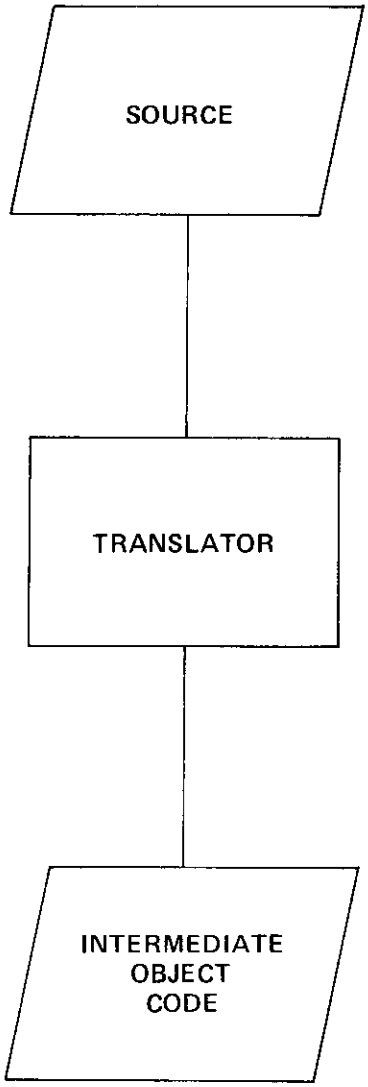
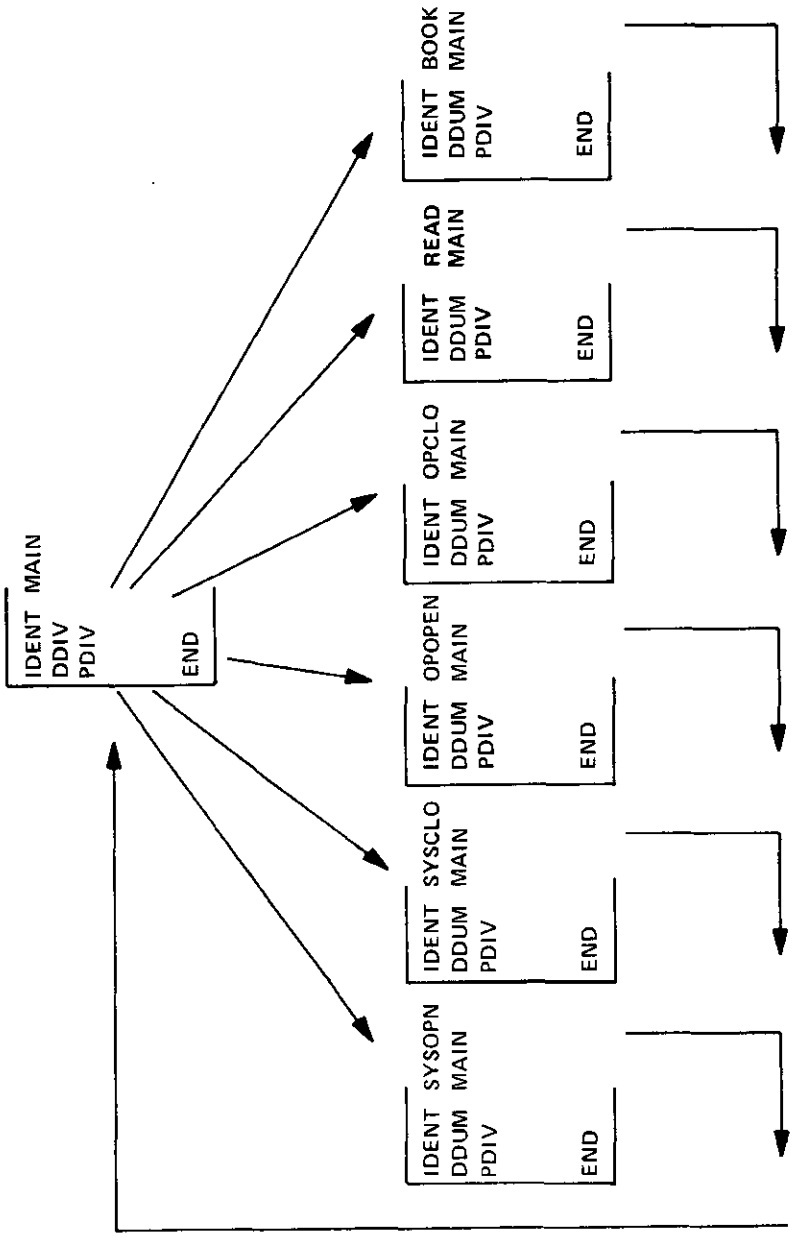
3. **CREDIT** PROGRAM STRUCTURE

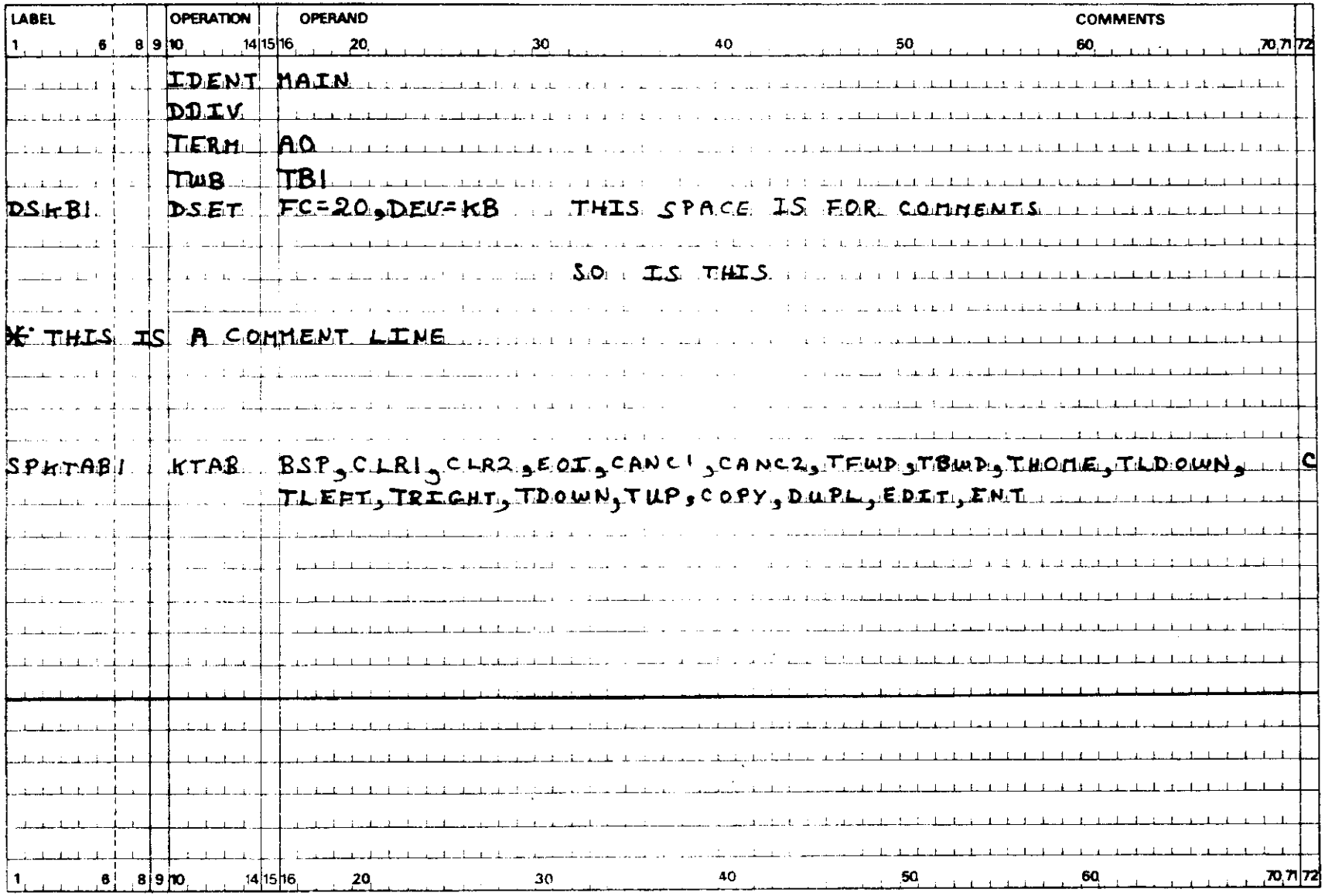
A CREDIT program consists of a number of statements. A statement may be a directive, declaration or instruction. Directives describe the module framework to the CREDIT translator, the translator is a special program which converts the application program into a form the machine can use, this output being called 'object code'. Declarations specify the type, length and use of all the variables, constants and tables used in the application and must always be located in the main module. Instructions direct the input, processing and output of information. They specify the actions to be carried out by the computer, and direct the sequence of events.


