
International Standard



4342

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION

Numerical control of machines — NC processor input — Basic part program reference language

Commande numérique des machines — Données d'entrée des processeurs CN — Langage de référence de base pour programme de pièce

First edition — 1985-12-15

UDC 681.323 : 621.9-52 : 681.3.06

Ref. No. ISO 4342-1985 (E)

Descriptors : machine tools, automation, data processing, numerical control, programming (computers), programming languages.

AVA Documents
Document Sharing Hub
Price based on 278 pages

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work.

Draft International Standards adopted by the technical committees are circulated to the member bodies for approval before their acceptance as International Standards by the ISO Council. They are approved in accordance with ISO procedures requiring at least 75 % approval by the member bodies voting.

International Standard ISO 4342 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems*.

Users should note that all International Standards undergo revision from time to time and that any reference made herein to any other International Standard implies its latest edition, unless otherwise stated.

Contents

	Page
0 Introduction	1
1 Scope and field of application	2
2 References	2
3 Coordinate system	2
4 Language structure	3
4.1 General comments	3
4.2 Letters	4
4.3 Digits	5
4.4 Special characters	6
4.5 Characters	7
4.6 Symbol for literal delimiter	8
4.7 Literal character string	9
4.8 Unsigned numbers	10
4.9 Keywords	11
4.10 Simple identifiers	12
4.11 Identifiers	13
4.12 Labels	14
4.13 Statements	15
4.14 Nesting	16
4.15 Part program	17
5 Arithmetic statements	18
5.1 General comments	18
5.2 Arithmetic operators	20
5.3 Arithmetic functions	21
6 Program definition statements	27
6.1 General comments	27
6.2 The synonym statement; SYN/	28
6.3 Reservation for subscripting; RESERV/	29
6.4 Definition of a macro; MACRO/	30

7	Program execution statements	32
7.1	Part program control	32
7.2	Part program identification and termination	33
7.3	Machine and no-postprocessing statements	35
7.4	Input/output statements	37
7.5	Loop start and end statements and transfer statements	42
7.6	Copy statement and index specification	46
7.7	Macro execution statement	49
7.8	Remark statement	50
7.9	Postprocessing print statement	51
8	Geometrical definitions statements	52
8.1	General comments	52
8.2	Declarations of reference system	54
8.3	Declarations of z -surface	57
8.4	Definitions of a point	64
8.5	Definitions of a pattern of points	80
8.6	Definitions of a line	99
8.7	Definitions of a plane	118
8.8	Definitions of a vector	127
8.9	Definitions of a circle	138
8.10	Definitions of a cylinder	159
8.11	Definitions of a sphere	164
8.12	Definitions of a cone	170
8.13	Definitions of an ellipse	173
8.14	Definitions of a hyperbola	175
8.15	Definitions of a lofted conic	177
8.16	Definitions of general conic	181
8.17	Definitions of a general quadric	185
8.18	Definitions of a tabulated cylinder	189
8.19	Definitions of a matrix	194
8.20	Definitions of a ruled surface	206

9	Geometric execution statements	209
9.1	General comments	209
9.2	General semantics	210
9.3	Continuous motion statement	220
9.4	Motion statements	225
9.5	Tool path control statements	233
9.6	Startup direction control statements	243
9.7	Transform cutter location statements	247
9.8	Processor output file control statement	248

Annexes

A	Rules for representing the RL on punched cards	249
B	Syntax description of the reference language	250
C	List of recommended synonyms	272
D	Alphabetical list of major words and locations	273

Numerical control of machines — NC processor input — Basic part program reference language

0 Introduction

0.1 General features

0.1.1 A part program is an ordered set of instructions in a language and in a format required to cause the generation and commands for operations to be carried out under numerical control (NC). The instructions of the language are processed sequentially in two stages, processor and postprocessor. The processor stage is substantially independent of the numerically controlled machine, and the postprocessor stage takes account of the features of the machine and control system to be numerically controlled.

0.1.2 Numerical control is applied to many types of machine, but the language defined in this International Standard has been developed primarily for numerically controlled machine tools — hence the words “cutter” or “tool”, and “workpiece” or “part”, are used in the description of the language to indicate the working element and processed element respectively. Many of the reserved words of the language are also derived from metalworking terminology.

0.1.3 This International Standard is partly informative and partly definitive, with the intention of enabling unsophisticated users and potential users to discuss and define their requirements with suppliers. The formal syntax description of the language has been included to aid implementers.

0.1.4 The language described in this International Standard is a reference language (RL). It has been divided into logical pages according to the type of statement or facility, with the intention that each logical page will become a unit of further standardization work. It is also intended that each logical page is suitably identified as belonging to the core or a module of the total reference language.

0.1.5 The output from the processor (CLDATA — derived from the term “cutter location data”), which is also input to the postprocessor, is the subject of separate standardization documents. The definition of CLDATA effectively encompasses the part programming language necessary for using capabilities of the postprocessor, and the CLDATA standard ISO 3592 is the prime reference source for this part of the total language.

0.1.6 Some processors may interface with workshop technology processors. That part of the total language which

encompasses the use of workshop technology will be described in an addendum to this International Standard.

0.2 Numerical control reference language

0.2.1 The numerical control (NC) reference language (RL) is a problem-oriented language developed for the machining of parts. It is similar to scientific programming languages, and contains many of the facilities and a large part of the computational ability of these; additionally the RL provides for both the description of shapes and commands for movement of the machine.

0.2.2 The RL is written in lines, and for the purpose of describing the RL, the level of communication assumed is the manuscript stage of programming. There is no special character for statement termination, but there is a special character to signify that a statement continues from one line to the next. This concept permits the description of the RL itself to be distinguished from the rules for representation of the RL on various media such as punched cards or paper tape. The rules for representing the RL on punched cards are given in annex A. The syntax description of the reference language is given in annex B.

0.2.3 The RL is a symbolic language; that is, an entity may be assigned to a symbolic name and the entity referenced later in the part program by that symbolic name. For other than arithmetical values a symbolic name may not normally be reassigned.

0.2.4 Unlike scientific programming languages there are no explicit or implicit type associations in the RL. The type of entity is determined by the type of statement in which the entity is defined.

0.2.5 The arithmetical type of entity has always a real value, that is, it is an approximate representation of a real number. It has an integer part and a fractional part and can only represent a certain number of the most significant digits of the number, depending on the implementation.

0.2.6 A significant feature of the RL is the “nesting” facility, by which an entity may be defined in parenthesis instead of using a symbolic name.

1 Scope and field of application

This International Standard defines a higher-level symbolic part-programming language which is processed by a digital computer to produce a NC machine program.

