Title:

Introduction to DOMAC Assembler.

A/S REGNECENTRALEN

RC SYSTEM LIBRARY: FALKONERALLE 1 DK-2000 COPENHAGEN F

RCSL No: 43-GL 5174
Edition: October 1977
Author: Harald Villemoes

Keywords: **Beginners guide, DOMUS, DOMAC, RC3600, assembler.**

Abstract:

This manual contains a short introduction to the RC3600 assembler language, a description of how to invoke the DOMAC assembler, and a list of possible error messages from the DOMAC assembler.

Copyright © A/S Regnecentrale 197

Printed by A/S Regnecentralen

Users of this manual are cautioned that the specifications contained herein are subject to change by RC at any time without notice. RC is not responsible for typographical or arithmetic errors which may appear in this manual and is not responsible for any damages

CONTENTS PAGE

References :

(I) Programmer's Reference Manual to RC3603 CPU

(II) DOMUS, User's Guide, Part I.

(III) DOMAC, Programmer's Reference Manual

(IV) MUSIL Codeprocedures

1. PREFACE

This manual is intended for persons who wants to get a basic knowledge of the RC3600 assembler language. After studying this manual carefully it should be possible for the reader to create minor assembler programs such as MUSIL codeprocedures or the like. This manual is not intended as a reference manual and all advanced facilities such as macro facilities, label generation, listing control etc. has been omitted. A description of these facilities can be found in the manual: DOMAC, Programmers Reference Manual.

Before studying this manual the reader should be familiar with:
Programmer's Reference Manual to RC3603 CPU, and
DOMUS, User's Guide, Part I.

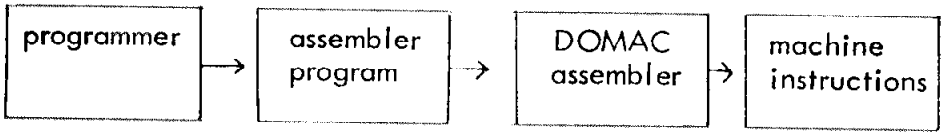
2. ASSEMBLER LANGUAGE

2.1 Assembly Process (General)

An RC3600 computer executes machine instructions which are numbers between 0 - 2^{16} (65536). To make a program means to create a sequence of numbers which is convenient for a computer but not for a human being who thinks in words. In order to ease the task of creating machine-level programs it is convenient to assign a symbolic name to each computer instruction, create a sequence of these symbolic names and then let the computer itself translate this sequence of words to a sequence of numbers.

This translation process is called an assembly and the program that makes the computer to do this is called an assembler. Finally an assembler language is a language consisting of these symbolic names for machine instructions.

When reading the following section, please consult the example in section 2.5.



The machine instruction 'load accumulator one with the contents of memory location 3' would be:

0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 in binary or
1 0 2 4 3 in decimal.

As input to the DOMAC assembler this would be:

```
LDA  1  3
```

In order to relieve the programmer from thinking of memory addresses as numbers, the DOMAC assembler facilitates symbolic addressing. This means that a name is assigned to a memory location and this location is then referenced by name rather than by its address.

```
TEMP:  7            ; this might be location 3
                    ; containing the value 7.
       LDA I TEMP   ; this would then be the
                    ; same instruction as above.
```

Whenever a semicolon is found on a line, the rest of this line is ignored by the assembler and may be used as comments.

It is convenient to think of an assembler program as a number of lines separated by carrage return or form feed. Each line consists of a maximum of 4 fields of which some may be optional depending on the context:

<label-field> <operation-field> <operator-field> <comments>

label-field:

This field is used to assign a symbolic name (a label) to a memory location.

A symbolic name consists of letters A-Z. Digits 0-9, period and question mark. The first character must not be a digit. A name may be any number of characters but only the first five will be used by the assembler for recognition, i.e. ABCDE and ABCDEFG is the same name seen from assembler point of view.

A label consists of a name followed by colon.

Examples:

```
TEMP:
LONGNAME:
```

operation field:

This field specifies the basic contents of a memory location. It may be the symbolic name of a machine instruction and will then in most cases be followed by one or more operators or it may be a value, either a constant or an expression. Finally it may be a pseudo-operation which is a directive to the assembler, e.g. .LOC (set the location counter to a specified value) or .END (end of assembler program). A list of all machine-instruction mnemonics is found in section 2.3 and a list of all pseudo-operations in section 2.4.

An expression is made of numbers, defined symbols, and operators. A number consists of one or more digits, which will be interpreted in current radix, by default radix 8. The input radix may be changed by the .RDX pseudo-operation. The following list shows the operators in the order they will be evaluated. When more operators of equal priority are shown on the same line they will be evaluated from left to right. The order of evaluation may, however, be changed by any number of parantheses; expressions inside parantheses are evaluated first.

Operator	Priority of evaluation
B	higest
+ - * / & !	next
< <= > >= == <>	lowest.

The bit alignment operator B evaluates according to the following formula:

$$\langle x \rangle \; B \; \langle y \rangle = \langle x \rangle * 2^{(15-\langle y \rangle)}$$

where <x> is a number of current radix or an expression enclosed in parantheses, and <y> is a number in radix 10 or if enclosed in paranthes-es an expression or number in current radix.