SOURCE
TRANSLATOR
INTERMEDIATE
OBJECT
CODE



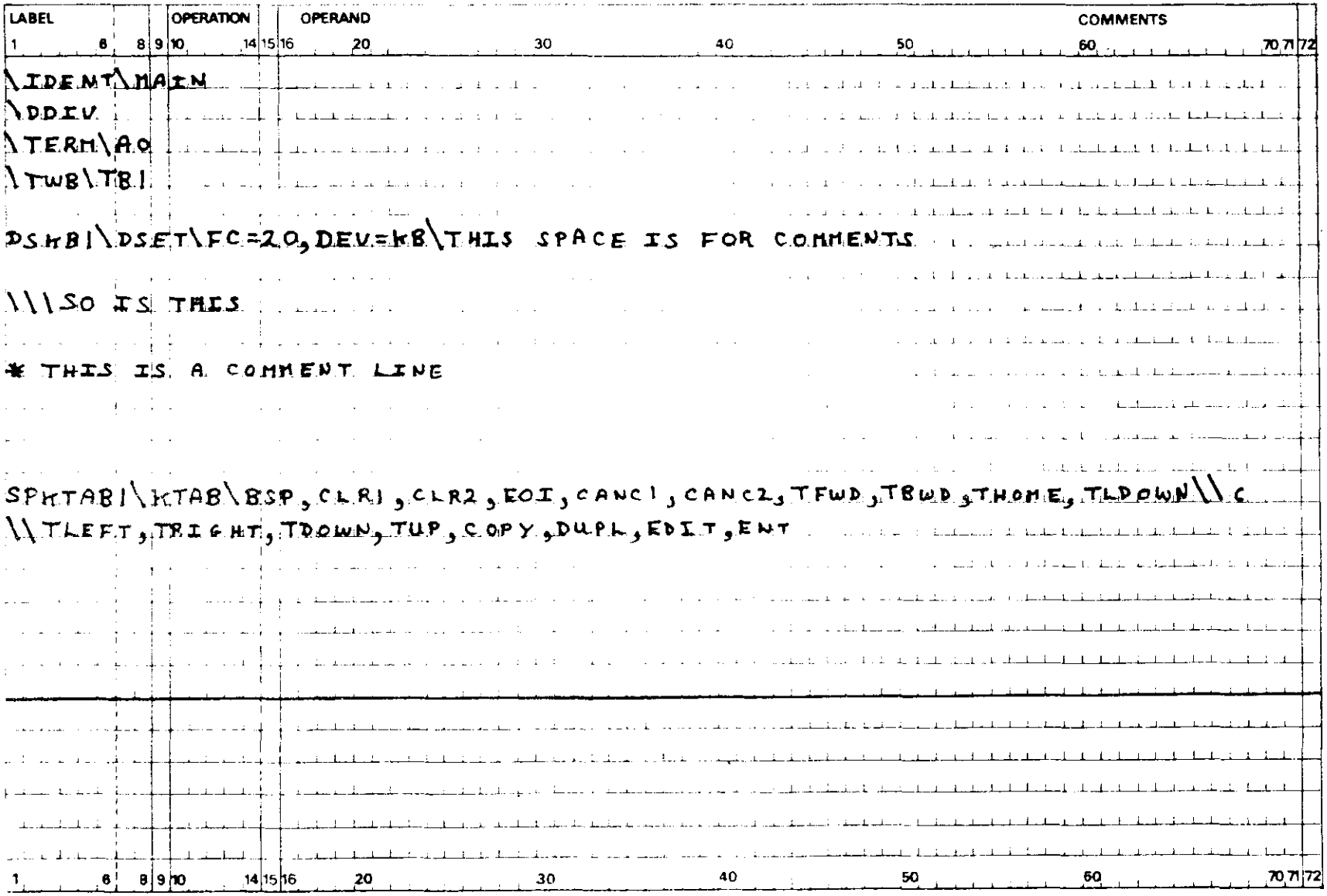
MODULAR STRUCTURE WITH A GLOBAL DATA DIVISION

The program is written as one or more modules; with the PTS system, as with many others, it is advantageous to have one module performing one specific activity, as this leads to more efficient design, writing, testing and subsequent maintenance of the application.


LABEL
OPERATION
OPERAND
COMMENTS
1 6 8 9 10 14 15 16 20 30 40 50 60 70 71 72
IDENT MAIN
DDIV
TERM A0
TWB TB1
DSKB1 DSET FC=20,DEV=KB THIS SPACE IS FOR COMMENTS
SO IS THIS
* THIS IS A COMMENT LINE
SPKTAB1 KTAB BSP,CLR1,CLR2,EOI,CANC1,CANC2,TFWD,TBWD,THOME,TLDOWN, C
TLEFT,TRIGHT,TDOWN,TUP,COPY,DUPL,EDIT,ENT

A CREDIT source program can be read into the PTS system using one of the following source input devices; cassettes or console typewriter for free format input, and cards for fixed format input. Regardless of the input device used, the source program must have follow the rules described below and shown on the attached coding sheet of examples.

A source line input to the translator is treated as an 80 character card image; if a free format input device is used then each record can only contain one source statement. Records longer then eighty characters will be truncated, while any shorter will be filled with spaces to pad them to eighty characters.

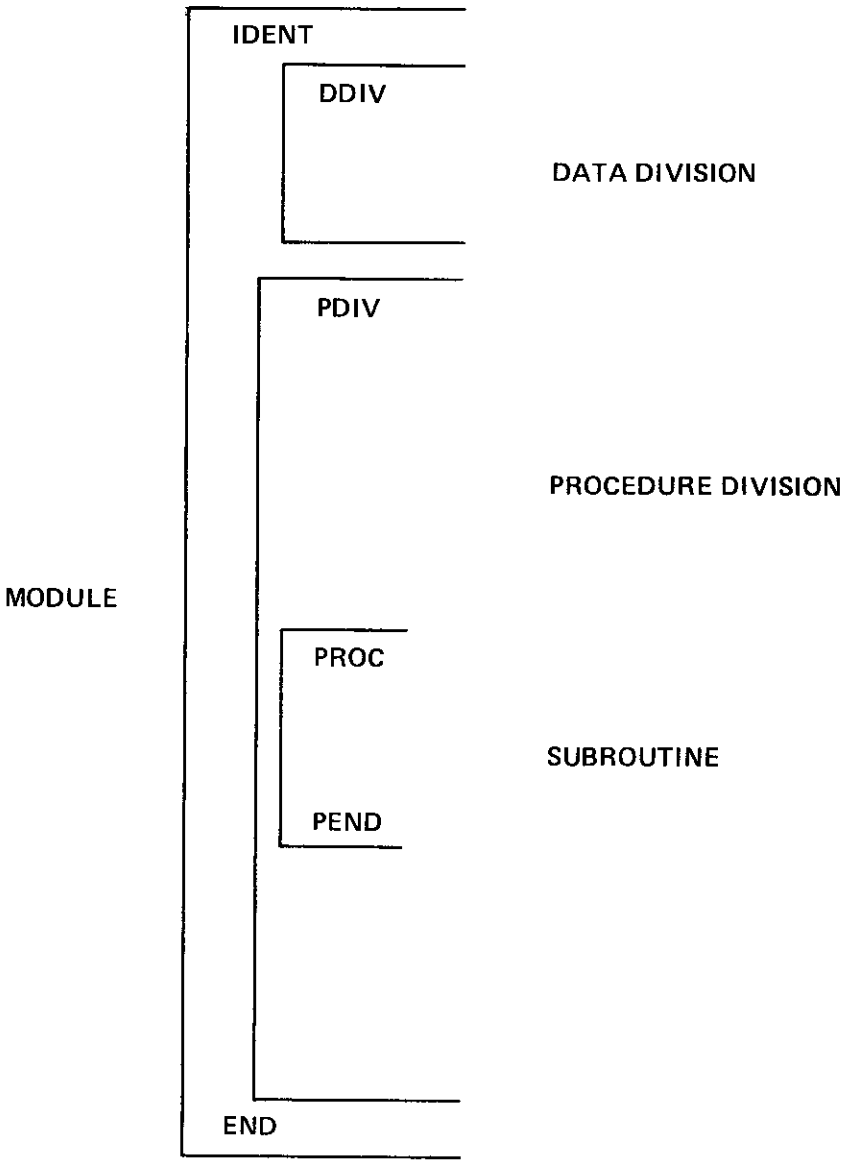

LABEL
OPERATION
OPERAND
COMMENTS
1 6 8 9 10 14 15 16 20 30 40 50 60 70 71 72
\IDENT\MAIN
\DDIV
\TERM\A0
\TWB\TB1
PSMB1\DSET\FC=20,DEV=KB\THIS SPACE IS FOR COMMENTS
\\\SO IS THIS
* THIS IS A COMMENT LINE
SPMTAB1\MTAB\BSP,CLR1,CLR2,EOI,CANC1,CANC2,TFWD,TBWD,THOME,TLDOWN\\C
\\TLEFT,TRIGHT,TDOWN,TUP,COPY,DUPL,EDIT,ENT

The source line is divided into four fields or zones:- label field, operation field, operand field and comment field. The label, operation and operand field are separated by a tabulation character (\) or at least one space. The label field begins in column one. The operand field can extend to column 71. If the operation and operand fields are blank to column 30 the rest of the record is treated as a comment. Columns 73 through 80 are ignored by the translator. If an asterisk is present in column one then the entire record will be treated as a comment. Continuation of a line is denoted by a 'C' in column 72 if card input is being used, or in the case of free format input by (\\C) two tab symbols followed by 'C', in the continuation lines the label and operation field must be left blank.

