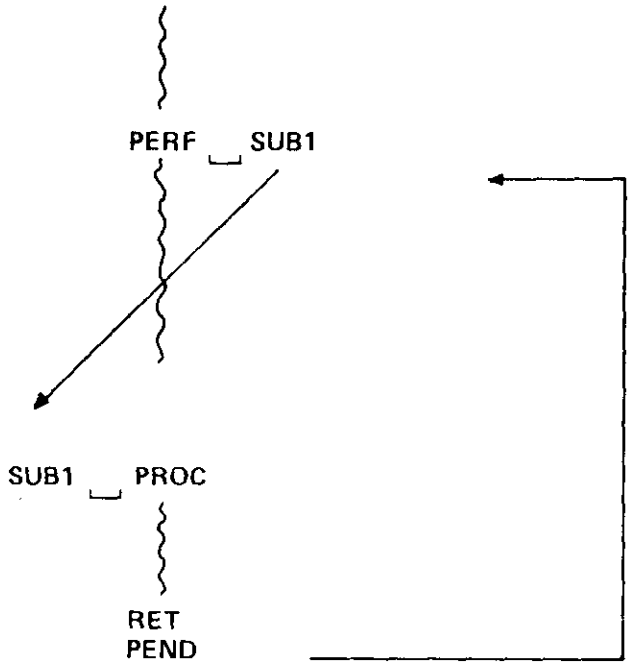
5.2. Execution of a subroutine

The transfer of control to the subroutine is achieved using the perform command (PERF) or the perform indexed command (PERFI), and the return of control to the calling routine with the return command (RET).

The PERF and PERFI command both store on the system stack the address of the next instruction to be obeyed after a normal return from the subroutine, along with other information on the task module. Each stack entry occupies six bytes, and the stack currently has a default size of 128 bytes though this may be altered by using the stack directive. If, for example, a large number of embedded performs are to be made the size may have to be increased, and if no embedded calls are to be made the stack size may be reduced.

REFERENCE	PAGE IN M04
ENTRY	1.2.7
EXT	1.2.9
PERF	1.4.134
PERFI	1.4.135
PEND	1.2.17
PROC	1.2.21
RET	1.4.139

CREDIT SUBROUTINE IN THE SAME MODULE



The perform command has the following format:-

```
PERF    subname{,p1{...,pn}}
```

where - subname is the name of the subroutine to be executed.
- p1 to pn are actual parameters which will be passed to the subroutine.

The indexed perform command has the following format:-

```
PERFI   index,s1{,s2...,sn}
```

where - index is a binary index to be used for selecting the subroutine that is to be executed
- s1,s2...,sn is a list of subroutine names, the first entry being regarded as entry one, so if the variable being used for the index has a value 2 then the 2nd subroutine will be activated.

The return command has the following format:-

```
RET     <opt. byte dis.>
```

The optional byte displacement specifies the number of bytes which are to be added to the return address before the return command is executed.

The RET instruction transfers control back to the calling routine at the instruction after the PERF command or PLIST directive (see below), and must therefore be the last logical instruction in a subroutine.

The address to which the return will be made is held on the system stack; it is the byte displacement from the perform instruction. The perform instruction may vary in length depending on the number of parameters and the type of addressing adopted; when the RET instruction is encountered control is returned to that instruction. It is possible to add a displacement to the return command to enable a return to a subsequent instruction, as described above.

Example:

```
PERF    ABC,A,B
B       LI
ADD     A,B
RET
```

On encountering the above return instruction, control will be returned to the branch instruction, which will transfer control to the instruction at statement identifier LI. However as the branch instruction occupies two bytes; if the subroutine ABC is terminated by the instruction:-

```
RET     2
```

then control will be returned to the ADD instruction, as the branch instruction occupies two bytes.

REFERENCE	PAGE IN M04
PERF	1.4.134
PERFI	1.4.135
RET	1.4.139

5.2 Parameter handling

5.2.1 General rules

A subroutine can access directly any data item defined in the data division which is available to the calling routine. In addition a subroutine may have formal parameters, which are local names listed in the operand section of the PROC directive and used solely within that subroutine. When the subroutine is executed then the actual parameters will be substituted for these formal parameters. The actual parameters are variable names used within the calling routine. The actual parameters are specified in the PERF command after the subroutine name, and if the PERFI command is used then they are specified in the PLIST directive immediately after the PERFI command. There can be up to eight formal parameters in a subroutine, depending on the contents of the LITADR option in the directive OPTNS.

The parameters in the calling program are called the "actual parameters" and those in the subroutine the "formal parameters". The valid types for actual parameters are listed below.

boolean (BOOL) data items
binary (BIN) data items
binary arrays (BINI)
binary coded decimal data items (BCD)
binary coded decimal arrays (BCDI)
string data items (STRG)
string arrays (STRGI)
data set identifiers
literals, but not those with an unspecified type or of type X

The indexed perform may pass parameters to a subroutine, in which case all the subroutines in the subroutine list will require the same number of parameters. The parameters to be transferred are specified in the parameter list directive (PLIST) located immediately after the PERFI command. The format of the PLIST directive is :-

```
        PLIST    p1{,...pn}
```

p1{,...pn} is the list of actual parameters which will be passed accross to the subroutine.

Example:

```
        PERFI    SUB1,SUB2,SUB3,SUB4
        PLIST    ACCNO,NAME,=D'1',BINW4
```

Note:-

If an array element is being passed to a subroutine, then the array and the subscript must be passed as separate parameters. In addition, the formal parameter for the array itself must be followed by open and close parenthesis, to indicate that it is an array.