+	is addition,
-	is subtraction,
*	is multiplication,
/	is division,
&	is the logical and, evaluated bit by bit
!	is the logical or, evaluated bit by bit.

The following 6 operators evaluate to a value of either 1 (true) or 0 (false).

<	means	less than,
<=		less than or equal to,
>		greater than,
>=		greater than or equal to,
==		equal to,
<>		different from.

Finally it should be mensioned that when a number is immediately followed by a punktuation mark, it is interpreted in radix 10.

Examples:

Expression (number octal)	Value (in decimal)
2 + 3 * 4	20
12./ 2	6
(2 & 3) * (1 ! 3)	6
3 B 14	6
14>14	0
14>=14	1
3==3	1
2+3* 3B14	30

operator field:

The content of this field depends highly on the operation field. Usually it is empty or contains one or more expressions.

Examples:

```
LDA   1     TEMP   ; load AC1 with contents of TEMP
JMP         REP    ; GOTO REP
HALT               ; stop computer execution.
ISZ         REL,2  ; increment the location with the address =
                   ; REL + (contents of AC 2) and skip if
                   ; the result is zero.
```

2.2 Addressing and Relocatability

As mensioned above a name has a value, e.g. a label has the value: memory address. Actually a name also has another characteristica called the relocatability which describes how the value is to be transformed when it is loaded into the computer memory. The output from the domac assembler is in relocatable binary format. The following relocatabilities exist:

absolute
pace zero relocatable
pace zero byte relocatable
normal relocatable
normal byte relocatable

Both the address of a storage word and the contents of the storage word may be relocated during load.

The DOMAC assembler maintains three registers called location counters which contain the next page zero relocatable location, the next normal relocatable location, and the next absolute location. Whenever the contents of a memory location has been assembled, it is assigned the address contained in current location counter and normally the location counter is then incremented by one. Only one of the three location counters is current at any moment. The current location counter is chosen by the pseudo-operations:

.NREL, .ZREL, and .LOC

Absolute relocatability means no relocation, i.e. the value will be unchanged after a load process.

E.g.:

```
        7.      ; constants are absolute
        .NREL   ; normal relocatable location counter
A:      0       ; these three constants are
B:      1       ; all absolute.
C:      A-B     ;
```

Page zero relocation and normal relocation means that a constant (one for page zero and one for normal) will be added to the value during load, i.e. the value is moved (relocated) in core.

E.g.:

```
        .NREL   ; if this program is loaded into core in location 1000
A:      B       ; and 1001, the location addressed with A will
B:      2       ; contain 1001 and B still 2.
        .END    ;
```

Page zero byte relocation and normal byte relocation just means that the appropriate constant is added twice instead of once.

One way of defining a symbol is to use it as a label. The symbol will then be assigned the value of the location counter.

Another way of assigning a value to a symbol is to equate it to a value,

<symbol> = <expression>

E.g.: ABC = 14.
ZERO = 0
FALSE = ZERO

Location counter value and relocation properties can easily be seen from the assembler listing which will be explained in the following:

Column	Contents
1 - 3	If no errors are detected in the input these columns contain a two-digit line number followed by a space.
4 - 8	The location counter, if relevant. Otherwise the column is left blank. The value is normally output in radix 8.
9	A one-character flag indicating the location counter relocatability: space absolute - page zero relocatable = page zero byte relocatable ' normal relocatable '' normal byte relocatable
10 - 15	The data field, if relevant, or blank. The data is normally output in radix 8.
16	A one-character flag indicating the data field relocatability: space absolute - page zero relocatable = page zero byte relocatable ' normal relocatable '' normal byte relocatable $ external displacement data

Column	Contents
17 -	The source line as written.

Normally the source program will be followed by an alphabetically sorted cross-reference listing of all symbols defined in the assembler program in the following format:

Column	Contents
1 - 5	Name of the symbol
7 - 12	Value of the symbol, normally in radix 8.
13	Relocation property of the value as defined for column 16 of the listing.
15 - 17	The type of the symbol as follows:

space	user symbol
EN	entry defined by .ENT
PS	semipermanent symbol, defined by .DUSR or the like.
XD	external displacement, defined by .EXTD or .EXTU
XN	external normal, defined by .EXTN

2.3 Machine Instructions

The DOMAC assembler maintains a number of pre-defined (semi-permanent) symbols in a disc file called DOMPS. These symbols are the names of machine instructions and modifications, and names from the MUS monitor. A complete list of all pre-defined symbols is found in section 5.

The following list contains all machine instructions recognized by DOMAC (I) and the corresponding octal value:

Memory reference instructions without accumulator:

JMP	000000
JSR	004000
ISZ	010000
DSZ	014000

Memory reference instructions with accumulator:

LDA	020000
STA	040000

Arithmetic/logic instructions:

COM	100000
NEG	100400
MOV	101000
INC	101400
ADC	102000
SUB	102400
ADD	103000
AND	103400

and skip modifications to above ALU instructions:

SKP	1
SZC	2
SNC	3
SZR	4
SNR	5
SEZ	6
SBN	7

I/O instructions:

NIO	060000
DIA	060400
DOA	061000
DIB	061400
DOB	062000
DIC	062400
DOC	063000

I/O skip instructions:

SKPBN	063400
SKPBZ	063500
SKPDN	063600
SKPDZ	063700

Special instructions:

INTEN	060177
INTDS	060277
READS	060477
INTA	061477
MSKO	062077
IORST	062677
HALT	063077

Two characters have special meaning as input to the assembler:

@ The 'at' sign found in a source line indicates indirect addressing. When found in a source line containing a memory reference instruction, bit 5 of the instruction (the indirect bit) will be set to one. When found in a source line containing data, bit 0 will be set to one.