3.1 DIRECTIVES

Directives enable the application programmer to pass information to the CREDIT translator. The directives do not occupy any `core' at run time. There are six categories of directives:-

- Structure
- Linkage
- Listing
- Equate
- Parameter
- Options



FRAMEWORK OF A CREDIT MODULE (WITH DATA DIVISION)

3.1.1 Structure directives

The framework of a CREDIT module is formed from the directives IDENT, DDIV, PDIV, PROC, PEND, INCLUDE and END. An example of their use is given below.

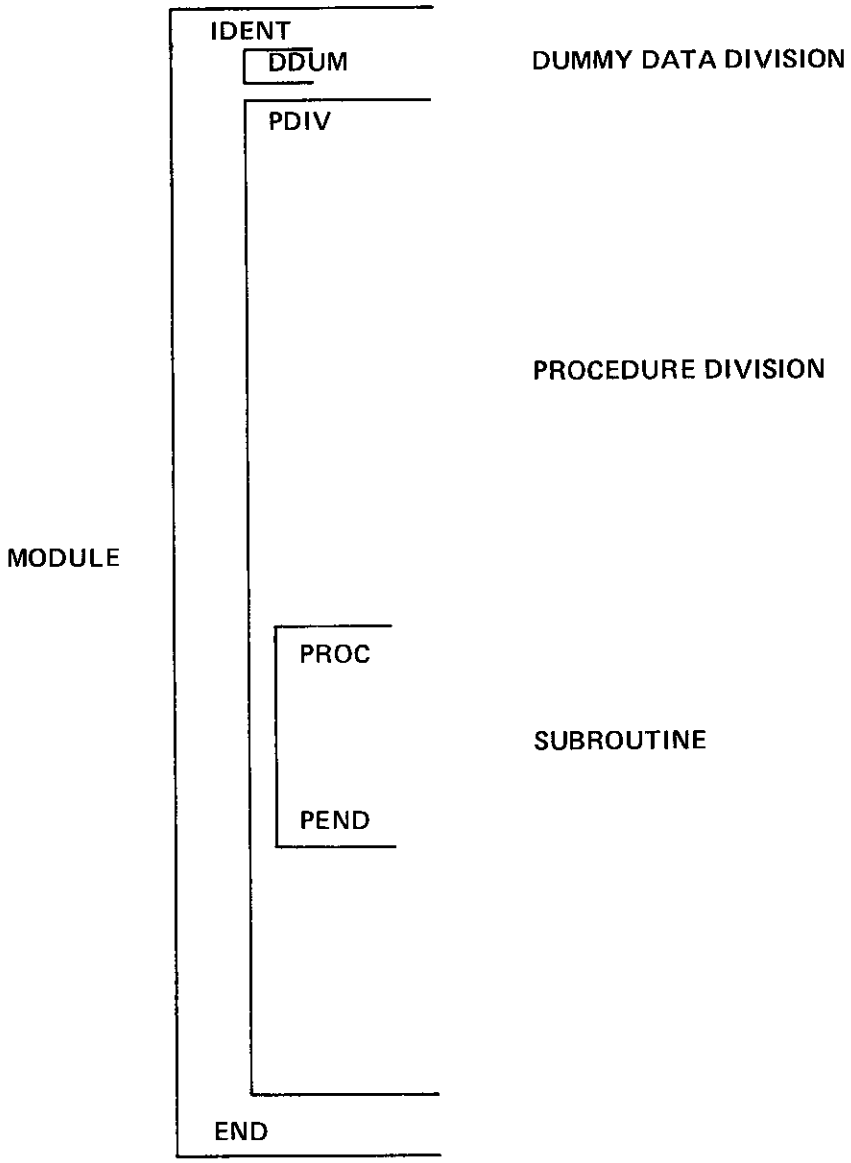
IDENT	Must be the first statement
DDIV(or DDUM)	Start of the data division

The data division contains declarations which define the type, length and value of data items used as operands in the program, together with declarations which define the interface between the applications program and the PTS System.

PDIV	Start of the procedure division

The procedure division contains the instructions which direct the input, processing and output of data. It also contains some declarations which must be used in conjunction with certain instructions.

PROC	Start of subroutine instructions
PEND	End of subroutine instructions Several subroutines may exist in one module.
INCLUDE	The contents of a source module are included in this module at this point
END	Must be the last statement



FRAMEWORK OF A CREDIT MODULE (WITH DUMMY DATA DIVISION)

3.1.1.1 Main program structure directives

The IDENT and END directives define the start and end of a module, and must be the first and last statements respectively of a module. The DDIV directive defines the start of the data division, and must be the second statement of the module containing the data division. The dummy data division directive (DDUM) is used in all other modules in place of the DDIV directive, and it refers to the IDENT of the module containing the required DDIV. The PDIV directive defines the start of the procedure division.

3.1.1.2 Subroutine structure directives

The PROC and PEND directives define the start and end of each subroutine. The IDENT, DDIV (or DDUM), PDIV and END directives can only appear once in each module; the PROC and PEND directives must be repeated for each subroutine present. However one subroutine can not be physically embedded within another, that is two or more PROC directives can not occur without an intervening PEND.

Example:

VALID

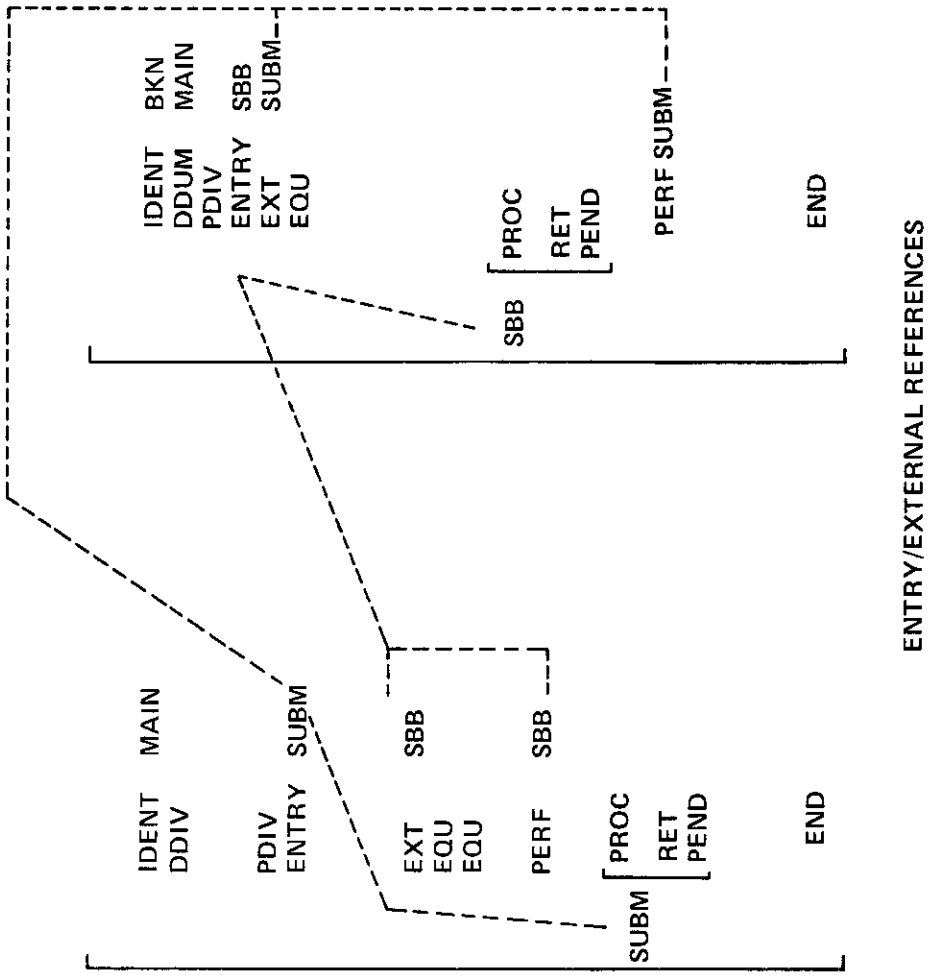
```
S1      PROC

        PEND
S2      PROC

        PEND
```

INVALID

```
S1      PROC

S2      PROC

        PEND

        PEND
```

The table below gives the structure directives and the page on which they are described in M04.