The language has been developed primarily for numerically controlled machine tools. It is a reference language, which means that it is not necessarily intended that the whole of the language be implemented. It is expected that parts, or subsets, of the language will be implemented to suit particular circumstances.

2 References

ISO 646, *Information processing — ISO 7-bit coded character set for information processing interchange*.

ISO 841, *Numerical control of machines — Axis and motion nomenclature*.

ISO 3592, *Numerical control of machines — NC processor output — Logical structure (and major words)*.

ISO 4343, *Numerical control of machines — NC processor output — Minor elements of 2000-type records (post-processor commands)*.

3 Coordinate system

3.1 ISO 841 is the basis for defining the coordinate system of the RL.

3.2 The coordinate system is a right-handed rectangular Cartesian system, related to a workpiece mounted on a machine and aligned with the principal linear slideways of that machine. The positive direction of movement of a component of a machine is that which causes an increasing positive dimension on the workpiece.

3.3 In the RL, the reference axes of the coordinate system are X , Y and Z . These are used in the description of the workpiece and it is assumed that the workpiece is stationary, with the tool or cutter moving relative to the workpiece coordinate system whether or not this is true for the actual NC machine tool operation.

3.4 When specifying angles of planes the positive direction is counterclockwise and the reference axis is as follows :

Plane	Reference axis
XY	X
YZ	Y
ZX	Z

3.5 The positive direction of angle is counterclockwise from the reference axis.

3.6 Angles are expressed in degrees and decimal fractions of a degree.

3.7 The output from the processor (CLDATA) uses the same conventions as the RL, and the output coordinates refer to a reference point on a cutter (usually the centre of the tip) relative to the workpiece coordinate system used in the part program.

4 Language structure

4.1 General comments

4.1.1 General semantics

Digits and letters are used to create unsigned numbers and keywords which in conjunction with characters and special characters may be used to create identifiers, labels and literal character strings. Any valid combination, if existing, may be used to construct a statement and a number of statements arranged in a specific order constitutes a part program.

4.1.2 Sub-contents

For

- 1) letters, see 4.2;
- 2) digits, see 4.3;
- 3) special characters, see 4.4;
- 4) characters, see 4.5;
- 5) symbol for literal delimiter, see 4.6;
- 6) literal character string, see 4.7;
- 7) unsigned numbers, see 4.8;
- 8) keywords, see 4.9;
- 9) simple identifiers, see 4.10;
- 10) identifiers, see 4.11;
- 11) labels, see 4.12;
- 12) statements, see 4.13;
- 13) nesting, see 4.14;
- 14) part program, see 4.15.

4.1.3 Limitations

None.

4.1.4 Syntax

$$\langle \text{language structure} \rangle : : = \left\{ \begin{array}{l} \langle \text{letters} \rangle \langle \text{digits} \rangle \langle \text{special characters} \rangle \langle \text{characters} \rangle \\ \langle \text{symbol for literal delimiter} \rangle \langle \text{literal character string} \rangle \langle \text{unsigned numbers} \rangle \\ \langle \text{keywords} \rangle \langle \text{simple identifiers} \rangle \langle \text{labels} \rangle \langle \text{statements} \rangle \langle \text{nesting} \rangle \\ \langle \text{part program} \rangle \end{array} \right\}$$

4.2 Letters

4.2.1 Semantics

Letters have no individual meaning, being used for forming keywords, simple identifiers, character strings or labels.

4.2.2 Limitations

None.

4.2.3 Syntax

<letter> :: = A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

4.3 Digits

4.3.1 Semantics

Digits have no individual meaning, being used for forming simple identifiers, unsigned numbers, character strings or labels.

4.3.2 Limitations

None.

4.3.3 Syntax

< digit > : : = 0|1|2|3|4|5|6|7|8|9

4.4 Special characters

4.4.1 Semantics

Special characters are used as operators for building up arithmetic expressions and as punctuation marks (or separators) in statements. When special characters are used in literal character strings they are treated as characters with no syntactical significance.

- + − ∗ / ↑ arithmetic operators (see 5.2).
-) closing parenthesis, used as a statement label separator (see 4.11) or in connection with opening parenthesis for nesting.
- (opening parenthesis, used with closing parenthesis in subscripting or nesting, or function arguments.
- . decimal point.
- = equals, used for assigning an entity to a name.
- / slant, used as a separator between a major keyword and the remainder of a statement.
- , comma, used as a separator between elements of a statement.
- : statement label separator (see 4.11).
- \$ dollar or other currency character used to couple statements and delimit the start of a comment field.
- ; semi-colon, used as a separator between statements.
- ' apostrophe, used for delimitation of a character string.

The space character has no significance except in character strings.

4.4.2 Limitations

None.

4.4.3 Syntax

< special character > : : = − | . | + | ∗ | / | , | = | (|) | \$ | ↑ | ; | : | ' |

NOTE — \$ is given as an example of a national currency character.

4.5 Characters

4.5.1 Semantics

A character is a letter, digit or special character or other valid character.

4.5.2 Limitations

None.

4.5.3 Syntax

$\langle \text{character} \rangle :: = \langle \text{letter} \rangle \mid \langle \text{digit} \rangle \mid \langle \text{special character} \rangle \mid \langle \text{other valid character} \rangle$

NOTE — Other valid characters have no significance within the language but are nevertheless considered as valid input. These characters are not otherwise defined in this International Standard. They should be manageable by the specific implementation and be selected from the character set defined by ISO 646 and ISO 840.

4.6 Symbol for literal delimiter

4.6.1 Semantics

The apostrophe is used at the beginning and at the end of a literal character string to indicate the extent of the literal string field.

4.6.2 Example

PARTNO/'VALVE HOUSING'

4.6.3 Limitations

None.

4.6.4 Syntax

< symbol for literal delimiter > : : = ' '

4.7 Literal character string

4.7.1 Semantics

A literal character string may be used in statements, for listing text, or in the postprocessor statements such as INSERT for passing special information through to the postprocessor on the CLDATA. The set of characters permissible is not limited to the set of letters, digits, and special characters defined in this International Standard. Within a literal character string, any special characters are treated simply as characters without syntactical significance.

4.7.2 Example

P1 = POINT/0,0,0 'COMPONENT DATUM'

4.7.3 Limitations

None.

4.7.4 Syntax

< literal character string > : : = < symbol for literal delimiter > $\overset{k}{0}$ [< character >] < symbol for literal delimiter >

NOTES

- 1 The syntax of a literal character string implies that the empty string is allowed.
- 2 Space characters are significant.
- 3 A remark (comment) following a single or double currency character need not be a delimited character string.
- 4 A literal character string not terminated by a closing apostrophe prior to an arbitrary line limit (for example the card column 73 in annex A) is continued on the next line without the need for a currency character.
- 5 An apostrophe is represented by two apostrophes in a character string delimited by apostrophes.