Example:

LDA 1 3	will assemble to 024003
LDA @ 1 3	will assemble to 026003
1 + 2	will assemble to 000003
@ 1 + 2	will assemble to 100003

\# A 'number' sign found in a source line containing an arithmetic/logic instruction indicates no-load and will set bit 12 of the assembled instruction to one.

Example:

AND 1, 2 SNR	will be 133405
AND # 1, 2 SNR	will be 133415

2.4 Assembler Directives

This section contains an alphabetically list of the most common used assembler directions (pseudo-operations) of the DOMAC assembler:

An assembler directive may be used in two different ways:

oper) When found as the first name of a source line, it is interpreted as a pseudo-operation, this means that some kind of assembler action is taken.

value) When found anywhere else in a source line, it is interpreted as a variable having a value related to the function of the directive.

NB. Not all assembler directives are defined in both of the above mentioned ways.

```
.                       ; dot
```

Value only.

Represents the current value of the location counter. E.g.:

```
.LOC 10.                ; set location counter to 10.
.                       ; value 10 in location 10.
JMP  . + 2              ; skip next instruction.
```

```
.BLK <exp>              ; block
```

Oper only.

Define a block of storage with size = <exp> number of words. No values are inserted, only the current location counter is incremented. E.g.:

```
.LOC  10.               ; location 10.
.                       ; value = 10.
.BLK  5                 ; 5 words are allocated.
.                       ; value = 16.
```

```
.DO <exp>               ; do repeatedly
```

Oper only

Repeat the source code from the next line and up to the next .ENDC the value of <exp> number of times. E.g.:

```
.DO   2                 ;
0                       ; two zero words will be
.ENDC                   ; assembled.
.DO   1 == 2            ; <exp> is zero so this line is
  5                     ; skipped.
.ENDC                   ;
```

```
.END or .END <exp>; end of program
```

Oper only

This pseudo-operation indicates end of source-file and if the source-file is the last specified, end of assembly. The value of <exp> will be used as start address in the end-block of the relocatable binary output.

.ENDC ; end of condition

Oper only.

This pseudo-operation indicates end of conditional assembly. See .DO

.ENT <user symbol> ; entry

Oper only.

The <user symbol> will be defined globally as an entry. <user symbol> may then be referenced in other programs and linked together by the linkage editor.
Example:

Program A	Program B
.ENT COUNT	.EXTN COUNT
COUNT: 0	GCOUNT: COUNT
.END	.END

The address of COUNT in program A will be placed in program B in location GCOUNT when both programs are linkage edited together.

.EOT ; end of tape

Oper only.

.EOT indicates end of source file and is equivalent to .END without an expression.

.EXTD <user symbol> ; external displacement
.EXTN <user symbol> ; external normal

Oper only.

These pseudo operations define that <user symbol> is defined in another program. The difference between .EXTD and .EXTN is that an external displacement may be used as the displacement of a memory reference instruction, if the value at linkage edit-time does not exceed 8 bits, while an external normal

may have any 16 bits value at linkage edit-time.
Examples:

```
        .EXTD       DISP1
        .EXTN       GLOB
        ISZ         DISP1, 2
        JMP    ~    .GLOB
.GLOB:  GLOB
```

.LOC <expression> ; set location counter
.LOC   ; current location counter value.

Oper (first format) and Value (second format).

Used as operation this pseudo-operation selects current location counter according to the relocation properties of <expression> and assigns the value of <expression> to the selected location counter.

Used as value it is equivalent to the pseudo-operation: . (DOT).
Examples:

```
   .LOC 3          ; absolute location counter
                   ; set to 3
   .NREL           ;
A: 0               ;
   .LOC  A + 5     ; NREL location counter
                   ; set to A + 5.
   .LOC  .LOC      ; no operation!
```

.NREL ; normal relocatability

Oper and Value.

When used as operation this pseudo-operation selects the normal relocatable location counter as current location counter. The value of .NREL is the value of the normal relocatable location counter.

```
.RDX <expression>     ; select input radix
.RDX                  ; current input radix
```

Oper (first format) and Value (second format).

Used as operation the input radix is set to the decimal value <expression> which must be inside the range:

2 <= <expression> <= 20

Used as value it represents the current input radix.
Examples:

```
.RDX      10        ;
11                  ; value 11 decimal
.RDX      8         ;
11                  ; value 9 decimal
.RDX      2         ;
11                  ; value 3 decimal
```

```
.TITL <symbol>        ; define title
```

Oper only.

This pseudo-operation assigns the title <symbol> to the program. This title is used to identify the program.
Example:

```
.TITL  MYPGM
```

```
.TXT <del> <text> <del>  ; pack a text
```

Oper only.