DIRECTIVE	PAGE IN M04
DDIV	1.2.3
DDUM	1.2.4
END	1.2.6
IDENT	1.2.10
INCLUDE	1.2.11
PDIV	1.2.17
PEND	1.2.17
PROC	1.2.21


IDENT BKN
DDUM MAIN
PDIV SBB
ENTRY SUBM
EXT
EQU
PERF SUBM
SBB
PROC
RET
PEND
END
IDENT MAIN
DDIV
PDIV SUBM
ENTRY
EXT SBB
EQU
EQU
PERF SBB
SUBM
PROC
RET
PEND
END
ENTRY/EXTERNAL REFERENCES

3.1.2 Linkage directives

Linkage to external modules

CREDIT modules which have to be linked into an application program may contain references to statements or subroutines in other modules. In order to achieve the correct linkages, entry points in this module and external references to other modules must be specified. The ENTRY and EXT directives are used for this purpose. They must be written in the procedure division.

Thus, in order for the references to be correctly handled, a module referring to a statement-identifier in another module must contain the EXT directive to specify that the reference is not in this module, and this must be paired with an ENTRY directive in the other module.

Start points

There must be at least one START directive for the entire application. When the system is started (i.e. the TOSS Monitor is loaded and the application program begins execution) tasks are activated as specified in the configuration data to be studied later. The tasks are activated at the start points specified in the START directives of the relevent terminal classes. The START directive(s) must be written in the data division and must be specified as entry points (ENTRY) in the procedure division (PDIV).

If more than one START directive appears in a terminal class, only the first start point will be activated when the system is started; the other points will be held pending and will be activated only after the first task has executed an EXIT instruction.

Error control with memory management, swappable work blocks or overlay

If memory management is being used and a REENTER point has been defined for handling disk errors, then this must also be declared as an ENTRY point.

The table below gives the linkage directives and the pages on which they are described in M04.

DIRECTIVE	PAGE IN M04
ENTRY	1.2.7
EXT	1.2.9
REENTER	1.2.22
START	1.2.23

3.1.3 Listing directives

These directives are used to control the printing of the application listing at translation time. The available directives are:-

- LIST
- NLIST
- EJECT

EJECT when encountered issues a form feed to the printer, NLIST stops the production of the program listing, LIST causes the listing to recommence.

The table below gives the listing directives and the pages on which they are described in MO4.

DIRECTIVE	PAGE IN MO4
LIST	1.2.7
NLIST	1.2.9
EJECT	1.2.22

3.1.4 Options directive

This directive, if required, must be located immediately after the DDIV or DDUM directive and controls the following items:-

- Lines per page for translator listings
- One or two byte addressing for data items
- One or two byte addressing for literal constants
- One or two byte addressing for keytables
- One or two byte addressing for pictures
- One or two byte addressing for format lists
- One or two byte addressing data sets
- Number of entries in work blocks

A one byte addressing system allows only 16 entries per work block whereas a two byte system would permit up to 255 entries. With one byte addressing up to 255 literal constants, keytables, pictures and format lists are allowed; if two byte addressing were used then there could be up to 32767.

The lines directive is overruled if the number of lines per page is given when entering the program development sytem (DOS-PTS).

The various options can be written in any order.

The format of this directive is:-

```
OPTNS     {LINES=decimal number,}{LITADR=decimal number,}{ADRMOD=decimal
                                                          number}
```

The LITADR option is followed by a four digit decimal number composed of one's or two's, a one representing one byte addressing a two, two byte addressing. The first digit of the number is for literal constants, the second keytables, the third pictures and the fouth format lists.

The address mode option (ADRMOD) is used to specify one or two byte addressing for data items, literal constants, data sets, keytables, format lists and pictures; the valid forms of ADRMOD are shown below:-

ADRMOD=1 one byte addressing is to be used (default).

ADRMOD=2 two byte addressing is to be used; LITADR will be set by the translator to 2222.

Example 1

```
OPTNS     LINES=72,LITADR=2212
```

In example 1 there will be seventy two lines per page on the program listings, and two byte addressing will be used for literal constants, keytables and format lists; but a one byte addressing system is to be used for pictures.

Example 2

```
    OPTNS    ADRMOD=2
```

In example 2 two byte addressing is to be used, permitting more workblock entries (data items), two byte addressing will also be used for the literals. Literals, keytables, format lists, pictures and data sets are described in sections 3.2.8, 6.2.3, 6.5 and 3.2.6 respectively.

3.1.5 Equate directive

The EQU directive is used to set up constants with mnemonic names; the prorammer uses the name when writing the instructions, and when the program is translated the constant is substituted for the name. The EQU directives can be located anywhere in the procedure division after the ENTRY and EXT directives. The maximum value that can be held in an equate directive is 255.

The directive format is:-

mnemonic-name EQU value-expression

eg

```
    BSP    EQU    X'09'          BSP to be replaced ty Hex. value 9
    NDBS   EQU    BSP            NDBS to be replaced by contents of BSP
    CHA    EQU    X'40'          CHA to be replaced ty Hex. value 40
    NGP    EQU    CHA+1          NGP to be replaced ty Hex. value 41
```

DIRECTIVE	PAGE IN M04
EQU	1.2.8

3.1.6 Parameter directives

These are special directives for use when passing format lists, keytables, literals and parameters to subroutines. There are four directives used within the subroutines:-

- PFRMT for format lists
- PLIT for literals
- PKTAB for keytables.
- PLIST for subroutine parameters

They are required when ADRMOD is set to two, or if the formal parameter is not preceeded by a $ sign, and are located after the directive PROC, see below:-

```
SUBF    PROC    FORM1       (ADRMOD=2)
        <opt>   FORM1

SUBF    PROC    $FORM1      (ADRMOD=1)

SUBF    PROC    FORM1       (ADRMOD=1)
        <opt>   FORM1
```

<opt> is either PFRMT, PLIT or PKTAB.

PLIST Actual parameter list

This is used to pass parameters to subroutines that have been activated using the indexed perform (PERFI) instruction.

The handling of subroutines, and the syntax of parameter passing is described in detail in section 5.

DIRECTIVE	PAGE IN MO4
PFRMT	1.2.18
PLIT	1.2.20
PKTAB	1.2.19
PLIST	1.4.210

3.2 Data division

3.2.1 Overview

The DDIV contains declarations which define the type, length and value of data items used by the program, together with those declarations which define the interface between the application and TOSS Monitor. The basic layout is shown below, and will be described in detail later.

```
          DDIV                    Directive for start of data division
          TERM      TO            Terminal class identifier
          TWB       TB1           Terminal work block TB1
DSKB1     DSET      FC=20,DEV=KB  Definition for keyboard input
          START     S1            First entry point for this terminal class
          START     S2            Next entry point for this terminal class
.....................................
Work block declarations are located
in this section
.....................................

          PDIV                    Start of procedure division
          ENTRY     S1            Entry point S1 is in this module
          ENTRY     S2            Entry point S2 is also in this module
          EXT       SCREEN        Externally held subroutine
KEY       EQU       D'56'
```

Explanation of the above example

i — DDIV - this indicates the start of the data division.

ii — TERM TO - the terminal class identifier; TO is the name of the terminal class, as described in section 3.2.3.

iii TWB — TB1 - TB1 is to be used as a terminal work block for this terminal class. A description of work blocks is given in section 3.2.5.

iv DSKB1 — DSET FC=20,DEV=KB - assigns a dataset device to be used by this terminal class, which will be referred to in the program as DSKB1. It has a file code of 20 and is a keyboard device. The DSET command is described in detail in section 3.2.6.

START - This gives the program entry point where execution will commence. The directive is described in section 3.2.4.

