Title:

Assembler Coded Subroutines (CALL-routines) in RC BASIC (RC3600/RC7000) Programmer's Guide

A/S REGNECENTRALEN

RC SYSTEM LIBRARY: FALKONERALLE 1 DK-2000 COPENHAGEN F

RCSL No: 43-GL 6678
Edition: April 1978
Author: Stig Møllgaard

Keywords:

RC3600, RC7000, RC BASIC, DOMAC, DOMUS, COPS, Assembler Routines.

Abstract:

This guide describes how to program assembler routines that can be called from RC BASIC programs.

Printed by A/S Regnecentralen, Copenhagen

CONTENTS PAGE

This page is intentionally left blank.

1. INTRODUCTION.

The RC BASIC System provides facilities which makes it possible for the user to program assembler-coded subroutines which can be called from a BASIC program.

An assembler-coded subroutine may be useful if, for instance, input/output to or from special devices (such as graphic displays or analog/digital equipment) has to be carried out fast, or if the user want to perform some kind of operation, which is not possible to perform directly from a BASIC-program.

The RC BASIC system is a multi-user system, where each user may be considered as a coroutine which is executing reentrant code. This means that each user must use its own data-areas, i.e. the code itself cannot contain data. To every coroutine corresponds a coroutine description (also called a user description). This user description contains information about the current state of the coroutine and it also contains a data area, which can be used in the assembler-coded subroutines (see section 2.6). The start of the description of the coroutine, which is runnning can at any time be found in a page-zero location, USER. This means that a location in the user description can be accessed like this:

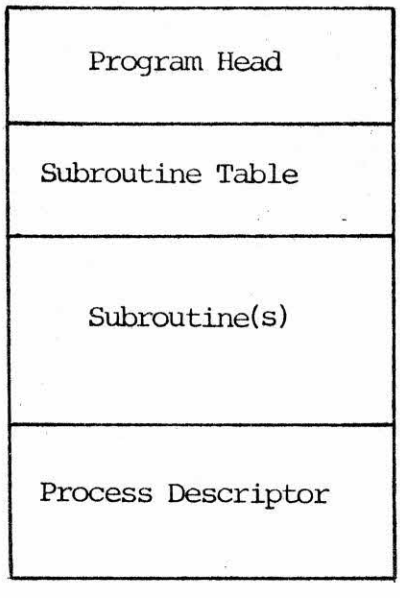
```
lda 3,  user      ; get start of description
lda 2, offset,3 ; get the word corresponding
                  ; to the value of 'offset'
```

It should be noticed, that the RC3600/RC7000 systems does not include any kind of memory-protection. This means that the programmer, who codes his own subroutines should be very careful. It also means, that Regnecentralen cannot take any kind of responsibility for system break-downs when user-coded subroutines are included in the system.

The user-coded subroutines must be coded as a seperate process (see ref. [1]) with the process name UCALL (see section 3). If Regnecentralen delivers subroutines, these will be coded as a process having the name RCALL.

2. CODING OF ASSEMBLY LANGUAGE SUBROUTINES.

A module containing one or more assembly language subroutines must look like this:



The program head and the process descriptor can be generated by means of two macroes defined in DOMAC, as described in chapter 3.

2.1 The Subroutine Table.

The subroutine table contains the names of the subroutines and the address of the first word of each subroutine.

The table is organized as follows:

```
address 1
name 1
address 2
name 2
  .
  .
  .
```

address 1, address 2 . . . address n are addresses referring to the first word of the first, second . . . n'th subroutine, respectively.
name 1, name 2 . . . name n are the names of the subroutines.

address n	Each name must fill exactly 4 words	2.1
name n	(8 bytes/characters).	
0	The names must be packed from left to right and padded with nulls (i.e. null-bytes).	
	The subroutine table must be terminated by a word containing a zero.	