This pseudo-operation is used to pack an ASCII text into a number of consecutive words. <del> is a one-character delimeter that must not occour inside the text. The text will be packed according to the text mode (see .TXTM) with at least one zero byte following the last character. <text> is a string of ASCII characters or angel bracket constructs. An angel bracket construct is a left angel (<) followed by an expression, whose seven bit value will

be stored, followed by a right angel (>).
Examples:

```
.TXT 'THIS IS A TEXT'
.TXT / TEXT FOLLOWED BY BELL AND CR<7> <13.> /
```

.TXTM <expression> ; set text mode
.TXTM

Oper (first format) and Value (second format).

When used as operation this pseudo-operation is used to change the packing of bytes using the .TXT pseudo-operation:

<expression>	Byte packing
zero	right before left
non-zero	left before right.

The value of .TXTM is the current text mode value.
Examples:

```
.TXTM 1              ; normal MUS-mode
.TXT 'NORMAL PACK.';
.TXTM 0              ;
.TXT 'SPECIAL PACK.';
.TXT 'ONMRLAP CA.K'; = 'NORMAL PACK.'
```

.XPNG ; expunge all symbols

Oper only.

This pseudo-operation removes all symbols except permanent from the DOMAC symbol table.

.ZREL ; page zero relocation location counter.

Oper and value.

When used as an operation this pseudo-operation selects the page zero relocation location counter as the current location counter.

The value of .ZREL is the current value of the page zero relocation location counter.

2.5 Example

2.5

This section contains the source text and the produced listing of a sample program, a LOGOR MUSIL code procedure. [IV]

Source text:

```
; PROCEDURE LOGOR(VAR    RESULT: INTEGER;
;                 CONST NEWBITS: INTEGER );


; RESULT:= RESULT OR NEWBITS;   16 BIT PARALLEL LOGICAL OR
;
```

```
        .TITL   LOGOR
        .RDX    10
        .NREL

ROR:                                    ; PROC LOGOR;
        JSR     @       MZSTART+2       ;  TAKEADDRESS(RESULT)
        STA       1     RSAVE           ;
        JSR     @       MZSTART+3       ;  TAKEVALUE(NEWBITS)
                                        ;
        LDA     @ 3     RSAVE           ;

; AC0: - AC1: NEWBITS AC2: CUR AC3: OLDBITS

        COM       1,1                   ;
        AND       1,3                   ;
        ADC       1,3                   ;  AC3:= AC1 OR AC3;
                                        ;
        STA     @ 3     RSAVE           ;
        LDA       2     CUR             ;
        JMP     @       MZSTART+1       ; END LOGOR;
                                        ;
RSAVE:  0                               ; ADDRESS OF PARAMTER 1

        .END    ROR                     ; ENTRY POINT ADDR
```

Produced Listing:

```
 0001 LOGOR DOMUS MACRO ASSEMBLER REV 01.00
01
02                ; PROCEDURE LOGOR(VAR    RESULT: INTEGER;
03                ;                  CONST NEWBITS: INTEGER );
04
05
06                ; RESULT:= RESULT OR NEWBITS;   16 BIT PARALLEL LOGICAL O
07                ;
08
                          .TITL   LOGOR
10          000012        .RDX    10
11                        .NREL
12
13                ROR:                                  ; PROC LOGOR;
14  00000'006236          JSR    @     MZSTART+2        ;  TAKEADDRESS(RESULT)
15  00001'044411          STA       1  RSAVE            ;
16  00002'006237          JSR    @     MZSTART+3        ;  TAKEVALUE(NEWBITS)
17                                                      ;
18  00003'036407          LDA    @ 3   RSAVE            ;
19
20                ; AC0: - AC1: NEWBITS AC2: CUR AC3: OLDBITS
21
22  00004'124000          COM       1,1                 ;
23  00005'137400          AND       1,3                 ;
24  00006'136000          ADC       1,3                 ;  AC3:= AC1 OR AC3;
25                                                      ;
26  00007'056403          STA    @ 3    RSAVE           ;
27  00010'030040          LDA       2   CUR             ;
28  00011'002235          JMP    @     MZSTART+1        ; END LOGOR;
29                                                      ;
30  00012'000000 RSAVE:   0                             ; ADDRESS OF PARAMTER 1
31
32                        .END    ROR                   ; ENTRY POINT ADDR

0000 SOURCE LINES IN ERROR


  0002 LOGOR

ROR    000000'       1/32
RSAVE  000012'       1/15    1/18    1/26    1/30
```

3. ASSEMBLY PROCESS 3.

3.1 Prerequisites 3.1

In order to execute the DOMAC assembler the following hardware and software requirements should be fullfilled:

RC3600 CPU with 32 K words
Console device
A direct access storage media
DOMUS Operating System
Magnetic tape unit or Paper Tape Reader.

3.2 Calling the DOMAC Assembler 3.2

DOMAC is invoked by a utility program load command to the DOMUS operating system S. [2]

Call:

DOMAC [MODE.<modespec>] [LIST. <listfile>] [BIN.<binfile>] !
! [LINES.<lines>] [PERM.<permfile>] [SYMB.<symbfile>] !
! [MACRO.<macrofile>] [XREF.<xreffile>] <sourcespec>

where

<modespec> is a name of max. five characters, where each character specifies a function of the DOMAC assembler:

char	function
A	Add referenced semi-permanent symbols to the cross reference listing.
O	Overwrite all listing suppression, i.e. list all source lines.
S	Skip pass 2 but create a new permanent symbol table with filename: <symbolfile> and with macro definition filename: <macrofile>.
X	Do not make a cross reference listing.