DATA DIVISION (1)

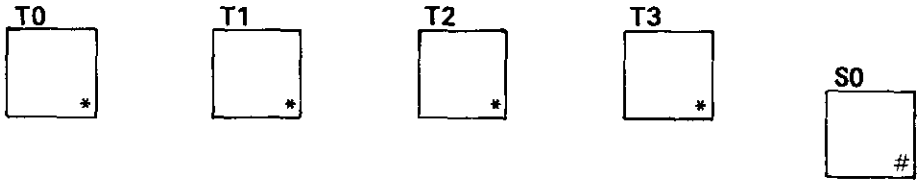
```
                IDENT   MAIN
                DDIV
                TERM    T0
                CWB     CB1
                CWB     CB2
                TWB     TB1
                TWB     TB2
                START   G0
DSKB            DSET    FC=20,DEV=KB
DSJT            DSET    FC=34,DEV=JT,BUFL=80
DSVO            DSET    FC=30,DEV=TP,BUFL=80

CB1             BLK
RDLA            BIN     X'0080'
ADDLA           BIN     X'0001'
INDX            BIN
DATE            BCD     10D'0'
WKSTR           STRG    5

CB2             BLK

TB1             BLK

TB2             BLK

                PDIV
```

```
vi    TB1       BLK - this is the start of the declarations which form work
                      block TB1, work block definitions must be located at
                      the end of the terminal class definitions

iii             PDIV - start of the procedure division

viii            ENTRY S1 - the entry point S1 will be located in this module

ix              EXT  SCREEN - this is reference to an externally held routine

x     KEY       EQU  D'56' - a constant is set up with the decimal value
                             56. The maximum value for a constant is 255.
```

3.2.2 Structure of the data division

The data division is divided into two sections, the first contains the terminal class definitions and the second defines the data items that make up the work blocks. An example of a data division is given below, with two terminal classes; note that the terminal classes must all be defined before the work blocks.

```
            DDIV

            TERM      T0              Terminal class identifier
            TWB       TB1             Terminal work block TB1
            CWB       CX1             Common work block
            CWB       CX2             Common  work block
DSKB1       DSET      FC=20,DEV=KB    Definition for keyboard input
DSDY1       DSET      FC=50,DEV=DY,BUFL=120         vdu display
            START     S1              Start point for this terminal class

            TERM      S0              Terminal class identifier
            TWB       TB1             Terminal work block TB1
            CWB       CX1             Common work block
DSKB1       DSET      FC=20,DEV=KB    Definition for keyboard input
DSPRT       DSET      FC=40,DEV=LP,BUFL=240         line printer
            START     S2              Start point for this terminal class
            STACK     128             Stack size for this class

.....................................
Work block declarations are located
in this section
.....................................

          PDIV
```




```
# * MAIN
# * INPUT
# * PROCESS
# * OUTPUT
  * CURRENT
    ACCOUNT
  * SAVINGS
    ACCOUNT
# CURRENCY
  EXCHANGE
  * OPEN
    ACCOUNT
  * CLOSE
    ACCOUNT
# CHEQUES
```

* MODULES (PROGRAMS)
USED BY TERMINAL CLASS T0

\# MODULES (PROGRAMS)
USED BY TERMINAL CLASS S0

TERM	T0
TWB	TB1
TWB	TB2
TWB	TB3
CWB	CB1

TERM	S0
TWB	TB4
TWB	TB5
TWB	TB6
CWB	CB1

TERMINAL CLASS

3.2.3 Terminal class declaration

The TERM declaration identifies the terminal class with a unique two character identifier. This declaration is followed by the relevent work blocks, start points, data set identifiers etc. for the terminal class. A terminal class is defined as a collection of work stations performing similar functions.

Each terminal class has its own specified work blocks, input/output devices, entry and reentry points stack; a terminal class may consist of several work stations. Each work station or task forming the terminal class will have its own copy of the above mentioned items. For example terminal class S0 has a specified stack size of 128 bytes, so each task forming that class will have its own stack 128 bytes in size. The STACK declaration is described in M04.

Each work station in a terminal class is identified with a 'task identifier', which is specified at system configuration time. The first task in terminal class T0 will have a task identifier of T0, the second will have a task identifier of T1, the nth task wil have an identifier of Tn-1.

In the example on page 3.2.6 the task T0 has been configured with four copies, the tasks forming this class have the identifiers T0, T1, T2 and T3.

DECLARATION	PAGE IN M04
STACK	1.3.22
TERM	1.2.26

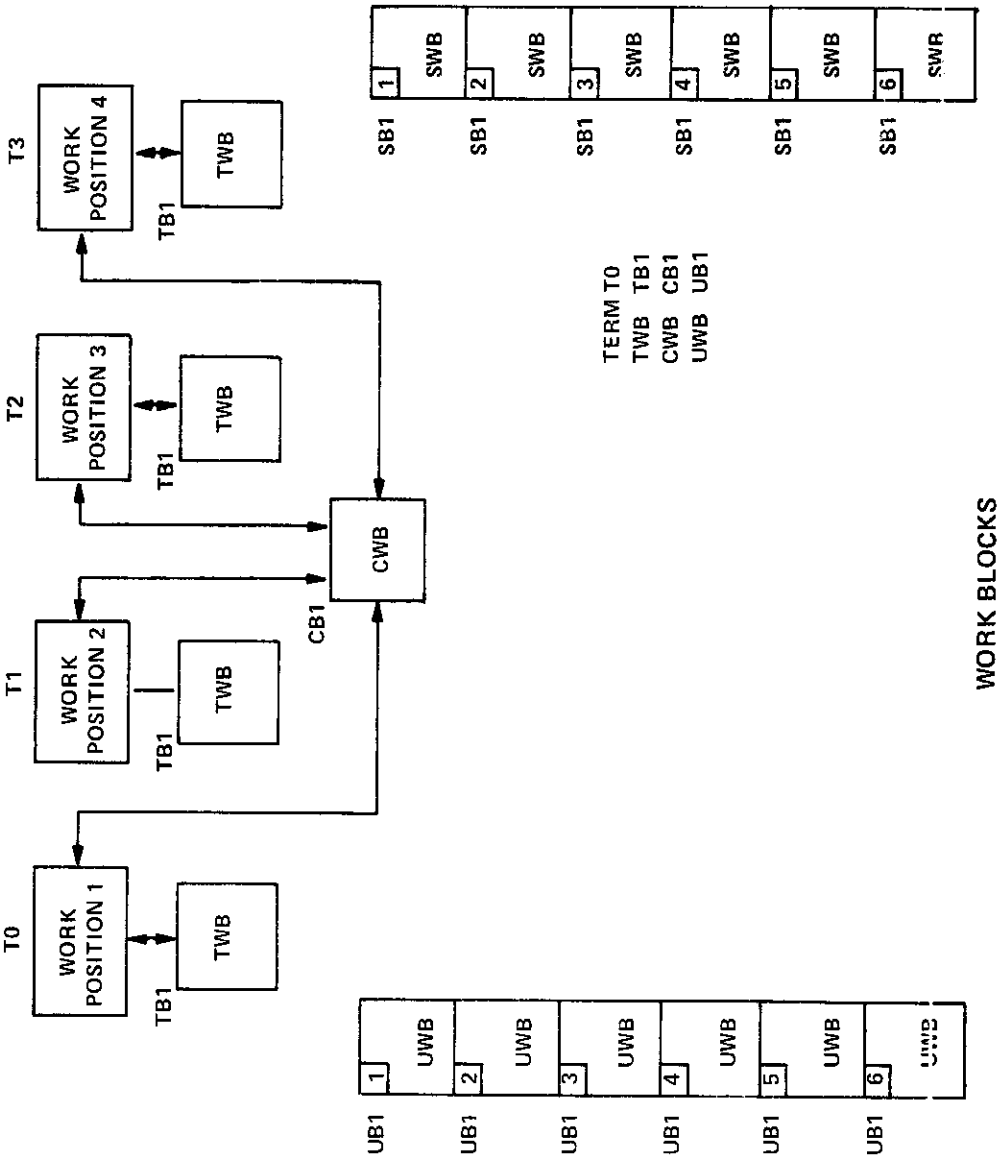
3.2.4 Start directive

The start directive gives the program entry point where program execution will commence for this task. There must be at least one start point in a program.

If a terminal class contains multiple start points the first will be used at the start, subsequent start points only being used when an EXIT is encountered. In the example shown in section 3.2.2, execution will commence at S1 but when an EXIT is encountered execution wil pass to entry point S2.

If a terminal class does not contain a START directive then it can only be invoked by another terminal class with the activate (ACTV) instruction

DECLARATION	PAGE IN M04
START	1.2.23


T3
WORK POSITION 4
TWB
TB1
T2
WORK POSITION 3
TWB
TB1
T1
WORK POSITION 2
TWB
TB1
T0
WORK POSITION 1
TWB
TB1
CWB
CB1
SWB
SB1
UWB
UB1
TERM T0
TWB TB1
CWB CB1
UWB UB1
WORK BLOCKS

3.2.5 Workblocks

These are used by the programmer to provide areas of store which can be used for input/output buffers and work locations. There are five types of work block:

- Terminal workblocks (TWB)
- Common workblocks (CWB)
- User workblocks (UWB)
- Dummy workblocks (DWB)
- Swappable workblocks (SWB)

There can be a maximum of fifteen work blocks in a terminal class and 16 non-boolean entries in a block, though this can be increased to 256 non-boolean entries in a block if the option ADRMOD=2 is specified in OPTNS directive, as described in section 3.1.4. Each workblock can contain up to sixteen boolean items, as one word is reserved for these data items per block. Non-boolean items occupy one entry in the workblock, except arrays which take up <u>two</u> entries in a work block, <u>unless</u> they are the last item in a work block when they occupy only one entry.