4.8 Unsigned numbers

4.8.1 Semantics

Numbers have their usual meaning, being built up of decimal digits and may have a decimal point. If no decimal point is included in the number, it is assumed to appear after the rightmost digit.

4.8.2 Examples

47711

58.

3.14

4.8.3 Limitations

There is no defined limit to the number of digits within a number, the limit being implementation dependent. No distinctions are made between integer and real numbers since all are used internally in the real mode. In every case where an integer value is required (for example in a subscript) the fractional part of the real value is truncated.

Number representation within a computer is not necessarily exact. Therefore, approximations are used where necessary to achieve the effect of exact operations. These approximations are computer dependent.

4.8.4 Syntax

$$\langle \text{unsigned number} \rangle : : = \begin{matrix} k \\ 1 \end{matrix} [\langle \text{digit} \rangle] [.] \begin{matrix} k \\ 1 \end{matrix} [\langle \text{digit} \rangle] . \begin{matrix} k \\ 1 \end{matrix} [\langle \text{digit} \rangle]$$

4.9 Keywords

4.9.1 Semantics

Keywords have a fixed meaning within the language. They may be regarded as entries in a vocabulary list.

The set of keywords is not fixed but may be enlarged in future revisions of this International Standard.

Keywords do not have the role of delimiters, in contrast to some scientific programming languages for example, as two adjacent keywords have to be separated from each other by a special character.

Keywords are divided into two classes, the major keywords which define the type of the statement, and the minor keywords which give additional information. These classes can be divided into two subclasses each : the processor and the postprocessor keywords. Postprocessor keywords are listed and defined in ISO 4343. An alphabetical list of all keywords is given in annex D.

4.9.2 Examples

POINT

INTOF

4.9.3 Limitations

Keywords consist only of letters and have at least two letters.

4.9.4 Syntax

$$\langle \text{keyword} \rangle : : = \overset{6}{2} [\langle \text{letter} \rangle]$$

4.10 Simple identifiers

4.10.1 Semantics

Simple identifiers have no inherent meaning and are used only as specifiers (see 8.1.4), labels or arithmetic symbols.

4.10.2 Examples

P2

L1

BOBBY1

4.10.3 Limitations

A simple identifier shall not be redefined within a part program except when used as an arithmetic symbol. It is recommended that processors should not accept keywords as simple identifiers. A simple identifier, unless used as a synonym, of more than one letter should contain a digit.

4.10.4 Syntax

$$\langle \text{simple identifier} \rangle :: = \langle \text{letter} \rangle \overset{5}{\underset{0}{[\langle \text{letter} \rangle \mid \langle \text{digit} \rangle]}}$$

4.11 Identifiers

4.11.1 Semantics

An identifier is similar to a simple identifier but normally carries a subscript.

4.11.2 Examples

P2(1)

P2(5)

4.11.3 Limitations

An identifier shall not be redefined within a part program except when used as an arithmetic symbol.

4.11.4 Syntax

$$\langle \text{identifier} \rangle :: = \langle \text{simple identifier} \rangle \begin{matrix} 1 \\ 0 \end{matrix} [(\langle \text{scalar} \rangle)]$$

4.12 Labels

4.12.1 Semantics

A part programmer may attach a label to a statement for statement identification. The label may be referred to in a conditional or unconditional transfer statement or a geometric transfer statement.

4.12.2 Example

A1) GOTO/P1

or alternatively A1 : GOTO/P1

where A1 is the label.

4.12.3 Limitations

None.

4.12.4 Syntax

$\langle \text{label} \rangle :: = \bigcup_1^6 [\langle \text{digit} \rangle] \mid \langle \text{simple identifier} \rangle$

4.12.5 Cross-reference (see annex D)

IF

JUMPTO

TRANTO

4.13 Statements

4.13.1 Semantics

A statement is a complete instruction or unit of information comparable with a sentence in a natural language. A statement consists of a single keyword or a syntactically defined sequence of elements separated by special characters. These elements are keywords, simple identifiers, unsigned numbers and character strings.

There are several types of statement, grouped for convenience of description into the following five main clauses of this International Standard : clause 5 arithmetic, clause 6 program definition, clause 7 program execution, clause 8 geometrical definition and clause 9 geometric execution.

Statements may be labelled or unlabelled. Wherever a keyword is allowed within a statement, it may be replaced by a previously defined synonym.

Everywhere within a macro where a keyword, a simple identifier, or unsigned number, is allowed, it may be replaced by a declared macro parameter name.

4.13.2 Limitations

None.

4.13.3 Syntax

$$\langle \text{statement} \rangle :: = \begin{matrix} 1 \\ 0 \end{matrix} [\langle \text{label} \rangle \{ \} | :] \langle \text{unlabelled statement} \rangle$$

$$\begin{aligned} \langle \text{unlabelled statement} \rangle :: = & \langle \text{arithmetic statement} \rangle \mid \langle \text{program definition statement} \rangle \mid \\ & \langle \text{program control statement} \rangle \mid \langle \text{geometrical definition statement} \rangle \mid \\ & \langle \text{geometrical execution statement} \rangle \end{aligned}$$

4.14 Nesting

4.14.1 Semantics

Any unlabelled arithmetic statement or unlabelled geometrical definition statement may be nested.

The statement is written between the opening and closing parentheses. Wherever a simple identifier may appear, it is possible to nest the full statement with a simple identifier which may be referenced later in the part program, or to nest the statement without a simple identifier or equals character. The three alternatives

- a) simple identifier,
- b) nested statement without simple identifier,
- c) nested statement with simple identifier,

are included in the term "specifier". For example, where an entity of the type "circle" is valid, the term "circle specifier" is used.

4.14.2 Example

The statements

```
PT4 = POINT/3,6
PT6 = POINT/8,9
L2 = LINE/PT4, PT6
```

may be written as

```
L2 = LINE/(PT4 = POINT/3,6), (POINT/8,9)
```

4.14.3 Limitations

None.

4.15 Part program

4.15.1 Semantics

A part program is a logically complete and ordered sequence of statements and the result of processing a part program is to produce a correspondingly complete and ordered machine control program.

A part program consists of a part program identification statement, a sequence of statements and a part program termination statement.

4.15.2 Limitations

None.