Example:

```
        push            ; addr of PUSH-routine
        .txt .PUSH<0><0><0><0>.
                        ; name : PUSH

        pop             ; addr of POP-routine
        .txt  .POP<0> <0><0><0><0>.
                        ; name : POP

        0               ; terminate table with zero;
```

If the starting address of a subroutine is equal to -1, this means that the subroutine itself is not included. The name of the subroutine must, however, be placed in the subroutine table. This means that it is possible to program the subroutines in different modules, which then can be linked together into one relocatable binary module by means of the linkage editor, LINK. In the command to LINK, the first input-module must contain the program head and the subroutine table, and the last module must contain the process-descriptor. In the module containing the subroutine table, the starting address of the subroutines must be defined as an 'external normal' symbol (.EXTN). In the modules containing the subroutines, the starting address must be defined as an entry point (.ENT).

2.2 Parameter Handling.

A subroutine may have any number of formal parameters. For each subroutine the programmer must specify the number of parameters and for each parameter a type must be specified. These specifications must be placed as the very first words of each subroutine i.e. the address (in the subroutine table) refers to the first of these specification words.

The first word contains the number of parameters, and the next n words (where n is the number of parameters) describes the type of the parameters - one word for each parameter.

Example:

```
push  :  2              ; the PUSH-routine must be called with
                        ; two parameters
         array + real   ; type of first parameter
             real       ; type of second parameter

         sub 0,0        ; first instruction of the subroutine
```

2.2.1 Parameter Types.

As mentioned before each parameter must be type-specified.

The following types may be specified: REAL, REAL + REFERENCE, REAL + ARRAY, STRING, STRING + REFERENCE, STRING + ARRAY.

When a subroutine is called from RC BASIC the type of the actual parameters are compared with the type-specifications. In case of a conflict, the BASIC-program is interrupted and an errormessage is printed (see section 2.3).

The meaning of the different parameter types are: 2.2.1.

REAL : the actual parameter may be any numeric or relational expression (see ref. [2]).

REAL + REFERENCE : the actual parameter must be a numeric variable or a numeric array element.

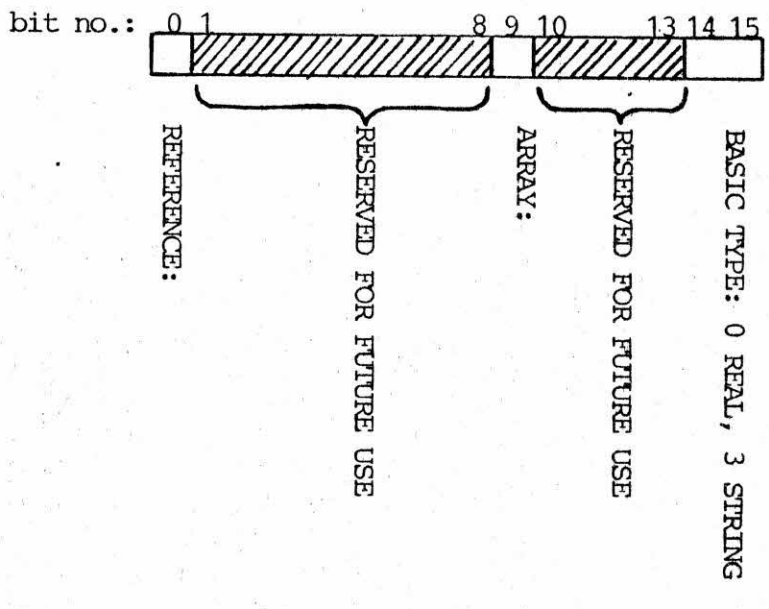
REAL + ARRAY : the actual parameter must be a numeric array.

STRING : the actual parameter may be any string expression (see ref. [2]).

STRING + REFERENCE : the actual parameter must be a string variable or a string array element.

STRING + ARRAY : The actual parameter must be a string array.