Valid 16 entries

```
TB1      BLK
I1       BOOL
NBIN1    BIN
NBIN2    BIN
ABIN3    BINI     (4)
ABCD4    BCD      5D'0'
ABCD5    BCDI     (12),5D'0'
TBCDX    BCDI     (8),12D'0'
ACCX     BCDI     (99),8D'0'
BRANCH   STRGI    (40),10C
MNGR     STRGI    (40,2),25C
TELNO    STRGI    (40,3),10C
```

Invalid 17 entries

```
TB2      BLK
I1       BOOL
NBIN1    BIN
NBIN2    BIN
ABIN3    BINI     (4)
TELNO    STRGI    (40,3),10C
ABCD5    BCDI     (12),5D'0'
TBCDX    BCDI     (8),12D'0'
ACCX     BCDI     (99),8D'0'
BRANCH   STRGI    (40),10C
MNGR     STRGI    (40,2),25C
ABCD4    BCD      5D'0'
```

The declaration for a terminal block consists of the mnemonic for the block type in the operator field and the name of the block in the operand field, eg:-

```
         TWB      TB1
```

Associated with each work block declaration will be a work block description where each data item forming part of that work block is described eg:-

```
TB1      BLK
CASID    STRG     6C'*'
DEP      BCD      12D'0'
WHDL     BCD      12D'0'
```

3.2.5.1 Terminal workblocks

A terminal class can contain one or more terminal workblock (TWB) definitions, each task within the terminal class having a separate copy of the work blocks. Each terminal work block will contain the data items to be used by that terminal class; the list of terminal work blocks to be used will be located after the terminal identifier (TERM), the work blocks being located after the terminal definitions.

For example:-

```
            TERM     A0
            TWB      TB1
            TWB      TB2
            TERM     B0
            TWB      TB1

  TB1       BLK
  NAME      STRG      20
  ADDR      STRG      30
  ACNO      BCD       12
  OVFT      BCD       8
  BAL       BCD       8
  TRAN      STRG      3
  FLAG      BCD       2D

  TB2       BLK
  FL        BOOL
  SPPROMPT  BOOL
  SPCHANGE  BOOL
  SPERCALL  BOOL
  SPBINW1   BIN
  SPBINW2   BIN
  SPBINW3   BIN
  SPBINW4   BIN
  SPINPUT   STRG      80C' '
  SPSTRGW1  STRG      2C' '
```

In the above example the data items making up terminal work block TB1 will be available to terminal classes A0 and B0, but TB2 is only available to terminal class A0. Each task making up the terminal classes will have a separate copy of the data items, and so it is not possible to use terminal workblocks to pass information to, or receive information from other tasks.

IDENTIFIER	PAGE IN M04
TWB	1.3.4

DATA DIVISION (2)

```
            IDENT    MAIN
            DDIV
            TERM     TØ
            CWB      CB1
            TWB      TB1
            START    GØ
DSKB        DSET     FC=20,DEV=KB

            TERM     SØ
            CWB      CB1
            TWB      TB2
            START    SØ GØ
DSKB        DSET     FC=20,DEV=KB

CB1         BLK
            {
RDLA        BIN      X'0080'
            {
TB1         BLK
            {
TB2         BLK
            {
            PDIV

            ENTRY GO
```

ILLEGAL

```
TERM    SØ
TWB     TB2
CWB     CB1
START   SØ GØ
```

3.2.5.2 Common workblocks

These are used to hold information required by more than one task, for example the current date and time, and may be used for passing information from one task to another; one task may write to a data item in a common workblock, and another task may subsequently access that data item. A common workblock (CWB) definition can be present in one or more terminal classes, and all the tasks in the terminal classes containing that common workblock declaration are able to access the information held in that common workblock.
For example:-

```
TERM    A0
TWB     TB1
CWB     CX2
CWB     CX1
TERM    B0
TWB     TB1
CWB     CX2
```

Each task in terminal class A0 will have a separate copy of terminal work block TB1 and will be allowed access to common work blocks CX1 and CX2. Tasks in terminal class B0 will have a separate copy of TB1 but they are only allowed access to common work block CX2. Information can be stored in a common work block by one task and accessed by another task. For example the supervisor may enter the current date as part of the start of day routine, then whenever a task requires the date it will obtain it from the date field in the common work block. Note that in terminal class A0 the order of work blocks is:-

```
TERM    A0
TWB     TB1
CWB     CX2
CWB     CX1
```

If CX1 and CX2 had been reversed then an error would occur when accessing work blocks in terminal class B0, as in terminal class B0 common work block CX2 is located after TB1.

IDENTIFIER	PAGE IN M04
CWB	1.3.16

3.2.5.3 User workblocks

In this context "user" is the work station operator. It may be necessary for the application program to maintain accumulators for each work station user, for example. However there need not be a fixed one-to-one relationship between work positions and users. There need not be the same number of users as work stations and they need not be assigned to particular stations.

To maintain user information, areas of memory are required which are associated with individual users and not with work stations. These areas of memory are called user work blocks (UWB). One or more user work block types may be defined for each terminal class.

Tasks may only refer to a user work block if it has been defined for their terminal class and then only when the program has executed a 'USE' instruction specifying the block identifier and index identifier, of that user work block. An index identifier is an integer in the range 1 to 999 which is used to differentiate between user work blocks of the same format.

For example:-

```
            TERM    A0
            TWB     TB1
            CWB     CX2
            UWB     UB1

UB1         BLK
CASID       STRG    6C'*'
DEP         BCD     12D'0'
WHDL        BCD     12D'0'
```

The number of copies of a user work block is entered in the configuration data, see M04 page 3.4.3 for details. If the user work block UB1 had been configured with four copies, then to access copy three of the user work block UB1 the following section of code would be executed.

```
            MOVE    INDX,=W'3'
            USE     UB1,INDX
            ADD     DEP,CSH
```

IDENTIFIER	PAGE IN M04
UWB	1.3.28

3.2.5.4 Dummy work blocks

Dummy work blocks can be used to redefine the data items forming another work block. For example:- it is only possible to read from disk into a string data item, so it is useful to have a work block which contains just the string data item and a dummy work block containing the field definitions to be imposed on this record. In the example below workblock TB1 contains the string data item which the record will be read into and DB1 contains the redefinition.

```
            TWB       TB1
            DWB       DB1(TB1)        Note work block to be redefined is TB1
.
TB1         BLK
BUF         STRG      66

DB1         DBLK                      Note dummy block definition begins DBLK
NAME        STRG      20
ADDR        STRG      30
POSTC       STRG      8
TELNO       STRG      8
```

For example if the contents of BUF are:-

FREDERIC SMYTHE 15, THE LOGWALK NEWTOWN LL5 I11 789-1276

Then the contents of the data item identifiers forming the dummy work block DB1 will be as shown below.

NAME	FREDERIC SMYTHE
ADDR	15, THE LOGWALK NEWTOWN
POSTC	LL5 I11
TELNO	789-1276

IDENTIFIER	PAGE IN M04
DWB	1.3.20

3.2.5.5 Swappable workblocks

These can form the transition between work blocks held in the program and disk held files. Ordinary workblocks can have preset or empty values; each time the application is started other than via IPL, then the program held workblocks are set to the initial state. However swappable (disk held) workblocks contain what ever they held when the machine was last closed down.

The name "swappable work block" is derived from the fact that they can be "swapped" or exchanged between memory and disk storage, enabling values to be updated and held for future use. For a task to use a swapppable work block it must issue the USE command, and when it no longer requires the swappable work block the UNUSE command is issued which releases the block and copies it back to disk. It is not possible for two tasks to have the same swappable workblock in memory at the same point in time, though there can be more than one copy of a work block on disk and different tasks could access these different copies.

The USE and UNUSE commands have two arguments:- the block identifier (SB1) and a data item identifier. The block identifier specifies the name of the swappable work block e.g. SB1 and the data item identifier is used to specify which copy is required.