<listfile>	is the name of the file or entry to which the listing is output. If not specified, no listing is produced.
<binfile>	is the name of the file or entry to which the binary is output. If not specified, no binary is produced.
<lines>	is an integer specifying the number of lines per page. If more than 63 is specified, 63 is used. If not specified, a maximum of 60 lines per page will be produced.
<permfile>	is the name of the file from which the semi-permanent symbols are read. Default is DOMPS.
<symbolfile>	is the name of the file to be used as symboltable file. Default is DOMST.
<macrofile>	is the name of the file to which macro definition strings are written. Default is DOMMC.
<xreffile>	is the name of the file to which cross references are written. Default is DOMXF.
<sourcespec>	is the source file specification containing any number of filenames in the following format: <filename> ! <filename>/S ! <filename>/N where /S means skip this file on pass 2. /N means do not produce a listing of this source file. When more than one sourcefile is specified they are processed in the order net, i.e. left to right.

<symbolfile>, <macrofile>, and <xreffile> are normally internal workfiles only and will be deleted after the assembly.

N.B.: The filenames specified must differ from the keywords.

Default:

DOMAC MODE. <0> LIST. <0> BIN. <0> LINES.60 PERM.DOMPS !
! SYMB.DOMST MACRO.DOMMC XREF.DOMXF

Examples:

Assemble file TEXT producing binary in RLBIN and listing including cross reference on the lineprinter. The entry $LPT describing the lineprinter driver exists on the disc.

DOMAC BIN.RLBIN LIST.$LPT TEXT

Assemble files PARM and PROG producing binary in PROGR and listing of PROG only in PROGL. PARM contains parameters only, i.e. no corestore locations.

DOMAC BIN.PROGR LIST.PROGL PARM/S PROG

Create a permanent symbol table file named DOMPS and a macro definition file named DOMPM containing the basic instruction definitions and the MUS system definitions.

DOMAC MODE.S SYMB.DOMPS MACRO.DOMPM PERM.NIHIL BIPAR MUPAR

NIHIL is a non-existing file and BIPAR is the basic instruction parameter tape.

3.3 DOMAC Functional Description

In order to fully utilize the capabilities of DOMAC and to understand the different I/O error messages that may occur it is necessary to have a brief understanding of the internal operation of DOMAC.

DOMAC operates with two sets of symbol table files in addition to the source, listing, and binary files. These are:

Permanent symbol files consisting of:

DOMPS,	containing the semi-permanent symbols, and
DOMPM,	containing the semi-permanent macro definition strings.

User symbol files consisting of:

DOMST,	containing the semi-permanent symbols with the user symbols added,
DOMMC,	containing the semi-permanent macro definition strings with the user macro definition strings added, and
DOMXF,	containing the cross-references of all user symbols.

When the DOMAC assembler has been loaded, it first checks the parameters including a check of the existence of all source files. The source, list, and binary files are opened using CONNECFILE, i.e. the list and binary file must be non-existing or an entry prior to loading DOMAC.

Now DOMAC initializes the symbol word by copying DOMPS to DOMST, DOMPM to DOMMC and if xref is wanted, DOMXF is created. However, if DOMPS does not exist, a '.XPNG' is executed in order to create an empty symbol file. If DOMST, DOMMC, and DOMXF exists prior to loading DOMAC, they are deleted and re-created.

The assembly is executed in two passes of the source files. In pass 1 the symbol table is build and the syntax is checked, and in pass 2 the assembly of 16 bit binary words is made and listing including syntax error messages and relocatable binary is produced.

When the assembly is completed, the line:

nnnn SOURCE LINES IN ERROR

is output to the console and optionally to the listing.

The core-resident parts of DOMST, DOMMC, and DOMXF are written back to disc if an xref listing is to be produced or the mode is S.

If an xref listing is to be produced, an internal request is sent to the father of DOMAC containing the command: DOMXR <symbolfile>. If no xref listing is to be produced, an internal request is sent to the father of DOMAC containing an empty command, i.e. 'FINIS'.

4. DOMAC ERROR MESSAGES 4.

The DOMAC assembler may produce two kinds of error messages; the one letter error indications in the listing (or if no listing is produced, on the console) as a result of syntactical or semantical errors in the source input, and the fatal error messages from the DOMAC assembler itself.

The first category is explained in section 4.1 below and the second category in 4.2.

4.1 Syntax Errors 4.1

Syntax errors are indicated by one-letter codes in the first three columns of the listing line in which the error occurs. If no listing is produced, the line with error indication will be output to the console. Below is a list of all possible error codes with a short explanation and an example showing one or more source lines that would result in the error.

A Addressing Error

Indicates that a memory reference instruction references an address outside the addressing range.

E.g.:

```
        ISZ    TEMP  ; addressing error
        .LOC   .+256.;
TEMP:   0
```

B Bad Character

Indicates an illegal character appearing in the line, e.g.:

```
A%B: 1          ; percent is illegal in a label
```

C Macro Error

Indicates either an attempt to do a nested macro continuation or that more than 63 arguments are specified.