The descriptor words are build as follows:



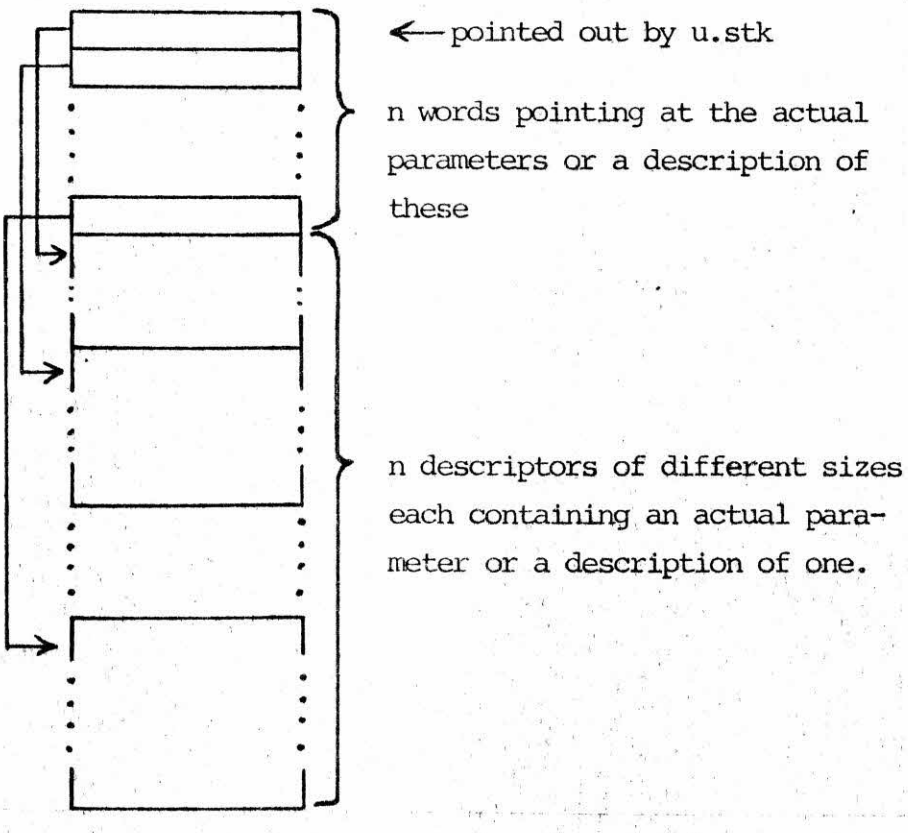
2.2.1. or REFERENCE = 1B0
ARRAY = 1B9
REAL = 0
STRING = 3

The symbols REAL, STRING, ARRAY and REFERENCE are symbols that are defined in the RC BASIC symbol tape, BAPAR (see ref. [3]).

2.2.2 Organization of Actual Parameters.

When a subroutine is called from a RC BASIC program, the actual parameters (or information about these) are passed to the subroutine in a core area pointed out by a word (U.STK) in the user description.

If the subroutine has n parameters the core area looks as follows:



The descriptors have different formats according to the type of the parameter as follows: (The program- and data-segments are described in section 2.5.2). 2.2.2.

REAL:

2 words containing the value of the actual parameter (floating point).

REAL + REFERENCE:

1 word containing the address of the first word of the variable (in the data-segment).

REAL + ARRAY:

3 words:

word 1: address of the first word in the first element of the array (in the data-segment).
word 2: number of rows in the array.
word 3: number of columns in the array.

STRING:

3 words:

word 1: address of the first byte of the string.
word 2: number of bytes in the string.
word 3: the number of the segment where the string is stored (0: program-segment, 1: data-segment).

STRING + REFERENCE:

3 words:

word 1: address of the first byte of the string variable (in the data-segment).
word 2: maximum number of bytes that can be hold in the string variable.
word 3: address of a word (in the data-segment) containing the actual (current) number of bytes in the string variable.