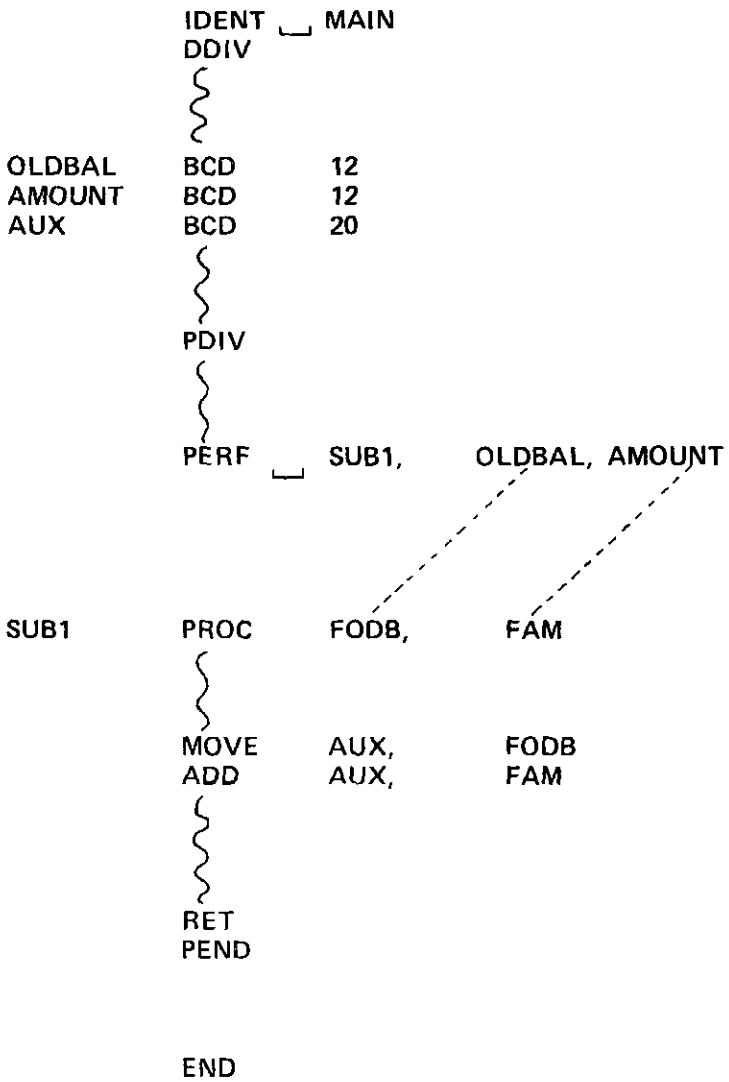
Example:

```
        PERF     SUB2,ARRAY,INDEX,AMOUNT

SUB2    PROC     FARR(),FINDX,FAM
```

DIRECTIVE	PAGE IN M04
PLIST	1.4.210

ACTUAL/FORMAL PARAMETERS



```
            SUBROUTINE  (FORMAL PARAMETERS, ARE
             (                KEY TABLE, FORMAT LIST, LITERAL)
             )
             )
             )
             )
            PERF        SUB1, LENGTH, FORM1, KTAB1, =D'1'
             (
             )
             )
             (
SUB1        PROC        INLEN, $FORM, $KTB, $LIT
             )
             (
            ADD         TRANR, $LIT
             )
             (
            K1          DSKB, INBUF, $KTB, INLEN, INDEX
             )
             (
            EDWRT       DSVOU, $FORM
             (
            PEND
```

5.2.2 Literals, keytables and format lists as parameters.

There are two ways of passing literals, keytables and format lists to subroutines, as follows:

Method 1

The actual parameter is specified in the normal way, and the formal parameter is given a name which starts with a $ (dollar sign). This tells the translator that the actual parameter to be substituted at execution time is one of the three types defined above, e.g:

```
        PERF    SUB1,LENGTH,FORM1,KTAB1,=D`1'

SUB1    PROC    INLEN,$FRM,$KTB,$LIT1
         |
        ADD     INLEN,$LIT1
         |
        RET
        PEND
```

Method 2

The formal parameter is not given a name starting with a dollar sign. In this case, the directives shown below must appear immediately after the PROC statement at the start of the subroutine.

PFRMT when using format lists
PLIT when using literals
PKTAB when using keytables

These are always required when ADRMOD is set to 2 in the OPTNS directive, and the formal parameter must not then be preceded by a $ sign, e.g:

```
        PERF  SUB1,LENGTH,FORM1,KTAB1,=D`1'

SUB1    PROC  INLEN,FRM,KTB,LIT
        PFRMT FRM
        PKTAB KTB
        PLIT  LIT
         |
        ADD   INLEN,LIT
         |
        RET
        PEND
```

A summary of rules for passing format lists, keytables and literals is shown on the next page.

DIRECTIVE	PAGE IN MO4
PFRMT	1.2.18
PKTAB	1.2.19
PLIT	1.2.20

Summary of rule for passing literals, keytables and format lists

name	PROC <opt>	FORM1 FORM1	(ADRMOD=2)	$ not required
name	PROC	$FORM1	(ADRMOD=1)	$ required
name	PROC <opt>	FORM1 FORM1	(ADRMOD=1)	$ not required

name is the name of the subroutine
<opt> is either PFRMT, PLIT or PKTAB.