D Radix Error

Indicates either an attempt to change current radix to a value outside the legal range or an attempt to input a digit outside current radix.

E.g.:

```
        .RDX 40         ; radix error
        .RDX 2          ;
        23              ; radix error
```

E Equivalence Error

Indicates that an equivalence contains an undefined symbol on the right side of the equal sign.

E.g.:

```
        A=    B         ; B undefined results in equivalence error
```

F Format Error

Indicates an illegal format for the type of operation, i.e. too many or too few operands.

E.g.:

```
        MOV 1,2,3,4 ; format error
```

G Global Error

Indicates an external or entry symbol error.

E.g.:

```
        .ENT NIHIL  ; nihil  undefined
        .EXTN LOCAL; local is defined
    LOCAL: 0        ;
        .END        ;
```

I Input Parity Error

Indicates that a character with the value 26 (SUB) is found in source input. This character is listed as back slash (\).

E.g.:

```
        LDA  I T\MP ; parity error
```

K Conditional Error

Indicates a superflous .ENDC, i.e. a .ENDC without a .DO.

E.g.:

```
        .DO 14          ;
          :
        .ENDC           ; end do
        .ENDC           ; conditional error
```

L Location Counter Error

Indicates that a .LOC or .BLK expression is undefined or outside range.

E.g.:

```
        .LOC  -14       ; location counter error
        .LOC 30000.     ;
        .BLK 10000.     ; location counter error.
```

M Multiple Definition Error

Indicates that a symbol is defined more than once within a program.

E.g.:

```
        TWIN : 0        ; multiple definition error
        TWIN : 0        ; multiple definition error
```

N Number Error

Indicates that a number exceeds the legal range, i.e. integers must be less than 2^{16}.

E.g.:

```
        .RDX 10.        ;
        100000          ; number error
```

O Overflow Error

Indicates that an instruction operand exceeds the legal range. It may also indicate over- or underflow when using the pseudo-operations: .PUSH, .POP, and .TOP.

E.g.:

```
LDA  5,TEMP ; overflow error
MOVZL 1,2,10; overflow error
```

P Phase Error

Indicates that DOMAC has detected an unexpected difference between pass 1 and pass 2 of the assembly, normally a symbol with different value in pass 1 and pass 2.

E.g.:

```
.TITL A        .TITL B
.NREL          .NREL
0              PHASE: 1

.EOT           .END
```

If above two source files are assembled and the first file (A) is skipped on pass 2, then the label PHASE will cause phase error.

```
DOMAC  LIST.$LPT A/S  B
```

Q Questionable Line

Indicates improper use of # or @ or when a page zero relocatable value is used where an absolute value is expected.

E.g.:

```
    .ZREL
A:  0
    .NREL
    LDA 0  A,3   ; Q-error
    MOV  # 2,3   ; Q-error
```

R Relocation Error

Indicates that an expression does not evaluate to a legal relocation property or that the expression is a mix of page zero- and normal relocative symbols.

E.g.:

```
        .ZREL
  Z:    0
        .NREL
  N:    1
        N + N + N   ; relocation error
        N + Z       ; relocation error
```

U Undefined Symbol Error

Indicates that a symbol is used but not defined.

E.g.:

```
        .TITL  UNDEF
        JMP HOME    ; undefined error
        .END
```

V Assembler Label Error

Indicates an error in the usage of the pseudo-operation .GOTO

X Text Error

Indicates an error in an 'angel bracket' construction within a string.

E.g.:

```
        .TXT 'TEXT ERROR <2 + 3 + >'
```

4.2 I/O Error Messages

Section 3.3 describes the internal function of DOMAC which includes a comprehensive input/output activity, that may cause error situations to arise and some of these are fatal and will cause DOMAC to terminate and output an error message to the console.

In addition to the standard I/O error messages, the following error messages may be produced:

0270 *** INTERNAL ERROR: nnnnn

This message indicates that an internal error situation has been detected. It should, however, be nearly impossible to get this message. Please contact the RC3600 BASIS group.

0271 *** DOMAC BREAK, NO: nn
0272 *** INSUFFICIENT CORE
0273 *** PARAMETER ERROR

Before the assembly is started, DOMAC checks the existence of all specified source files and this message may indicate a non-existing source file as well as a syntactical error in the DOMUS load command.

0274 *** VIRTUAL CORE ERROR

DOMAC symbols are placed in a virtual core file on disc. Each segment of this file contains its own logical segment-number. When a segment has been read into core, the segmentnumber is checked and, if it is found to be erronous, assembly is terminated with this error message.

5. PREDEFINED SYMBOLS

5.

A number of symbols are predefined in the DOMAC assembler, permanent symbols and semi-permanent symbols. Permanent symbols cannot be redefined by the user.

5.1 Permanent Symbols

5.1

These symbols, which are also called pseudo-operations or assembler directives, are defined inside the assembler itself and are used as commands to the assembler. Below is a list of all permanent symbols, the ones marked with asterisk are described in this manual.