2.2.2. STRING + ARRAY:

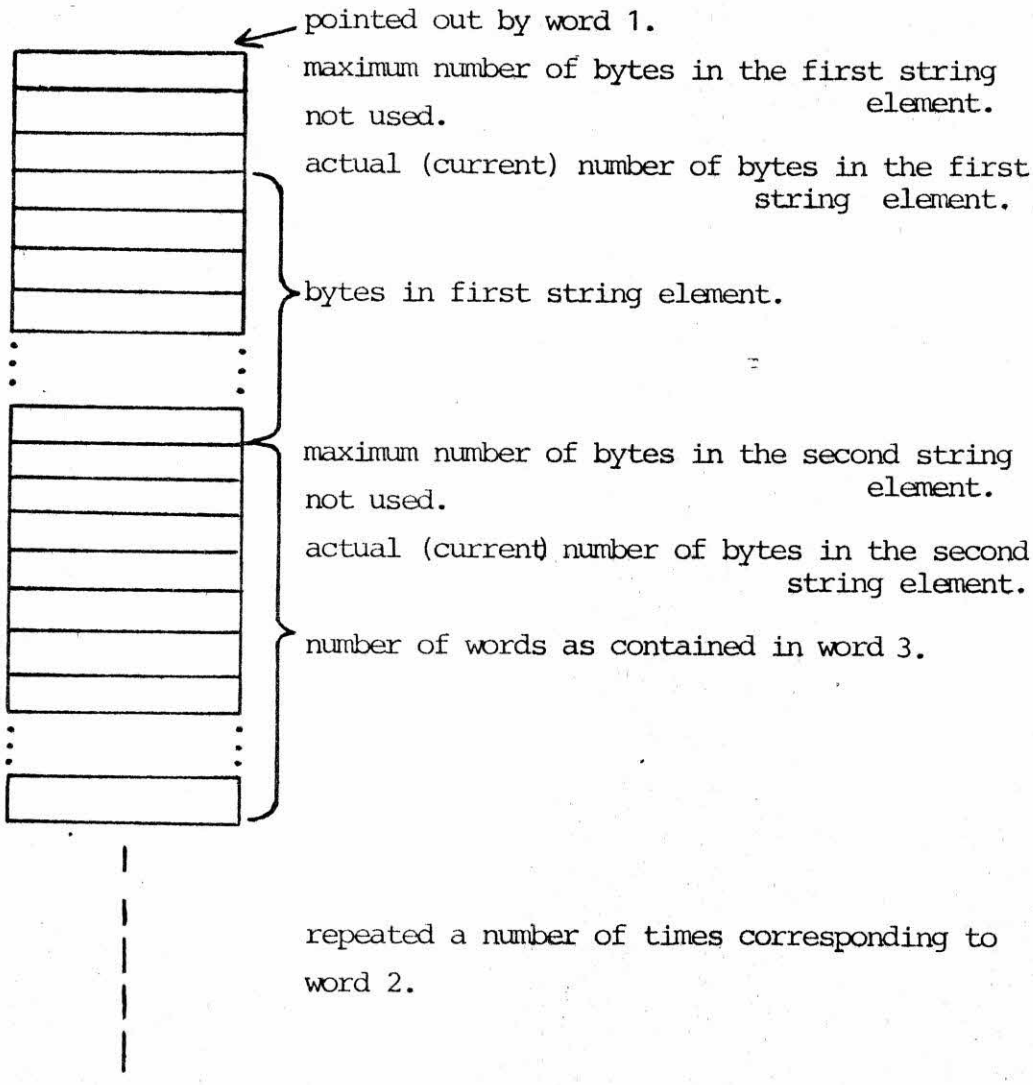
3 words:

word 1: address of futher description (in the data-segment).

word 2: number of elements in the string-array.

word 3: length of each element (in words).

Word 1 points to a part of the data-segment organized as follows:



The core area that contains the parameter descriptions is as mentioned pointed out by a word in the user description, U.STK. The user description is pointed out by a word, USER, in page-zero, so the first word of the core area can be loaded into accumulator 1 by the following sequence of instructions: 2.2.2.

```
lda 3, user       ;
lda 2, u.stk, 3   ;  AC1:= contents (user + u.stk)
lda 1,  0,2       ;
```

or

```
lda 3, user       ;
lda  @ 1, u.stk,3;
```

2.3 Calling a Subroutine from RC BASIC. 2.3

A subroutine may be called from a BASIC program in a statement with the following format:

CALL { <svar> / <slit> } [, { <var> / <svar> / <mvar> / <slit> / <expr> }] . . . .

Where the meaning of <svar>, <slit>, <var>, <mvar> and <expr> can be found in ref. [2].