4.15.3 Syntax

$$\langle \text{part program} \rangle :: = \langle \text{part program identification statement} \rangle \overset{k}{\underset{0}{[}} \langle \text{statement} \rangle] \\ \langle \text{part program termination statement} \rangle$$

5 Arithmetic statements

5.1 General comments

5.1.1 General semantics

The arithmetic features of the language allow a wide range of algebraic expressions to be evaluated within a part program. In general, the operators used in part program arithmetic are those used in conventional algebra, the exception being the algebraic multiplication $\times$ which is replaced by $*$.

Conventional algebra allows a description to occupy more than one line. Since this is not conveniently converted into suitable computer input, this is not allowed in the part-program language.

For example :

$$A = \frac{1 + 2 + 3}{4}$$

allowable in algebra has to be converted to :

$$A = (1 + 2 + 3)/4$$

before use in the part-program language. For the same reason it is found necessary to introduce an exponential operator ($**$ or $\uparrow$) into the part-program language so that

$$A = 2^3$$

is written as

$$A = 2**3 \text{ or } A = 2 \uparrow 3$$

Throughout, normal mathematical limitations apply, for example in using $\text{SQRT}(A)$, A shall have a positive value or in using $\text{ASIN}(B)$, B shall have a value lying between ± 1 .

The character $=$ is used in arithmetic statements, but may not have the meaning "equals" which it has in algebra. In part program arithmetic this character should be read as "is replaced by" or "is given the value of". It is therefore permissible to write $A = A + 1$ which means that one is to be added to the value of A , and A now given the value of the result.

Normal algebraic operation priorities apply. For example, in the statement $A = 2 + 3*4$, the $*$ operation will be performed before the $+$ operation. Brackets can be used in arithmetic statements and as in algebra may alter the sequence in which operations are carried out. If, for example, the above statement is written as $A = (2 + 3)*4$ the $+$ operation is performed before the $*$ operation.

The implied multiplication feature of algebra is not allowed. For example $A = 5(B + 2)$ which is permissible in algebra shall be written as $A = 5*(B + 2)$.

The hierarchy of operational priorities is as follows :

- | | | |
|----------|---|-------------------|
| Priority | 1 | () |
| | 2 | $**$, $\uparrow$ |
| | 3 | $*$, $/$ |
| | 4 | $+$, $-$ |

In the examples given later in this clause, the numeric values shown are for illustration only, and may be approximations to the real values.

5.1.2 Sub-contents

For

- 1) arithmetic operators, see 5.2;
- 2) arithmetic functions, see 5.3;
- 3) the algebraic functions, see 5.3.2;
- 4) the trigonometrical functions, see 5.3.3;
- 5) the exponential functions, see 5.3.4;
- 6) the vector functions, see 5.3.5;
- 7) the miscellaneous functions, see 5.3.6.

5.1.3 Limitations

None.

5.1.4 Syntax of arithmetic

$\langle \text{arithmetic symbol} \rangle :: = \langle \text{identifier} \rangle$
 $\langle \text{adding operator} \rangle :: = + \mid -$
 $\langle \text{multiplying operator} \rangle :: = * \mid /$
 $\langle \text{exponential operator} \rangle :: = ** \mid \uparrow$
 $\langle \text{primary} \rangle :: = \langle \text{unsigned number} \rangle \mid \langle \text{arithmetic symbol} \rangle \mid$
 $\quad \langle \text{scalar function} \rangle \mid (\langle \text{arithmetic expression} \rangle)$
 $\langle \text{factor} \rangle :: = \langle \text{primary} \rangle \mid \langle \text{primary} \rangle \langle \text{exponential operator} \rangle \langle \text{primary} \rangle$
 $\langle \text{term} \rangle :: = \langle \text{factor} \rangle \mid \langle \text{term} \rangle \langle \text{multiplying operator} \rangle \langle \text{factor} \rangle$
 $\langle \text{arithmetic expression} \rangle :: = \langle \text{term} \rangle \mid \langle \text{adding operator} \rangle \langle \text{term} \rangle \mid$
 $\quad \langle \text{arithmetic expression} \rangle \langle \text{adding operator} \rangle \langle \text{term} \rangle$
 $\langle \text{scalar} \rangle :: = \langle \text{primary} \rangle \mid \langle \text{adding operator} \rangle \langle \text{primary} \rangle \mid$
 $\quad (\langle \text{arithmetic symbol} \rangle = \langle \text{scalar} \rangle)$
 $\langle \text{arithmetic statement} \rangle :: = \langle \text{arithmetic symbol} \rangle = \langle \text{scalar} \rangle$
 $\langle \text{algebraic function} \rangle :: = [\text{ABS} \mid \text{SQRT}] (\langle \text{scalar} \rangle)$
 $\langle \text{trigonometric function} \rangle :: = [\text{SIN} \mid \text{COS} \mid \text{TAN} \mid \text{ASIN} \mid \text{ACOS} \mid \text{ATAN}] (\langle \text{scalar} \rangle) \mid$
 $\quad \text{ANGL} (\langle \text{circle spec} \rangle , \langle \text{point spec} \rangle) \mid$
 $\quad \text{DIST} (\langle \text{point spec} \rangle , \langle \text{point spec} \rangle) \mid$
 $\langle \text{exponential function} \rangle :: = [\text{EXP} \mid \text{LOG}] (\langle \text{scalar} \rangle)$
 $\langle \text{vector function} \rangle :: = \text{LNTH} (\langle \text{vector spec} \rangle) \mid$
 $\quad \text{DOT} (\langle \text{vector spec} \rangle , \langle \text{vector spec} \rangle)$
 $\langle \text{miscellaneous functions} \rangle :: = \text{ANGL} (\langle \text{circle spec} \rangle , \langle \text{point spec} \rangle) \mid$
 $\quad \text{DIST} (\langle \text{point spec} \rangle , \langle \text{point spec} \rangle) \mid$
 $\quad \text{NUM} (\langle \text{pattern spec} \rangle)$
 $\langle \text{scalar function} \rangle :: = \langle \text{trigonometric function} \rangle \mid$
 $\quad \langle \text{algebraic function} \rangle \mid$
 $\quad \langle \text{exponential function} \rangle \mid$
 $\quad \langle \text{vector function} \rangle \mid$
 $\quad \langle \text{miscellaneous function} \rangle$

5.2 Arithmetic operators

5.2.1 Semantics

The adding operators $+$ and $-$ shall be used in either of the following ways :

- a) As a unary operator, in which case the operator denotes the sign which has to be given to the following term :
 - $+$ denotes that the following term is to be positive;
 - $-$ denotes that the following term is to be negative.
- b) As a binary operator, in which case the operator, preceded by an arithmetic expression and followed by a term, denotes how the term is to be added to the arithmetic expression :
 - $+$ denotes that the term is to be added to the arithmetic expression;
 - $-$ denotes that the term is to be subtracted from the arithmetic expression.

