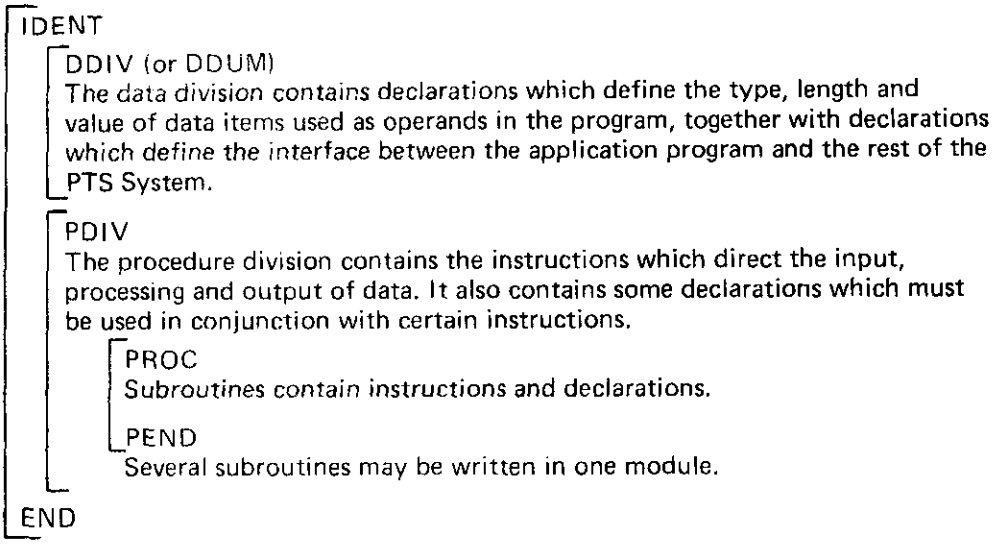
1.2 Directives

1.2.1 *Structure Directives*

The framework of a CREDIT module is constructed from the directives IDENT, DDIV, DDUM, PDIV, PROC, PEND and END. The use of these directives is illustrated below:



The IDENT and END directives define the start and end of a module. They must be the first and last statements, respectively, of the module.
The DDIV or DDUM directive defines the start of the data division. It must be the second statement in the module. DDIV is used in the main module of a program. DDUM is used in all other modules.
The PDIV directive defines the start of the procedure division. The PROC and PEND directives define the start and end of a subroutine.
The IDENT, DDIV (or DDUM), PDIV and END directives must appear once only in each module. The PROC and PEND directives may be repeated. However, subroutines may not be nested. That is, two or more PROC directives cannot be written without an intervening PEND directive. Subroutines may, though, perform other subroutines.

1.2.2 *Linkage Directives*

CREDIT modules which have to be linked into an application program contain references to statements or subroutines in other modules. In order to achieve the correct inter-module linkages, entry points and external references must be specified. The ENTRY and EXT directives are used for this purpose. They must be written in the procedure division.
There must be at least one START directive for each program. When the TOSS System is started (i.e. the TOSS Monitor is loaded and the application program begins execution) a task is activated for each work position in the System. The tasks are activated at the start points specified in the START directives of the relevant terminal classes. The START directive(s) must be written in the data division and must be specified as entry points (ENTRY).

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 start points become pending. They will be activated after the first task has executed an EXIT instruction. When using memory management, the REENTER directive refers to a closing routine in case of a page fault or read error on disc. The statement identifier in this directive must be declared as ENTRY in the module it is defined.

1.2.3 *Listing Directives*

The directives LIST, NLIST and EJECT are used to control the CREDIT listing during the translation process. They may be written in any part of the module. The OPTNS directive must follow after the DDUM or DDIV directive.

1.2.4 *Equate directive*

This directive is used to equate an identifier with a value. When this identifier is used in an instruction the Translator automatically replaces it with the specified value, i.e. the instruction is translated as if the programmer had actually written the value in the instruction.
The directive may be used free in the procedure division (PDIV), but it should follow the ENTRY and EXT directives.

1.2.5 *Parameter directive*

The directives PFRMT, PKTAB and PLIT define the type of formal parameter, declared in the heading of the subroutine and must be used in two byte addressing mode or when a format table is used as formal parameter, or when in one byte addressing mode the literal constant name, keytable name or formatlist name, does not begin with a $ sign. These directives follow the PROC directive immediately.

1.2.6 *Directive reference*

This section describes the syntax and use of each directive. The possible values of the variables in the directives is given in appendix 1. The notation conventions are described in section 1.1.5.

DDIV *Data Division* DDIV

Syntax: ␣ DDIV ␣ [module-name]

Type: Structure directive

Description: Indicates the beginning of the data division of a module and causes a page feed in the listing during translation.