Example:

```
CALL "PUSH", STACK, ELEM
or
NAME$ = "PUSH"
CALL NAME$, STACK, ELEM
```

2.3. When the CALL-statement is executed the following happens:

a. If a module containing user-coded subroutines is present in core, then the subroutine table in this module is searched for the name of the subroutine. If the name is found operation continues at point c.

b. As a. except that the searching is carried out in the module containing subroutines coded by RC. If the subroutine is not found then the BASIC program is interrupted with error no. 0046: PROCEDURE DOES NOT EXIST.

c. Now the number and the type of the actual parameters are checked against the parameter specifications in the subroutine (see section 2.2). If a conflict is found then the BASIC program is interrupted with error no. 0047: PARAMETER ERROR.

d. The actual parameters are organized as described in section 2.2.2 and then a jump is made to the word following immediately after the description of the formal parameters (see example in section 2.2).

When the subroutine is entered, the contents of the accumulators are as follows:

AC0 : undefined
AC1 : undefined
AC2 : USER. U.STK (points at the description of the actual parameters).
AC3: USER (points at the user description).

2.4 Return from a Subroutine. 2.4

Return from a subroutine can be carried out in three different ways depending on whether an error is detected or not.

2.4.1 Normal Return. 2.4.1

Normal return is made by means of the instruction RET1 (which is defined in the RC BASIC symbol tape, BAPAR (see ref. [3])).

The BASIC program will continue in the statement following the CALL-statement.

2.4.2 Return in case of an Error. 2.4.2

If some kind of error (not input/output errors) is detected in the subroutine the user might want to return the information about this error to the BASIC-program. This can be done by means of the two words

```
ERROR
<errno>
```

where <errno> is the number of the error (between 0 and 99) corresponding to the RC BASIC error messages.

The function of the ERROR-function is:

a. <errno> is stored in a word in the user-description.

b. a return is executed by means of the RETO-instruction (see sec. 2.7.).

When the return is executed, the BASIC-program will be interrupted (unless an ON ERR-statement has been executed) and the error-message will be output.

2.4.2. If one does not want to return to BASIC in case of an error but still wants to register the error (which can later be fetched by means of the BASIC-function SYS(7)), this can be done as shown in the following example:

```
        mov  0,0  szr   ;  if aco = o then
        jmp       lab1  ;
            execute     ;  execute error
            erfun       ; !see sec. 2.7 !
lab1:         .         ; !return from error !
              .
              .
              .
            ret1        ; !normal return to BASIC !
erfun:                  ; error:
            error       ;      error (31); ! SUBSCRIBT !
             31.        ; !return to lab1 !
```

The texts corresponding to error number 90 and 91 are

0090 : USER CALL ERROR 1
0091 : USER CALL ERROR 2

These can be used if none of the standard BASIC error messages fits the error situation.

2.4.3 Return in case of an Input/Output Error.

If an error occurs during an input/output operation this will imply that the input/output function used (see sec. 2.5.3) will return at (link + 0).

In this case the programmer must call the system function IOERR, which will set up the error code in the user description, set the word in the user description corresponding to the user file number (see sec. 2.5.3) to zero, close the zone in question and return by means of the RETO-instruction.

The IOERR-function is called by means of a macro, BCALL. As this macro contains two assembler-instructions, the call can not be placed immediately after the call of the input/output function. The following example shows how IOERR may be used: 2.4.3.

```
        lda 0  ---          ;  AC0 = zoneaddr
        lda 1  ---          ;  AC1 = character
        lda 2  cur          ;  AC2 = cur
            f.ochar         ;  f.outchar (zone, char),
        jmp    err05        ;  if error then goto err05
            .
            .
            .
            .

err05:  bcall ioerr         ;  execute ioerror,
                            ;  return to BASIC
```

2.5. System Functions Used in a Subroutine. 2.5.

2.5.1. Arithmetic Functions. 2.5.1.

If one wishes to perform arithmetic operations on numeric values, this can be done by means of routines included in the RC BASIC system. These routines may be called by means of a macro:

```
BCALL  <name>
```

where <name> is the name of the routine to be used. The macro BCALL will be assembled as <u>two</u> words

```
lda 3,  u.s21,3
jsr@    n,3
```

where the value of n depends on <name>.