The multiplying operators $*$ and $/$ denote how the preceding term is to be multiplied by the following factor :

- $*$ the term is to be multiplied by the factor;
- $/$ the term is to be divided by the factor.

The exponential operator $**$ or $\uparrow$ denotes that the preceding primary is to be raised to a power given by the following primary.

5.2.2 Examples

$A = B + C$	A contains the sum of B and C
$A = B * C$	A contains the result of B multiplied by C
$A = B ** C$	A contains the result of B to the power C

5.2.3 Limitations

None.

5.2.4 Syntax

$\langle \text{adding operator} \rangle :: = + \mid -$

$\langle \text{multiplying operator} \rangle :: = * \mid /$

$\langle \text{exponential operator} \rangle :: = ** \mid \uparrow$

5.3 Arithmetic functions

5.3.1 General comments

NOTE — Throughout this clause the preferred form of function names has been used, i.e. SIN, ABS, etc. However, it is recognized that the older “F” forms, i.e. SINF, ABSF, etc., may, in some instances, remain in use for some time. The comments and observations made on the preferred forms apply equally well to the “F” versions.

5.3.1.1 Sub-contents

For

- 1) algebraic functions, see 5.3.2;
- 2) trigonometric functions, see 5.3.3;
- 3) exponential functions, see 5.3.4;
- 4) vector functions, see 5.3.5;
- 5) miscellaneous functions, see 5.3.6.

5.3.1.2 Syntax

< algebraic functions > : : = [ABS/SQRT] (< scalar >)

< trigonometric functions > : : = [SIN/COS/TAN/ASIN/ACOS/ATAN] (< scalar >)

< exponential functions > : : = [EXP/LOG] (< scalar >)

< vector functions > : : = LNTH (< vector spec >) |
DOT (< vector spec > , < vector spec >)

< miscellaneous functions > : : = ANGL (< circle spec > , < point spec >) |
DIST (< point spec > , < point spec >) |
NUM (< pattern spec >)

5.3.2 The algebraic functions; ABS and SQRT

5.3.2.1 Semantics

The algebraic function designator denotes the type of arithmetic process to be performed using the following bracketed arithmetic expression.

ABS : the absolute value of the arithmetic expression is to be taken.

SQRT : the square root of the arithmetic expression is to be taken.

5.3.2.2 Examples

$A = \text{ABS} (-2)$; A contains the absolute value of (-2) , i.e. 2

$A = \text{SQRT} (9)$; A contains the square root of (9) , i.e. 3

5.3.2.3 Limitations

None.

5.3.2.4 Syntax

$\langle \text{algebraic function} \rangle : : = [\text{ABS} \mid \text{SQRT}] (\langle \text{scalar} \rangle)$

5.3.3 Trigonometric functions; SIN, COS, TAN, ASIN, ACOS, ATAN

5.3.3.1 Semantics

The trigonometric function designator denotes the type of trigonometric process to be performed on the following bracketed arithmetic expression :

SIN : the sine of the following arithmetic expression is to be taken.

COS : the cosine of the following arithmetic expression is to be taken.

TAN : the tangent of the following arithmetic expression is to be taken.

ASIN : the angle whose sine is given by the following arithmetic expression is to be taken.

ACOS : the angle whose cosine is given by the following arithmetic expression is to be taken.

ATAN : the angle whose tangent is given by the following arithmetic expression is to be taken.

In the case of the direct functions, the value of the arithmetic expression will be taken to represent degrees.

In the case of the inverse functions, the result given will be in degrees in the following ranges :

ASIN – $90 < A < 90$

ACOS $0 < A < 180$

ATAN – $90 < A < 90$

5.3.3.2 Examples

$A = \text{SIN}(30)$	A is given the value 0,5
$A = \text{COS}(30)$	A is given the value 0,866
$A = \text{TAN}(45)$	A is given the value 1,000
$A = \text{ASIN}(0.5)$	A is given the value 30
$A = \text{ASIN}(-0.5)$	A is given the value – 30
$A = \text{ACOS}(0.866)$	A is given the value 30
$A = \text{ACOS}(-0.866)$	A is given the value 150
$A = \text{ATAN}(1)$	A is given the value 45
$A = \text{ATAN}(-1)$	A is given the value – 45

5.3.3.3 Limitations

None.

5.3.3.4 Syntax

< trigonometric function > : : = [SIN | COS | TAN | ASIN | ACOS | ATAN] (< scalar >)

5.3.4 The exponential functions; EXP and LOG

5.3.4.1 Semantics

The exponential function designator denotes the type of exponential process to be performed on the following bracketed arithmetic expression.

EXP : the value of e to a power given by the value of the following arithmetic expression is taken.

LOG : the natural logarithm of the following arithmetic expression is taken. In this case the expression must be positive.

5.3.4.2 Examples

$A = \text{EXP}(2);$ A contains the value of e^2 i.e. 7,389 1

$A = \text{LOG}(2);$ A contains the value of $\log_e 2$ i.e. 0,693 1

5.3.4.3 Limitations

None.

5.3.4.4 Syntax

$\langle \text{exponential function} \rangle : : = [\text{EXP} \mid \text{LOG}] (\langle \text{scalar} \rangle)$

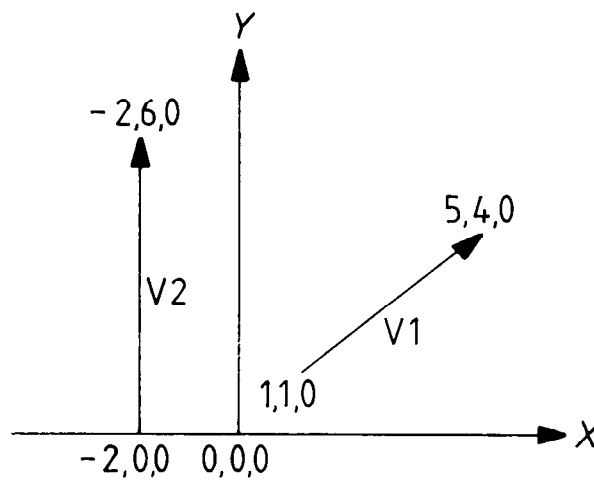
5.3.5 The vector functions; LNTH and DOT

5.3.5.1 Semantics

The LNTH vector function designator denotes that the length of the vector represented by the following bracketed vector specifier is to be calculated.