AT IPL THE THE DISK HELD FILE WILL BE RE-SET TO THE INITIAL CONTENTS.

```
            TERM      A0
            TWB       TB1
            SWB       SB1       Swappable work block
SB1         BLK                 Start of definition for SB1
.....................
Definitions for SB1
.....................
```

The disk file for holding swappable work blocks is called $SWAP and is created at sytem load time, and the configuration file contains the details of the block definitions, see M04 page 3.4.4. Each copy of the work block is referenced by a data item identifier for example:-

```
            TERM      A0
            TWB       TB1
            CWB       CX2
            SWB       SB1
SB1         BLK
CASID       STRG      6C'*'
DEP         BCD       12D'0'
WHDL        BCD       12D'0'
```

If, for example, the swappable work block SB1 had four copies specified in the configuration file, then the following instructions would be required to access information held in the third copy of the swappable work block.

```
            MOVE      INDX,=W'3'
            USE       SB1,INDX
            ADD       DEP,CSH
```

IDENTIFIER	PAGE IN M04
SWB	1.3.25

DATA SET DECLARATION

```
            IDENT    MAIN
            DDIV
            TERM     T0
            TWB      TB1
            TWB      TB2
            CWB      CB1
            START    TGØ
DSKB        DSET     FC=20, DEV=KB
DSCAS       DSET     FC=12, DEV=TC, BUFL=100
DSSOPI      DSET     FC=10, DEV=S1
             )
             }
             }
            PDIV
             )
             }
             }
            END
```

3.2.6 Data set directive (DSET)

In a CREDIT program input and output devices are specified by terminal class, and they are defined using the dataset directive (DSET). This must occur after the appropriate terminal class directive (TERM) in the data division. The DSET directive is used to associate a data set identifier used in the procedure division with the TOSS device type and file code. With the DSET directive it is also possible to specify the buffer length to be used, and if the buffer is to be shared with any other device. The format of the DSET directive is:-

```
data-set-identifier DSET FC=file code
                         {,BUFL=decimal number  }
                         {,DEV=device type      }
                         {,BUFDS=buffer data set}
```

FC folllowed by a hexadecimal number gives the TOSS file code, in the example below the numeric display is using file code 41.

The keyword BUFL is followed by the length in decimal notation of a fixed length buffer to be used with this device. This must not be specified if the I/O operations on the device use a system buffer.
Device type (DEV) is an optional field and its sole purpose is as an aide-memoire to the programmer. It is recommended to use the TOSS device types as listed in M04, as at system generation these device codes will be used to assign the file codes to the required devices for each terminal class.

The keyword BUFDS specifies that the buffer is to be shared with another data set in the same terminal class.

For example:-

```
SCRN      DSET      FC=50,DEV=DY,BUFL=240
AUX       DSET      FC=41,DEV=DN,BUFDS=SCRN
```

Here the VDU (TOSS device type DY) shares a buffer with the numeric display (TOSS device type DN).

DIRECTIVE	PAGE IN M04
DSET	1.3.18

3.2.7 Data items

Before studying all the different types of data items available in CREDIT it is important to have a basic understanding of how information is held within a computer.

The system used by PTS to hold information is called binary; the presence of an item being denoted by the value one and the absence of an item by the value zero. An analogy can be made with a lightbulb, which is either lit (having information) or out (no information).

Within the computer this binary information is held in BITS - one bit holding one binary item; for convenience bits are assembled into larger units called words - each word consisting of sixteen bits. However as much of the work of a modern computer is handling character strings, and all possible characters can be represented by eight bits then the computer word has been divided into two equal sections called BYTES, these bytes are also subdivided into two hexadecimal (base sixteen) digits. This arrangement of units is shown in the table below:-

NAME	INT. REPRESENTATION	CONTENTS
BIT	|.|	can hold a zero or a one
DIGIT	|....|	four bits, can hold one hexadecimal character
BYTE	|........|	eight bits, can hold one ISO-7 character or two hexadecimal digits
WORD	|................|	sixteen bits, two characters, four hexadecimal digits

3.2.7.1 CREDIT data items

CREDIT has several types of data items, and use either bits, digits, bytes or words depending on how the data item is defined. The different types of data items used in CREDIT are described on the next few pages, and listed below.

BOOLEAN
BINARY
BINARY ARRAYS
BINARY CODED DECIMAL
BINARY CODED DECIMAL ARRAYS
STRING
STRING ARRAYS
LITERALS

3.2.7.2 Boolean data items (BOOL)

Each work block can have up to sixteen boolean data items and each boolean data item occupies one BIT. The bit is set to one if the data item holds the value TRUE and zero if it holds the value FALSE. The format of the boolean data item declaration is:-

```
data-item-identifier BOOL [value]
```

The data item identifier is the means by which this data item will be referenced in the PDIV.

The value is an optional field which allows the data-set-identifier to have a preset value. If the value is omitted then the default value of false is assumed. Valid values are shown below:-

```
TRUE
T
FALSE
F
```

Boolean data items must always be the first entries in the work block.

Below are some examples of boolean declarations.

NAME		VALUE	MEMORY CONTENTS
FLAG	BOOL	TRUE	1
NEW	BOOL	T	1
CHNGE	BOOL	FALSE	0
LITE	BOOL	F	0
DLTE	BOOL		0

IDENTIFIER	PAGE IN M04
BOOL	1.3.15

BINARY

BIN	W'20'	⟶ X'0014'
BIN	X 'F3'	⟶ X'00F3'
BIN	5X' 315FA'	⟶ X'15FA'
BIN	4D'0100'	⟶ X'0100'
BIN	D'123456'	⟶ X'3456'
BIN	1D'5'	⟶ X'0005'
BIN	C'NO'	⟶ X'4E4F'
BIN	C'ANO'	⟶ X'4E4F'
BIN		⟶ X'0000'

3.2.7.3 Binary data items (BIN)

A binary data item occupies one word (sixteen bits) and can be used for holding items shown in the table below. The interpretation given to the contents of the data item is dependant on the value code, the default value being type word (W) and value zero. The binary data item declaration format is shown below:

data-item-identifier BIN [[[length]value-type]['value']]

The data item identifier is the means by which this data item will be referenced in the PDIV.

The value type is one of the following:-

Type	Int. representation	Notes
W	One word (1*16 bits)	Number in range -32768 to +32767
C	Two bytes (2*8 bits)	Can contain two ISO-7 characters
X	Four hexadecimal digits (4*4 bits)	Hexadecimal number
D	Three digit decimal number with sign (4*4 bits)	Unused elements set to zero, the sign bit is B for positive, D for negative

Below are listed examples of binary data items, some with values assigned, together with the internal representation.

Data item identifier		Value	Machine form hexadecimal
SP1	BIN	W'32767'	7FFF
SP2	BIN	4D'100'	B100
SP3	BIN	X'FF'	00FF
SP4	BIN	2C'NO'	4E4F
SP5	BIN		0000
SP6	BIN	C'DOMINO'	4E4F
SP7	BIN	3D	B000

Note:-

In SP2 the specified length includes the sign.
In SP6 only the last two characters are held in the item.
In SP7 the largest possible value is 999 although it was only specified as a two digit field plus sign.

IDENTIFIER	PAGE IN M04
BIN	1.3.12

DATA-ITEM-SPECIFICATION

	LENGTH ITEM SIZE	VALUE TYPE	'VALUE'
BCD	NUMBER OF (4 BITS) DIGITS	D	'DECIMAL NUMBER'
		X	'HEXADECIMAL INTEGER'
BIN	1 WORD	W	'DECIMAL NUMBER'
	NUMBER OF (4 BITS) DIGITS	D	'DECIMAL NUMBER'
	NUMBER OF (8 BITS) CHARACTERS	C	'CHARACTER STRING'
	NUMBER OF (4 BITS) HEXADECIMAL DIGITS	X	'HEXADECIMAL INTEGER'
STRG	NUMBER OF (8 BITS) CHARACTERS	C	'CHARACTER STRING'
	NUMBER OF (4 BITS) HEXADECIMAL DIGITS	X	'HEXADECIMAL INTEGER'

3.2.7.4 Binary coded decimal data items (BCD)

BCD data items consist of a number of digits (4 bits) holding either a decimal (base 10) or hexadecimal (base 16) number. The maximum number of digits that can be held in one data item is 255, though in the case of decimal numbers the first digit position is reserved for the sign. The format of this declaration is:-

```
data-item-identifier  BCD  { length, value type [ 'value' ]  }
                           { [length,] value type 'value'    }
                           { [length[, value type]] 'value'  }
```

The data item identifier is the means by which this data item will be referenced in the PDIV.

Length is the number of digits to be used for this data item, including the sign.

Value type - this specifies whether the data item is to be hexadecimal digits (value type X) or decimal digits (value type D); the default type is 'D'

The value field allows a preset value to be given to a data item, as shown in the table below.