2.5.1. U.s21 is a word in the user description pointing at a table, which contains entrypoints to the routines. The macro BCALL is defined in the RC BASIC Symbol tape, BAPAR, (see ref. [3]).

The functions that can be used are:

FIX: Convert a floating point number to a double-word integer.

	call	return
AC0	1. word of floating point number	result [0:15]
AC1	2. word of floating point number	result [16:31]
AC2	irrelevant	destroyed
AC3	user	user

call: BCALL FIX

After return, AC0[0] is the sign of the result: 0: positive, 1: negative.

FLOAT: Convert a double-word integer to floating point.

	call	return
AC0	integer [0:15]	result [0:15]
AC1	integer [16:31]	result [16:31]
AC2	irrelevant	destroyed
AC3	user	user

call: BCALL FLOAT

When called, AC0[0] is the sign of the integer.

In order to carry out floating-point arithmetic, the user may call four functions to add, subtract, multiply and divide, respectively.

2.5.1.

The functions all operate on 2 32-bit floating-point numbers, FN1 and FN2. When the functions are called, (AC0, AC1) should contain (FN2 [0:15], FN2 [16:31]) and AC2 must contain an address pointing at FN1. The exact conversions, which should be followed, are as follows:

Floating add: SUM : = FN2 + FN1

	call	return
AC0	FN2 [0:15]	SUM [0:15]
AC1	FN2 [16:31]	SUM [16:31]
AC2	addr of FN1	destroyed
AC3	user	user

call: BCALL FADD

Floating subtract: DIF : = FN2 - FN1

	call	return
AC0	FN2 [0:15]	DIF [0:15]
AC1	FN2 [16:31]	DIF [16:31]
AC2	addr of FN1	destroyed
AC3	user	user

call: BCALL FSUB

Floating multiply: PROD : = FN2 * FN1

	call	return
AC0	FN2 [0:15]	PROD [0:15]
AC1	FN2 [16:31]	PROD [16:31]
AC2	addr of FN1	destroyed
AC3	user	user

call: BCALL FMPY

2.5.1. Floating divide: QUOT : = FN1/FN2

	call	return
AC0	FN2 [0:15]	QUOT [0:15]
AC1	FN2 [16:31]	QUOT [16:31]
AC2	addr of FN1	destroyed
AC3	user	user

call: BCALL FDIV

If FN2 is zero then the return from FDIV is made by means of the RETO-instruction (error no. 16: ARITHMETIC ERROR). See section 2.7.

The three functions IMPY, IMPYA and IDIV operates on 2 or 3 16-bit integers (I1, I2 and I3). They should be used as follows:

Integer multiply: PROD = I1 x I2

	call	return
AC0	irrelevant	PROD [0:15]
AC1	I1	PROD [16:31]
AC2	I2	unchanged
AC3	user	user

call: BCALL IMPY

Integer multiply and add: RES = I1 x I2 + I3

	call	return
AC0	I3	RES [0:15]
AC1	I1	RES [16:31]
AC2	I2	unchanged
AC3	user	user

call: BCALL IMPYA

Integer divide: (QUOTIENT, REMINDER) : = I1 DIV I2 2.5.1.

	call	return
AC0	irrelevant	REMINDER
AC1	I1	QUOTIENT
AC2	I2	unchanged
AC3	user	user

call: BCALL IDIV

2.5.2. Fetch- and Store-functions. 2.5.2.

The running BASIC-program is stored in a so called virtual storage, which means that at any time only a small part of the BASIC-program will be present in the computers internal core while the rest will be placed on the disc.

Therefore, data belonging to the BASIC-program (such as actual parameters to subroutines) cannot be accessed by means of the LDA and STA instructions. If the user wants to access these data this can only be done by means of the system-functions

A.PBYTE, A.PWORD, A.PDOUBLE
A.GBYTE, A.GWORD, A.GDOUBLE

The virtual storage is divided into two segments: the program segment (no. 0) and the data segment (no. 1).

Usually the user will only have to access the datasegment, but when a string literal is an actual parameter, this will be placed in the program segment.