. *	.ARGC	.BLK *
.DALC	.DIAC	.DICD
.DIO	.DIOA	.DISD
.DMR	.DMRA	.DO *
.DUSR	.DXOP	.EJEC
.END *	.ENDC *	.ENT *
.EOT *	.EXTD *	.EXTN *
.EXTU	.GOTO	.IFE
.IFG	.IFL	.IFN
.LIST	.LOC *	.MACR
.MCAL	.MSG	.NOCO
.NOLO	.NOMA	.NREL *
.PASS	.POP	.PUSH
.RDX *	.RDXO	.TITL *
.TOP	.TXT *	.TXTE
.TXTF	.TXTM *	.TXTN
.TXTO	.XPNG *	.ZREL *

5.2 Semi-permanent Symbols

AC0	000017 PS	AC1	000020 PS	AC2	000021 PS
AC3	000022 PS	ADC	102000 PS	ADD	103000 PS
ADDRE	000026 PS	AFIRS	000065 PS	AND	103400 PS
AREAP	000064 PS	BINDE	006232 PS	BIT	000101 PS
BREAD	000016 PS	BREAK	006012 PS	BSIZE	000012 PS
BUF	000025 PS	BUFFE	000011 PS	CAC1S	000004 PS
CALL	006355 PS	CBUFF	000054 PS	CCONY	000115 PS
CCORO	000041 PS	CDELA	006334 PS	CDEVI	000050 PS
CDISC	000112 PS	CDUMP	000077 PS	CERAS	000111 PS
CEXIT	000001 PS	CHAIN	000002 PS	CHANG	006350 PS
CIDEN	177777 PS	CLATO	000002 PS	CLEAN	006011 PS
CLEAR	100166 PS	CLINT	000032 PS	CLOSE	006220 PS
COM	100000 PS	COMLI	000362 PS	COMNO	000363 PS
COMON	006354 PS	CONBY	006173 PS	CONVT	000031 PS
CORE	000361 PS	CORES	000070 PS	COROU	000017 PS
COUNT	000027 PS	CPASS	006345 PS	CPOSI	000113 PS
CPRIN	006341 PS	CPU	000077 PS	CREAT	006346 PS
CRESE	000116 PS	CRETU	000003 PS	CSEND	006364 PS
CTERM	000114 PS	CTEST	006340 PS	CTOP	006367 PS
CTOUT	006342 PS	CUDEX	000053 PS	CUR	000040 PS
CUR2	000112 PS	CWANS	006337 PS	DECBI	006233 PS
DELAY	000061 PS	DEVTA	000370 PS	DIA	060400 PS
DIB	061400 PS	DIC	062400 PS	DIV	073101 PS
DIVID	006177 PS	DOA	061000 PS	DOB	062000 PS
DOC	063000 PS	DSZ	014000 PS	EFIRS	000057 PS
EVENT	000007 PS	EXIT	000056 PS	FFIRS	000060 PS
FREES	006210 PS	FREQU	000066 PS	GETAD	006357 PS
GETBY	006174 PS	GETPO	006360 PS	GETRE	006200 PS
GOS	006000 PS	GOT	002000 PS	GOTO	006356 PS
HACTI	000043 PS	HALT	063077 PS	HANSW	000044 PS
HDELA	000045 PS	IEQ	102415 PS	IGR	102433 PS
ILS	102032 PS	INBLO	006205 PS	INC	101400 PS
INCHA	006207 PS	INE	102414 PS	ING	102432 PS
INITC	006352 PS	INL	102033 PS	INNAM	006223 PS
INTA	061477 PS	INTBR	000230 PS	INTDS	060277 PS
INTEN	060177 PS	INTGI	000226 PS	INTPR	006225 PS
IORST	062677 PS	ISZ	010000 PS	JMP	000000 PS

JSR	004000 PS	LATIM	000042 PS	LDA	020000 PS
LOOKU	006347 PS	M	000040 PS	MASK	000067 PS
MCORO	000052 PS	MESS0	000006 PS	MESS1	000007 PS
MESS2	000010 PS	MESS3	000011 PS	MONIT	000054 PS
MOV	101000 PS	MOVE	006224 PS	MSEM	000051 PS
MSKO	062077 PS	MUL	073301 PS	MULTI	006176 PS
MZSTA	000234 PS	NAME	000004 PS	NEG	100400 PS
NEXT	000000 PS	NEXTO	006164 PS	NIO	060000 PS
O	000025 PS	OP	000034 PS	OPEN	006221 PS
OPMAS	177776 PS	OUTCH	006212 PS	OUTEN	006214 PS
OUTNL	006213 PS	OUTOC	006216 PS	OUTSP	006211 PS
OUTTE	006215 PS	PC	000033 PS	PCWSI	000006 PS
PFIRS	000052 PS	POWIN	000076 PS	PREV	000001 PS
PRIOR	000015 PS	PROCE	000054 PS	PROG	000012 PS
PROGR	000071 PS	PSPEC	000000 PS	PSTAR	000001 PS
PSW	000023 PS	PUTBY	006175 PS	PUTRE	006201 PS
PWSIZ	000014 PS	R	000032 PS	READS	060477 PS
RECEI	000005 PS	RECHA	006015 PS	REMOV	006351 PS
RESER	000030 PS	RETUR	006165 PS	RTIME	000054 PS
RUNNI	000054 PS	SADDR	000002 PS	SAVE	000024 PS
SAVE1	000025 PS	SAVE2	000026 PS	SAVE3	000027 PS
SAVE4	000030 PS	SAVE5	000031 PS	SBLOC	000111 PS
SBN	000007 PS	SBUSY	000103 PS	SCOUN	000001 PS
SDATA	000112 PS	SDEV1	000104 PS	SDEV2	000105 PS
SDEV3	000106 PS	SDISC	000101 PS	SEARC	006010 PS
SEM	000114 PS	SENDA	006007 PS	SENDE	000004 PS
SENDM	006004 PS	SEOF	000110 PS	SEQ	102414 PS
SETCO	006172 PS	SETEN	006353 PS	SETIN	006170 PS
SETPO	006217 PS	SETRE	006171 PS	SEZ	000006 PS
SFIRS	000006 PS	SGR	102432 PS	SIGCH	006344 PS
SIGGE	006365 PS	SIGNA	006343 PS	SILLE	000107 PS
SIZE	000003 PS	SKP	000001 PS	SKPBN	063400 PS
SKPBZ	063500 PS	SKPDN	063600 PS	SKPDZ	063700 PS
SLS	102033 PS	SNC	000003 PS	SNE	102415 PS
SNEXT	000004 PS	SNG	102433 PS	SNL	102032 PS
SNR	000005 PS	SOFFL	000102 PS	SOPER	000000 PS
SPARI	000113 PS	SSIZE	000007 PS	SSPEC	000003 PS
SSTAT	000005 PS	STA	040000 PS	START	006014 PS