The DOT vector function designator denotes that the dot product of the vectors represented by the following bracketed pair of vector specifiers is to be obtained. In algebraic terms, the result is (length of first vector) $\times$ (length of second vector) $\times$ COS (angle between the vectors), or diagrammatically it is the product of the length of one vector and the length of projection of the other vector on it.

5.3.5.2 Example



V1 = VECTOR /1, 1, 0, 5, 4, 0

V2 = VECTOR/ - 2, 0, 0, - 2, 6, 0

A1 = LNTH (V1) A1 contains the value 5

A2 = LNTH (V2) A2 contains the value 6

A3 = DOT (V1, V2) A3 contains the value 18, i.e. $6 \times 5 \times 3/5$

Figure 1

5.3.5.3 Limitations

None.

5.3.5.4 Syntax

< vector function > : : = LNTH (< vector spec >) |
DOT (< vector spec > , < vector spec >)

5.3.6 The miscellaneous functions; ANGL, DIST, NUM

5.3.6.1 Semantics

The miscellaneous function designator denotes the type of process to be performed on the following bracketed miscellaneous function :

ANGL : the angle between the positive *X*-axis and the line joining the centre of the specified circle with the specified point.

DIST : the distance between the specified points.

NUM : the number of points in the specified pattern.

5.3.6.2 Limitations

None.

5.3.6.3 Syntax

< miscellaneous function > : : = ANGL (< circle spec > , < point spec >) |
DIST (< point spec > , < point spec >) |
NUM (< pattern spec >)

6 Program definition statements

6.1 General comments

6.1.1 General semantics

Part program statements are normally processed in a sequential manner but it is sometimes desirable to repeat a sequence of statements. Program definition statements provide this facility by allowing symbols to be redefined, synonyms to be defined for keywords and altering the normal sequential processing of the part program statements.

6.1.2 Sub-contents

For

- a) synonym statements, see 6.2;
- b) reservation for subscripting, see 6.3;
- c) definition of a macro, see 6.4.

6.1.3 Limitations

None.

6.1.4 Syntax

< program definition statement > :: = { < synonym declaration > < reserve declaration > < macro block > }

6.2 The synonym statement; SYN/

6.2.1 Semantics

The synonym statement is used in a part program where for some reason it is desirable to use an alternative form of spelling for a keyword.

The general form of the synonym-keyword pairs is

SYN/list of synonym-keyword pairs

The synonym given in a SYN statement will, where it appears subsequently in a part program, be taken to represent the keyword associated with it in the SYN/ statement and will be processed as such. Once a symbol has been defined as a synonym it cannot be redefined in any other context. A list of preferred synonyms is given in annex C.

The assignment of a synonym to a keyword does not preclude the subsequent use of a keyword as desired.

6.2.2 Examples

SYN/P, POINT, C, CIRCLE, TT, TANTO, L, LINE

this allows the statement

L1 = L/RIGHT,TT,C1,RIGHT,TT,C2

which is taken as

L1 = LINE/RIGHT,TANTO,C1,RIGHT,TANTO,C2

The following type of statement is not allowed because P was defined as synonym

P = POINT/0,1,2

6.2.3 Limitations

None.

6.2.4 Syntax

$$\langle \text{synonym parameter list} \rangle ::= \langle \text{simple identifier} \rangle, \langle \text{keyword} \rangle \mid \langle \text{synonym parameter list} \rangle, \langle \text{simple identifier} \rangle, \langle \text{keyword} \rangle$$
$$\langle \text{synonym declaration} \rangle ::= \text{SYN} / \langle \text{synonym parameter list} \rangle$$

6.3 Reservation for subscribing; RESERV/

6.3.1 Semantics

The RESERV statement is used to indicate that, within a part program, identifiers (by definition subscripted) may appear.

The general form of the statement is **RESERV/** followed by a string of couplets the first part of each being the identifier to be subscripted and the second being the largest value of the subscript which may be given to that identifier.

The RESERV statement containing a particular identifier shall precede any use of an identifier.

6.3.2 Example

RESERV/P1,10,P2,7,PAT1,12

this allows the statement

P1(8) = POINT/INTOF,L1,L2

but would not allow

P2(8) = POINT/INTOF,L1,L2

6.3.3 Limitations

In certain implementations, the types of identifier which can be subscripted may be restricted.

An identifier which is in RESERV shall normally not be used without a subscript.

6.3.4 Syntax

$$\langle \text{reserve parameter list} \rangle ::= \langle \text{simple identifier} \rangle, \langle \text{scalar} \rangle \mid \langle \text{reserve parameter list} \rangle, \langle \text{simple identifier} \rangle, \langle \text{scalar} \rangle$$
$$\langle \text{reserve declaration} \rangle ::= \text{RESERV} / \langle \text{reserve parameter list} \rangle$$

6.4 Definition of a macro; MACRO/

6.4.1 Semantics

A MACRO definition is used to specify a group of part program statements that can subsequently be executed in a part program. The definition allows an identifier to be attached to the group for reference by subsequent CALLs.

A statement of the form

- a) MAC1 = MACRO
- or b) MAC2 = MACRO/A,B,C
- or c) MAC3 = MACRO/A = 1.2, B = P1, C = 5
- or d) MAC4 = MACRO/A = 1.2, B, C = 5

denotes the start of a MACRO sequence of part program statements. The MACRO statement may contain a parameter list [as in b), c) or d)] any item of which may be given a normal value [as in c)]. Parameters may be used anywhere within the macro and are only allocated values when the macro is called. Normal values are used only in the absence of an actual value in the call.

The end of the macro sequence of part program statements is denoted by the statement

TERMAC

6.4.2 Examples

MAC1 = MACRO

GOTO/P1

GODLTA/0.5

GODLTA/ - .5

TERMAC

MAC2 = MACRO/A, B

GOTO/A

GODLTA/B

GODLTA/ - B

TERMAC

6.4.3 Limitations

The following part program statements may not appear within a MACRO definition :

- a) PARTNO
- b) FINI
- c) LOOPST
- d) LOOPND
- e) MACRO

In addition implementation restrictions may bar the use of SYN/ and RESERV/ within a macro.

6.4.4 Syntax

< normal macro item > : : = < keyword > | < simple identifier > | < unsigned number > | (< arithmetic expression >)

< macro item list > : : = < simple identifier > | < simple identifier > = < normal macro item > |
 < macro item list > , < simple identifier > | < macro item list > ,
 < simple identifier > = < normal macro item >

< macro assignment statement > : : = < simple identifier > = MACRO [/ < macro item list >]

< macro termination statement > :: = TERMAC

< macro block > :: = < macro assignment statement > $\overset{k}{\underset{0}{[}}$ < statement > < macro termination statement >

6.4.5 Cross-reference (see annex D)

PARTNO

FINI

LOOPST

LOOPND

MACRO

CALL

7 Program execution statements

7.1 Part program control

7.1.1 General semantics

With the exception of "part program identification statement" and "part program termination statement" which have special places in a part program, the other "program execution statements" are used where control over things like input, output or program flow are desirable. These statements are grouped together as "program control statements".