2.5.2. The functions should be used according to the following description.

a.gbyte: fetch one byte from (segment no, byteaddr)

	call	return
AC0	segment no	byte
AC1	byte addr	unchanged
AC2	cur	cur
AC3	irrelevant	user

call: a.gbyte

a.gword: fetch one word from (segment no., wordaddr)

	call	return
AC0	segment no	word
AC1	wordaddr	unchanged
AC2	cur	cur
AC3	irrelevant	user

call: a.gword

a.gdouble: fetch two words from (segment no., wordaddr) and (segment no., wordaddr + 1)

	call	return
AC0	segment no	word 1
AC1	wordaddr	word 2
AC2	cur	cur
AC3	irrelevant	user

call: a.gdouble

a.pbyte: store one byte at (segment no., byte addr) 2.5.2.

	call	return (at link + 1)
AC0	byte	unchanged
AC1	byteaddr	unchanged
AC2	cur	cur
AC3	irrelevant	user
Link + 0	segment no	destroyed

call: a.pbyte
segment no.

a.pword: store one word at (segment no., wordaddr)

	call	return (at link + 1)
AC0	word	unchanged
AC1	wordaddr	unchanged
AC2	cur	cur
AC3	irrelevant	user
Link + 0	segment no	destroyed

call: a.pword
segment no.

a.pdouble: store two words at (segment no., wordaddr) and (segment no., wordaddr + 1)

	call	return (at link + 2)
AC0	word 1	unchanged
AC1	word 2	unchanged
AC2	cur	cur
AC3	irrelevant	user
Link + 0	segment no	destroyed
Link + 1	wordaddr	destroyed

call: a.pdouble
segment no
wordaddr.

2.5.2. It should be noticed that

1) An attempt to store information outside the part of the storage belonging to the current user may cause a system break down.

2) A call of any of the fetch- and store-functions may provoke, that another user will be activated. Therefore, all subroutines that call these functions must be re-entrant.

In systems without a disc the same accessmethod must be used as the BASIC-programs are organized in the same way as in virtual-storage systems.

2.5.3. Input/Output Functions.

All input/output operations must take place via a zone (see ref. [1]). Before input or output can be carried out from or to a file, this file must be opened (i.e. a zone must be connected to the file). The opening of a file can only be done in a BASIC-program (by means of the OPEN FILE-statement). When an OPEN statement is executed, the address of the zone used will be stored in one of eight words in the user description. When an input/output function is used, this zoneaddr must be fetched before the function is called. The eight words in the user description corresponds to the eight user file-numbers that can be used in the BASIC program. The number(s) of the file(s) to be used in the subroutine must be given as parameters to the subroutine. The words corresponding to the 8 user filenumbers can be found in the user-description from U.UCH and on, as shown in the following example:

```
        .
        .
        .
        .
lda  3   user     ;  AC1 = filenumber
add  1,3          ;  (0<= ac1 <= 7)
lda  0   u.uch,3  ;  AC0:= USER.(U.UCH+FILENO)
```

The userdescription contains 3 addresses of "standardzones": PIO, CIN and COUT:

PIO (primary input/output) is the zone corresponding to the terminal

CIN (current input) is usually equal to pio, but may be changed. In BATCH-mode for instance, cin will be the zone corresponding to the card reader.

COUT (current output) is usually equal to pio, but may be changed. The RUNL-command for instance will set cout to the zone corresponding to the lineprinter.

The input/output functions all have two returning points. If an error occurs during the input/output operation, return is made to (link + 0).

In this case AC2[8:15] contains an error code corresponding to the RC BASIC error-messages with values larger than 100. AC2[1] is equal to one. In case of an input/output error the system function IOERR should be called as described in section 2.4.3.