STATE	000013 PS	STIME	000117 PS	STOPP	006013 PS
SUB	102400 PS	SZC	000002 PS	SZR	000004 PS
TABLE	000045 PS	TIMER	000014 PS	TLENG	000036 PS
TOPDE	000464 PS	TOPTA	000046 PS	TRANS	006204 PS
TRECO	000047 PS	TRETU	000046 PS	WAIT	006002 PS
WAITA	006005 PS	WAITC	006336 PS	WAITE	006006 PS
WAITG	006366 PS	WAITI	006003 PS	WAITO	006167 PS
WAITS	006335 PS	WAITT	006202 PS	WAITZ	006222 PS
Z	000032 PS	Z0	000024 PS	W1	000025 PS
Z2	000026 PS	Z3	000027 PS	Z4	000030 PS
Z5	000031 PS	ZAUX	000006 PS	ZBLOC	000011 PS
ZBUFF	000013 PS	ZCONV	000012 PS	ZFILE	000010 PS
ZFIRS	000017 PS	ZFORM	000015 PS	ZGIVE	000007 PS
ZKIND	000005 PS	ZLENG	000016 PS	ZMASK	000006 PS
ZMODE	000004 PS	ZN	000041 PS	ZNAME	000000 PS
ZREM	000023 PS	ZSHAR	000022 PS	ZSIZE	000014 PS
ZTOP	000020 PS	ZUSED	000021 PS	.0	000055 PS
.1	000120 PS	.10	000126 PS	.1024	000106 PS
.12	000127 PS	.120	000141 PS	.127	000142 PS
.128	000111 PS	.13	000130 PS	.15	000131 PS
.16	000114 PS	.1638	000102 PS	.1B0	000101 PS
.1B1	000102 PS	.1B10	000113 PS	.1B11	000114 PS
.1B12	000115 PS	.1B13	000116 PS	.1B14	000117 PS
.1B15	000120 PS	.1B2	000103 PS	.1B3	000104 PS
.1B4	000105 PS	.1B5	000106 PS	.1B6	000107 PS
.1B7	000110 PS	.1B8	000111 PS	.1B9	000112 PS
.2	000117 PS	.2048	000105 PS	.24	000132 PS
.25	000133 PS	.255	000143 PS	.256	000110 PS
.3	000121 PS	.32	000113 PS	.3276	000101 PS
.4	000116 PS	.40	000134 PS	.4096	000104 PS
.48	000135 PS	.5	000122. PS	.512	000107 PS
.56	000136 PS	.6	000123 PS	.60	000137 PS
.63	000140 PS	.64	000112 PS	.7	000124 PS
.8	000115 PS	.8192	000103 PS	.9	000125 PS
.CLEA	002166 PS	.CLOS	002220 PS	.CONB	002173 PS
.CR	000130 PS	.DIVI	002177 PS	.EDOC	000124 PS
.EVEN	000124 PS	.FF	000127 PS	.FREE	002210 PS
.GETB	002174 PS	.GETR	002200 PS	.INBL	002205 PS
.INCH	002207 PS	.LF	000126 PS	.M16	000146 PS
.M256	000147 PS	.M3	000144 PS	.M4	000145 PS

.MESS	000123 PS	.MULT	002176 PS	.NAME	000116 PS
.NEXT	002164 PS	.NL	000126 PS	.OPEN	002221 PS
.OPER	000035 PS	.OUTB	002206 PS	.OUTC	002212 PS
.OUTE	002214 PS	.OUTN	002213 PS	.OUTO	002216 PS
.OUTS	002211 PS	.OUTT	002215 PS	.PUTB	002175 PS
.PUTR	002201 PS	.REPE	002203 PS	.RETU	002165 PS
.RTC	000127 PS	.SETC	002172 PS	.SETI	002170 PS
.SETP	002217 PS	.SETR	002171 PS	.SSIZ	000124 PS
.TRAN	002204 PS	.WAIT	002202 PS	.Z	000150 PS