If module-name is specified, the data division statements will be fetched from the module indicated by module-name.

However, these data divisions will be entirely separate at execution time. The use of *common* data divisions is achieved through the DDUM directive.

No more than one module in a CREDIT program may contain a DDIV directive. The remaining modules must use the DDUM directive.

DDUM *Data division dummy* **DDUM**

Syntax: ␣ DDUM ␣[module-name]

Type: Structure directive.

Description: Beginning of a data division is indicated and the data division statements will be fetched from the module indicated by module-name.

No object code is output during processing of the data division.

Note: A DDUM-module may contain declaration of data items. The only difference between DDUM and DDIV directives is, when DDIV is declared the object modules of the data division are output on a /0 type file.

EJECT

Eject

EJECT

Syntax: ␣ EJECT ␣

Type: Listing directive.

Description: Listing will be continued at the top of the next page.

END *End* **END**

Syntax: ␣ END ␣

Type: Structure directive.

Description: This directive terminates a module. It must be the last statement appearing in each module.

ENTRY *Entry Point* **ENTRY**

Syntax: ␣ ENTRY ␣ { subroutine-identifier / statement-identifier }

Type: Linkage directive.

Description: The statement-identifier or subroutine-identifier within this module may be referred to by other modules.

Each identifier may comprise a maximum of six characters.

EQU *Equate* EQU

Syntax: equate-identifier ␣ EQU ␣ value-expression

Type: Equate directive.

Description: Equate-identifier takes on the type and value as specified by value-expression. The value must be between 0 and 255 inclusive.
It should be used after the ENTRY and EXT directives, but may be written everywhere in the procedure division.

Example:
```
KEY     EQU     X'09'
KEY2    EQU     KEY+2
```

EXT *External reference* **EXT**

Syntax: ␣ EXT ␣ external-identifier

Type: Linkage directive.

Description: External-identifier points to a statement identifier or a subroutine identifier which is not in this module.

External-identifier consists of a maximum of six characters.

IDENT *Identification* **IDENT**

Syntax: ␣ IDENT ␣ module-name

Type: Structure directive.

Description: This directive specifies the name given to a module. It must always be present and must be the first statement in a module. Module-name may consist of a maximum of 8 characters, but it is truncated by the translator to 6 characters.
If comment follows the IDENT directive, a module name of at least 6 characters is recommended.

INCLUDE *Include* **INCLUDE**

Syntax: ␣ INCLUDE ␣ module-name [, LIST]

Type: Directive.

Description: This directive enables source code to be shared between modules in an application. The directive is used in the procedure division. The contents of the auxiliary file specified by module-name is included in the translation process. The auxiliary input file must not include the module directives IDENT, END or an INCLUDE directive.
If the option, LIST is specified, the source statements from the auxiliary file are listed. Otherwise they are not listed.
In the program listing, lines from the auxiliary file are identified by a hypen which follows directly the line number.

Example: ␣ INCLUDE MAUX, LIST

LIST *List* **LIST**

Syntax: ␣ LIST ␣

Type: Listing directive.

Description: CREDIT source code listing is resumed after it has been suspended by a NLIST directive.

NLIST *No list* **NLIST**

Syntax: ␣ NLIST ␣

Type: Listing directive.

Description: CREDIT source code listing is suspended from the point where this directive is given until either the END directive or LIST directive is met. Lines which contain syntax errors will continue to be printed during this phase.

OPTNS *Options* OPTNS

Syntax: OPTNS LINES=value [,LITADR=decimal-integer]
[,ADRMOD=decimal-integer]
LITADR=decimal-integer [,LINES=value]
[,ADRMOD=decimal-integer]
ADRMOD=decimal-integer [,LINES=value]
[,LITADR=decimal-integer]

Type: Directive.

Description: This directive should follow immediately after a DDIV or DDUM directive and is valid for the whole program (global). The keyword parameters are LINES, LITADR and ADRMOD. They have the following significance.

LINES — Value specifies the number of lines per page on the output listing.

LITADR — Literal constant, pictures, keytables and formats are addressed by a one or two byte addressing system. A four digit, decimal integer indicates which addressing system is used. The first digit refers to literal constant, the second digit to keytables, the third to pictures and the fourth to formats. Each digit may have a value 1 or 2.

First digit:
- 1 One byte addressing system selected for literal constants.
- 2 Two byte addressing system selected for literal constants.

Second digit:
- 1 One byte addressing system selected for keytables.
- 2 Two byts addressing system selected for keytables.

Third digit:
- 1 One byte addressing system selected for pictures.
- 2 Two byte addressing system selected for pictures.

Fourth digit:
- 1 One byte addressing system selected for formats.
- 2 Two byte addressing system selected for formats.

Default value LITADR=1111.