7.1.2 Sub-contents

For

- 1) part program identification and termination, see 7.2;
- 2) part program identification statement, see 7.2.1;
- 3) part program termination statement, see 7.2.2;
- 4) machine and no-postprocessing statements, see 7.3;
- 5) machine statement, see 7.3.1;
- 6) no-postprocessing statement, see 7.3.2;
- 7) input/output statements, see 7.4;
- 8) print statement, see 7.4.1;
- 9) punch statement, see 7.4.2;
- 10) read statement, see 7.4.3;
- 11) cutter location print statement, see 7.4.4;
- 12) titles statement, see 7.4.5;
- 13) loop start and end statements and transfer statements, see 7.5;
- 14) loop start and end statement, see 7.5.1;
- 15) unconditional transfer statement, see 7.5.2;
- 16) conditional transfer statement, see 7.5.3;
- 17) geometric transfer statement, see 7.5.4;
- 18) copy statement and index specification, see 7.6;
- 19) copy statement, see 7.6.1;
- 20) index specification, see 7.6.2;
- 21) macro execution statement, see 7.7;
- 22) remark statement, see 7.8;
- 23) postprocessing print statement, see 7.9.

7.1.3 Limitations

None.

7.1.4 Syntax

$$\begin{aligned} \text{< program execution statement >} : : = & \left\{ \begin{array}{l} \text{< part program identification statement >} | \\ \text{< part program termination statement >} | \text{< machine statement >} | \\ \text{< no-postprocessing statement >} | \text{< print statement >} | \\ \text{< punch statement >} | \text{< read statement >} | \text{< cutter location print statement >} | \\ \text{< titles statement >} | \text{< loop start and end statement >} | \\ \text{< unconditional transfer statement >} | \text{< conditional transfer statement >} | \\ \text{< geometric transfer statement >} | \text{< copy statement >} | \text{< index specification >} | \\ \text{< macro execution statement >} | \text{< remark statement >} | \\ \text{< postprocessing print statement >} \end{array} \right\} \end{aligned}$$

7.2 Part program identification and termination

7.2.1 Part program identification statement

PARTNO/literal character string

7.2.1.1 Semantics

The PARTNO statement is used for part program identification.

7.2.1.2 Example

PARTNO/'VALVE HOUSING, DRAWING NO 72445'

7.2.1.3 Limitations

None.

7.2.1.4 Syntax

< part program identification statement > : : = PARTNO/ < literal character string > ^u₀ [, < literal character string >]

7.2.2 Part program termination statement

FINI

7.2.2.1 Semantics

The FINI statement defines the physical end of a part program.

7.2.2.2 Limitations

None.

7.2.2.3 Syntax

< part program termination statement > : : = FINI

7.3 Machine and no-postprocessing statements

7.3.1 Machine statement

MACHIN/machine parameter list

7.3.1.1 Semantics

MACHIN statements allow specification of one or more postprocessors to be called. Each postprocessor is defined by a name and optionally an identifying number. Additional information to initialize postprocessor parameters may be given.

7.3.1.2 Context dependency

Postprocessor execution is inhibited if a NOPOST statement is present anywhere in the part program.

7.3.1.3 Example

MACHIN/EXPOST, 0

7.3.1.4 Limitations

None.

7.3.1.5 Syntax

< machine statement > : : = MACHIN/ < simple identifier > $\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}$ [, < postprocessor parameter list >]

7.3.1.6 Cross-reference (see annex D)

NOPOST

NOTE — The postprocessor parameter list is dependent on a particular postprocessor so that its formal syntax description is not contained within the reference language document.

7.3.2 No-postprocessing statement

NOPOST

7.3.2.1 Semantics

Postprocessing is inhibited even if one or more MACHIN statements are present.

7.3.2.2 Example

NOPOST

7.3.2.3 Limitations

None.

7.3.2.4 Syntax

< no postprocessing statement > : : = NOPOST

7.3.2.5 Cross-reference (see annex D)

MACHIN

7.4 Input/output statements

7.4.1 Print statement

```

PRINT/ON
PRINT/OFF
PRINT/0
PRINT/PAGE
PRINT/2, S1, . . . , Sn
PRINT/NONAME, S1, . . . , Sn
PRINT/3, S1, . . . , Sn
PRINT/3, ALL
PRINT/S1, . . . , Sn

```

7.4.1.1 Semantics

A PRINT statement causes internally stored information to be printed. PRINT/ON : this feature permits the programmer to obtain a printout of the canonical form immediately following the definition statement. The statement has this effect on all subsequent definitions, until countermanded by a PRINT/OFF statement.

A PRINT/0 or PRINT/PAGE statement causes the page to be ejected.

A PRINT/2, . . . or PRINT/NONAME, . . . statement causes printing of canonical form or scalar value of the quantities specified, without their identification.

A PRINT/. . . or PRINT/3, . . . statement causes printing of the canonical form or scalar value of the quantities specified, with their identification.

PRINT/ALL or PRINT/3, ALL; this feature has the same effect as the PRINT/. . . statement, except that it affects all available quantities.

The symbolic print type specification is preferred usage.

7.4.1.2 Example

```
PRINT/L1,C2,ANGL4,PT2
```

7.4.1.3 Limitations

None.

7.4.1.4 Syntax

< print statement > :: = PRINT/ < print parameter list >

< print parameter list > :: = [ON | OFF] | [0 | PAGE] | $\overset{1}{0}$ [NONAME | 2 | 3], < identifier > $\overset{u}{0}$ [, < identifier >] | [3,] ALL

7.4.2 Punch statement

PUNCH/ n , S_1 , . . . , S_n

PUNCH/ n , ALL

PUNCH/ON

PUNCH/OFF

PUNCH/ S_1 , . . . , S_n

PUNCH/ALL

7.4.2.1 Semantics

A PUNCH statement causes internally stored information to be punched. n is implementation dependent and controls the form of the output. ALL specifies that all of form n is to be punched. S_1 , . . . , S_n are the quantities to be punched.

ON causes punching to be started.

OFF stops all punching.

ALL causes all information to be punched.

7.4.2.2 Example

PUNCH/A,B,C

7.4.2.3 Limitations

None.