Contents of value field	Contents of data item	Example	
		Specified	Contents
Not Given	Zero		0
Less Than specified Length	Value right justified within data item	6D'23'	B'0023
Greater than specified length	Least significant digits will be held	4D'2600'	B'00

Note:-
Either the value or the type and length must be specified, the value must be enclosed in quotes.

Examples of BCD data declarations are given below:-

Data item identifier		Value	Machine form hexadecimal	
BCD1	BCD	6D'-23'	D00023	negative value sign set to D
BCD2	BCD	3D'100'	B100	rounded up to evan no. of bytes
BCD3	BCD	X'FF'	FF	
BCD4	BCD	6D	000000	
BCD5	BCD	'12300'	B12300	(implied D)
BCD6	BCD	X'ACE560'	ACE560	
BCD7	BCD	5X	000000	

DATA-ITEM-SPECIFICATION

	LENGTH ITEM SIZE	VALUE TYPE	'VALUE'
BCD	NUMBER OF (4 BITS) DIGITS	D	'DECIMAL NUMBER'
		X	'HEXADECIMAL INTEGER'
BIN	1 WORD	W	'DECIMAL NUMBER'
	NUMBER OF (4 BITS) DIGITS	D	'DECIMAL NUMBER'
	NUMBER OF (8 BITS) CHARACTERS	C	'CHARACTER STRING'
	NUMBER OF (4 BITS) HEXADECIMAL DIGITS	X	'HEXADECIMAL INTEGER'
STRG	NUMBER OF (8 BITS) CHARACTERS	C	'CHARACTER STRING'
	NUMBER OF (4 BITS) HEXADECIMAL DIGITS	X	'HEXADECIMAL INTEGER'

3.2.7.5 String data items (STRG)

String data items are composed of 1-4095 bytes, each byte holding one alpha-numeric character. The format of the string data item identifier is:-

```
data-item-identifier STRG {[[ Length,] Value type]  'Value'}
                          { [ Length [,Value type]] 'Value'}
                          { Length, Value type ['Value']}
```

The data item identifier is the means by which this data item will be referenced in the PDIV.

Length - is the number of characters that will make up the string

Value type - is either character (type C) or hexadecimal (type X), the default being 'C'.

Value - is an optional field and allows a data item to have a preset value

Contents of value field	Contents of data item	Example	
		Specified	Contents
Not given	Spaces		
Less than specified length	Value left justified within data item. Last character repeated	6C'AC'	ACCCCC
		6C'AC '	AC
Greater than specified length	Leftmost characters will be stored	4C'BRANCH'	BRAN

Data item identifier	Type	Value	Machine form hexadecimal
STRG1	STRG	6C	202020202020
STRG2	STRG	3C'ABC'	414243
STRG3	STRG	X'FF'	FF
STRG4	STRG	6C'ABC'	414243434343
STRG5	STRG	6C'ABC '	414243202020
STRG6	STRG	5C'BANK ID'	42414E4B20
STRG7	STRG	5X	00000

IDENTIFIER	PAGE IN M04
STRG	1.3.23

3.2.7.6 Arrays

There are three types of arrays in CREDIT: BCDI, BINI and STRGI. They can be either one or two dimensional. The subscript must be a binary data item. The maximum subscript is 32767 for one-dimensional arrays. For two-dimensional arrays neither subscript can exceed 255. Arrays occupy two entries in a workblock, unless an array is the last item in the workblock when it occupies only one. The rules for the storage of the values are the same as for BCD, BIN and STRG data items described above.

The general format for an array declaration is:-

data-item-identifier type (S1[,S2])[,[length] type] ['value'...]

S1 - is the "row" subscript

S2- is the "column" subscript for two dimensional arrays

'value'... - this enables the array to be initialised. If fewer values are provided than there are elements in the array, then all remaining elements are filled with the last provided value, e.g.

```
BRANCH    STRGI     (40),10C'LONDON ','ROME ','PARIS ','*'
```

This sets up a forty element array containing branch locations; as only three exist at this point the unused elements are filled with asterisks. The first six elements of this array are shown below:-

element	contents
(1)	LONDON
(2)	ROME
(3)	PARIS
(4)	**********
(5)	**********
(6)	**********

If the contents of an element are less then the string length then the last character is repeated, if for example the declaration had taken the following form:-

```
BRANCH    STRGI     (40),10C'LONDON','ROME','PARIS','*'
```

Then the contents of the first six elements would be:-

element	contents
(1)	LONDONNNNN
(2)	ROMEEEEEE
(3)	PARISSSSSS
(4)	**********
(5)	**********
(6)	**********

For a two dimension array the data items should be ordered by rows then columns, for example:-

```
MONQUA    STRGI    (3,4),10C'JANUARY ','FEBRUARY ','MARCH ','APRIL ',
                   'MAY','JUNE ','JULY ','AUGUST ','SEPTEMBER ',
                   'OCTOBER ','NOVEMBER ','DECEMBER '
```

This would set up a table of the months in each of the four quarters of the year, the month in the quarter being the first subscript, the second the quarter of the year. The machine representation would be:-

Element	Contents	Element	Contents	Element	Contents
(1,1)	JANUARY	(2,1)	FEBRUARY	(3,1)	MARCH
(1,2)	APRIL	(2,2)	MAY	(3,2)	JUNE
(1,3)	JULY	(2,3)	AUGUST	(3,3)	SEPTEMBER
(1,4)	OCTOBER	(2,4)	NOVEMBER	(3,4)	DECEMBER

The binary declaration is similar to that for strings, an example of a two dimensional binary array is given below.

```
TAB    BINI    (4,4),'1','2','3','4','5','6','7','8','9','10',X'B',
               X'C',X'D',X'E',X'F',X'10'
```

Element	Contents	Element	Contents	Element	Contents	Element	Contents
(1,1)	0001	(2,1)	0002	(3,1)	0003	(4,1)	0004
(1,2)	0005	(2,2)	0006	(3,2)	0007	(4,2)	0008
(1,3)	0009	(2,3)	000A	(3,3)	000B	(4,3)	000C
(1,4)	000D	(2,4)	000E	(3,4)	000F	(4,4)	0010

IDENTIFIER	PAGE IN M04
BCDI	1.3.11
BINI	1.3.13
STRGI	1.3.24

3.2.8 Literals - overview

With CREDIT there are four distinct catagories of literals; these are literal constants, keytables, pictures and format lists; after translation each of these literals will be held in separate pools.

3.2.8.1 Literal constants

These are the normal form of constants used in a program and have the general form:-

=[Value type]'value'

Value type is one of those listed in the table below

Type	Int. representation	Notes
W	One word (1*16 bits)	Number in range -32768 to +32767
C	(n) bytes (n*8 bits)	Can contain (n) ISO-7 characters
X	(n) hexadecimal digits (n*4 bits)	Hexadecimal number n digits long
D	(n-1) digit decimal number with sign (n*4 bits)	Unused elements set to hex F Sign bit is B for positive, D for negative

Note:-
Literal constants can never form the destination part of an instruction, in that a constant may be added or moved to a data item, but a data item can not be added or moved to a literal constant.

Examples of literals are:-

='2'	Typeless literal
=W'45'	One word containing value 45
=C'BANK ID.'	Character string
=6D'-97892'	BCD constant
=X'2030'	Hexadecimal constant

3.2.8.2 Keytables

These are used for holding lists of character codes which could be used to terminate keyboard input. Only hexadecimal (type X) or character (type C) data items may be used in a keytable. A description of keytables is given in section 6.2.3. The format of a keytable declaration is:-

```
    key-table-name KTAB {literal constants  }
                        {EQU data items     }
```

For example

```
    BSP         EQU         X'05'
    CLEAR       EQU         X'19'
    CLR2        EQU         X'00'
    EOI         EQU         X'12'
    CANC1       EQU         X'14'
    CANC2       EQU         X'15'
    SPKTAB1     KTAB        BSP,CLEAR,CLR2,EOI,CANC1,CANC2
```

3.2.8.3 Picture literals

These are used when formatting numeric items for display or printing purposes; either for output of for re-displaying an input item. Picture literals can only be used with the FMEL instruction. Some examples of picture literals are shown in the table below:

Picture	Data item	Result
'AAA999'	BFF0456	0456
'XXY-XX'	2702	27-02
'XXY-XX'	FFFF	
'9909'	B123	1203
'F+**V9'	DF11	*-1.1
'99,99'	B123	12,34
'99B99'	6521	65 02

3.2.8.4 Format lists

These are used to hold format layouts for the output of information. Examples of format lists are shown below, and are described in more detail in section 6.5.

```
ERFM01    FRMT
          FSL
          FTEXT     'TOO FEW INPUT CHARACTERS'
          FMEND
*
ERFM02    FRMT
          FCOPY     =X`2031'
          FTEXT     'RETYPE ANWER YES OR NO: '
          FMEND
```