OPTNS *Continued* **OPTNS**

ADRMOD With one digit, a two byte addressing mode can be selected for data-items, data-sets, literal constants, keytables, formats and pictures.

1 One byte addressing for data-items, data-sets, literal constants, keytables, formats and pictures. With LITADR two byte addressing can be selected for literal constants, keytables, pictures and/or formats.

2 Two byte addressing for data-items, data-sets, literal constants, keytables, formats and pictures. LITADR is automatically set to two byte addressing.

Default value ADRMOD=1.

PDIV *Procedure division* **PDIV**

Syntax : ␣ PDIV ␣

Type : Structure directive

Description : Indicates the beginning of the procedure division and causes the data division processing to be terminated. A new page of program listing is thrown.

PEND *Subroutine end* **PEND**

Syntax: ␣ PEND ␣

Type: Structure directive.

Description: This directive marks the end of a subroutine. Formal parameters may not be referred to by an instruction which is written after a subroutine PEND directive.

PFRMT *Formal format list parameters* **PFRMT**

Syntax: — PFRMT — {identifier / $identifier}

Type: Parameter directive.

Description: In two-byte addressing mode this directive defines a formal parameter of the type format list reference. It follows the PROC directive and must also be used when the formal parameter corresponds to a format table reference (FTABLE).

The $ sign is counted in the total number of characters in the identifier. (formal parameter) When the formal parameter does not start with a $ sign and the referenced type is format list, then the type has to be defined by a PFRMT directive, also in one byte addressing mode.

Example:

```
SUBF_PROC_$FORM1          (ADRMOD=2)
      PFRMT  $FORM1
        .
        .
        .
SUBF_PROC_$FORM1          (ADRMOD=1)
        .
        .
        .
SUBF_PROC_FORM1           (ADRMOD=1)
      PFRMT  FORM1
        .
        .
        .
```

PKTAB *Format key table parameter* **PKTAB**

Syntax: ⎵ PKTAB ⎵ {identifier / $identifier}

Type: Parameter directive

Description: In two-byte addressing mode, this directive defines a formal parameter of the type key table reference. It follows the PROC directive.

The $ sign is counted in the total number of characters in the identifier (formal parameter). When the formal parameter does not start with a $ sign and the referenced type is key table, then the type has to be defined by a PKTAB directive, also in one byte addressing mode.

Example:

```
SUBK ⎵ PROC ⎵ $SKTB1          (ADRMOD=2)
       PKTAB   $KTB1
         .
         .
         .
SUBK ⎵ PROC ⎵ $KTB1           (ADRMOD=1)
         .
         .
         .
SUBK ⎵ PROC ⎵ KTB1            (ADRMOD=1)
       PKTAB   KTB1
         .
         .
         .
```

PLIT *Formal literal parameter* **PLIT**

Syntax: ␣ PLIT ␣ {identifier / $identifier}

Type: Parameter directive

Description: In two byte addressing mode, this directive defines a formal parameter of the type literal reference. It follows the PROC directive.

The $ sign is counted in the total number of characters in the identifier. (formal parameter) When the formal parameter does not start with a $ sign and the referenced type is literal constant then the type has to be defined by a PLIT directive, also in one byte addressing mode.

Example:

```
SUBL ␣ PROC ␣ $LITC          (ADRMOD=2)
       PLIT   $LITC
        .
        .
SUBL ␣ PROC ␣ $LITC          (ADRMOD=1)
        .
        .
        .
SUBL ␣ PROC ␣ LITC           (ADRMOD=1)
       PLIT   LITC
        .
        .
        .
```

PROC *Subroutine start* **PROC**

Syntax: Subroutine-identifier␣PROC␣[formal-parameter] [,formal-parameter] ....

Type: Structure directive.

Description: The PROC directive forms the heading of a subroutine. A maximum of eight formal parameters may be specified. (ADRMOD=2).
With ADRMOD=1 the number of formal parameters is limited to 8 bytes. When using literal constants, keytables or format lists as formal parameter in two byte addressing mode, selected with the LITADR option, the maximum number of formal parameters is decreased.

REENTER *Reenter* **REENTER**

Syntax: ⎵ REENTER ⎵ statement-identifier.

Type: Linkage directive.

Description: When using memory management, this directive refers by means of the statement identifier to a closing routine. This routine, written by the user, will be executed when it is impossible to read a code segment in main memory or from disk.
This directive should be located following the start directive.
Statement-identifier must be declared as ENTRY in the module it is defined.

START *Start Point* **START**

Syntax: ␣ START ␣ { statement-identifier / external-identifier }

Type: Linkage directive.

Description: For each terminal class, START indicates the first instruction to be executed in the program.
This directive should precede the STACK declaration.
The start address has to be specified as an ENTRY in the relevant module.