7.4.2.4 Syntax

< punch statement > : : = PUNCH/ < punch parameter list >

< punch parameter list > : : = [ON | OFF] $\big|_0^1$ [< digit >], < identifier > $\big|_0^u$ [, < identifier >] $\big|_0^1$ [< digit >], ALL

7.4.2.5 Cross-reference (see annex D)

READ

7.4.3 Read statement

READ/1

READ/ALL

READ/1, $S_1, \dots, S_n$

7.4.3.1 Semantics

READ/1 instructs the computer program to read cards in the same code and format as produced by the PUNCH statement.

READ/1, $S_1, S_2, S_3, \dots, S_n$ instructs the computer program to read the cards and to assign a new symbol to each canonical form or scalar value on the cards.

S_1, S_2, S_3 , etc., are the new symbols.

7.4.3.2 Context dependency

The symbols S_1, S_2, S_3 , etc., are the symbolic names assigned to the canonical forms or scalar values read from the cards. The cards are in the same code and format as produced by the PUNCH statement.

7.4.3.3 Limitations

With the READ/ALL or READ/1 without new symbols there shall be an implementation dependent read termination.

7.4.3.4 Syntax

< read statement > :: = READ/ < read parameter list >

< read parameter list > :: = [1 | ALL] | 1, < identifier > $\overset{u}{0}$ [, < identifier >]

7.4.3.5 Cross-reference (see annex D)

PUNCH

7.4.4 Cutter location print statement

CLPRNT

CLPRNT/ON

CLPRNT/OFF

CLPRNT/OMIT, n

7.4.4.1 Semantics

CLPRNT calls for a print of all cutter location data.

CLPRNT/ON starts printing of cutter location data.

CLPRNT/OFF stops printing of cutter location data.

CLPRNT/OMIT, n prints first point then points $n + 1$, $2n + 1$, $3n + 1$, . . . and last point.

7.4.4.2 Context dependency

The CLPRNT statement operates so that the information produced during processing in the computer is printed.

7.4.4.3 Limitations

None.

7.4.4.4 Syntax

< cutter location print statement > : : = CLPRNT [/ < clprint parameter list >]

< clprint parameter list > : : = [ON | OFF | OMIT, $\overset{u}{1}$ [< digit >]]

7.4.5 Titles statement

TITLES/literal character string

7.4.5.1 Semantics

The TITLES statement permits labelling of the printed output. The literal character string following TITLES in the statement is printed immediately at the time the statement is executed. The string starts at the left printing margin and extends across the paper up to the length of the string. The literal character string may be continued beyond a single input line. Successive strings are concatenated into a single string for printing.

7.4.5.2 Example

```
PRINT/PAGE
TITLES/'SCALARS S1, S2, S3, S4 ARE THE', $
'CALCULATED SPINDLE SPEEDS FOR STEEL'
PRINT/S1, S2, S3, S4
```

7.4.5.3 Limitations

If the literal character string is greater than the print width of the paper, the folding back of the necessary continuation line(s) is implementor dependent.

7.4.5.4 Syntax

< titles statement > : : = TITLES/ < literal character string > ^u₀ [, < literal character string >]

7.4.5.5 Cross-reference (see annex D)

PRINT

7.5 Loop start and end statements and transfer statements

7.5.1 Loop start and end statements

LOOPST

LOOPND

7.5.1.1 Semantics

LOOPST and LOOPND define the bounds (extent) of the group of statements which is a loop. The statements between LOOPST and LOOPND comprise the loop. LOOPST and LOOPND are implementation dependent.

Within a loop the normal sequential execution of the part program may be altered by transfer statements (IF, JUMPTO). Branching to LOOPND is permitted but not to LOOPST.

7.5.1.2 Context dependency

The MACRO statement implies LOOPST, and TERMAC implies LOOPND, i.e. a MACRO as a whole has by definition all properties of a loop.

7.5.1.3 Examples

```
J = 1
LOOPST
A)  . . .
    . . .
    IF (J - 100) B,C,C
B)  J = J + 1
    JUMPTO/A
C)  LOOPND
```

7.5.1.4 Limitations

LOOPST and LOOPND shall always be used as a pair.

7.5.1.5 Syntax

< start of a loop > : : = LOOPST

< end of a loop > : : = LOOPND

7.5.1.6 Cross-reference (see annex D)

IF
JUMPTO
MACRO
TERMAC

7.5.2 Unconditional transfer statement

JUMPTO/label

7.5.2.1 Semantics

Normal sequential execution of the part program is interrupted and control continues at the statement with the label referenced by the JUMPTO statement.

7.5.2.2 Examples

```

      LOOPST
      ...
A)    ...
      ...
      JUMPTO/A
      ...
      LOOPND
      LOOPST
      ...
A)    ...
      ...
      IF (VALUE - 2) A,B,C
      ...
C)    ...
      ...
B)    LOOPND

```

7.5.2.3 Limitations

If LOOPST and LOOPND are used, then an unconditional transfer command and the statement associated with the label referred to by this command shall occur within the same loop. In a macro, an unconditional transfer shall be to a statement within that macro.

7.5.2.4 Syntax

< unconditional transfer statement > : : = JUMPTO/ < label >

7.5.2.5 Cross-reference (see annex B)

Label

7.5.3 Conditional transfer statement

IF (*n*) label1, label2, label3

7.5.3.1 Semantics

Normal sequential execution of a part program may be altered by discrimination on the value of an arithmetic expression or single scalar quantity (*n*) relative to zero; control is transferred to the statement with the first label if the value is less than zero, to the second label if it is zero and to the third label if it is greater than zero. Branching forward and backward is permitted.

7.5.3.2 Limitations

See 7.5.2.

7.5.3.3 Syntax

< conditional transfer statement > :: = IF (< arithmetic expression >) < label > , < label > , < label >

7.5.3.4 Cross-reference

Label	}	(see annex B)
Arithmetic expression		
LOOPST	}	(see annex D)
LOOPND		
MACRO		
TERMAC		

7.5.4 Geometric transfer statement

TRANTO/label

7.5.4.1 Semantics

Control is transferred to the statement with the specified label.

7.5.4.2 Context dependency

The TRANTO statement is intended for use in conjunction with multiple check surface motion statements.

7.5.4.3 Examples

TRANTO/A

.....

.....

A)

7.5.4.4 Limitations

Only motion and post processor statements may be used as TRANTO aims.

7.5.4.5 Syntax

< geometric transfer statement > : : = TRANTO/ < label >

7.5.4.6 Cross-reference

Label (see annex B)

Multiple check surface (see 9.3)