2.5.3. The input/output functions should be used according to the following description.

f.ochar: output one character

	call	return (error) link + 0	return (ok) link + 1
AC0	zoneaddr	zoneaddr	zoneaddr
AC1	character	character	character
AC2	cur	errorcode	cur
AC3	irrelevant	user	user

call: f.ochar

f.otext: output a text

	call	return (error) link + 0	return (ok) link + 1
AC0	zoneaddr	zoneaddr	zoneaddr
AC1	byteaddr	byteaddr	byteaddr
AC2	cur	errorcode	cur
AC3	irrelevant	user	user

call: f.otext

The text must be terminated by a null-byte.

f.oblock: empty an output-buffer

	call	return (error) link + 0	return (ok) link + 1
AC0	zoneaddr	zoneaddr	zoneaddr
AC1	irrelevant	destroyed	destroyed
AC2	cur	errorcode	cur
AC3	irrelevant	user	user

call: f.oblock

f.ichar: input one character

	call	return (error) link + 0	return (ok) link + 1
AC0	zoneaddr	zoneaddr	zoneaddr
AC1	irrelevant	destroyed	character
AC2	cur	errorcode	cur
AC3	irrelevant	user	user

call: f.ichar

f.cheof: see if end of file has been reached

	call	return (true) link + 0	return (false) link + 1
AC0	zoneaddr	zoneaddr	zoneaddr
AC1	irrelevant	unchanged	unchanged
AC2	cur	cur	cur
AC3	irrelevant	user	user

call: f.cheof

return to link + 0 if end of file

return to link + 1 if not end of file

f.setpos: set position to a certain record number

	call	return (error)	return (ok)
AC0	zoneaddr	zoneaddr	zoneaddr
AC1	record no	record no	record no
AC2	cur	errorcode	cur
AC3	irrelevant	user	user

call: f.setpos

2.5.3. It should be noticed, that

a) If the user file has not been opened, the corresponding word in the userdescription will be equal to zero. If an input/output function is called with zoneaddr. equal to zero, this will cause a system-break-down.

b) Incorrect use of the input/output functions may cause system-break-down, and in certain cases data can be destroyed (on a secondary storage).

c) A call of any of the input/output functions may cause that another user will be activated. Therefore, all subroutines that call these functions must be reentrant.

2.6. Variables That Can Be Used.

As mentioned in section 1, the subroutines should as a main rule be reentrant. This is expecially important if a change of user can occur when the subroutine is executed. (A change of user may occur if any kind of input/output is performed or if the "fetch- and store functions" (section 2.5.2.) is used). In order to provide the possibility of coding reentrant subroutines, there must be a data-area for each user that might enter the subroutine. This data-area is a part of the user-description and therefore it must always be accessed relatively to the current value of USER.

21 consequtive words may be used:

USER.U.S00 - USER.U.S21 , for instance

```
lda  3,  user
lda  0,  U.S01,3
sta  2,  U.S18,3
```

2.7. Calling Local Procedures. 2.7.

As mentioned before, the subroutines must be reentrant. This means, that if a local procedure is used the return-address can not be saved locally. Consider the following example:

```
                          ;  start of call routine
            .             ;
            .             ;
A)    jsr     proc1       ;  first call of procedure
            .
            .
            .
B)    jsr     proc1       ;  second call of procedure
            ,
            ,             ;
          ret1            ;  return to BASIC

proc1:                    ;  start of procedure
      sta 3,  proc2       ;  save return address
            .
            .
            .
      jmp @   proc2       ;  return

proc2:      0             ;
```

If one user calls the procedure at B) then proc2 = B) + 1. Now if a change of user occurs in the procedure, and the next user calls the procedure at A) then proc2 = A) + 1. When the first user returns from the procedure, he will return to A) + 1 instead of B) + 1.

In order to avoid this problem, another way of calling a local procedure has been implemented in the RC BASIC system. A procedure can be called by means of the instruction

```
EXECUTE
<procedure>
```

2.7. where <procedure> is the address of the actual procedure (i.e. proc1 in the example). The return-address is automatically stored in the actual user description by the system. Returning from the procedure can be carried out by means of one of the instructions

RET0, RET1, RET2

RET0: return to the first word after <procedure>
RET1: return to the second word after <procedure>
RET2: return to the third word after <procedure>

The example might now look like this:

```
        .           ; start of call-routine
        .
    execute         ;
    proc1           ;
    jmp     oct1    ; if ret0
    jmp     oct2    ; if ret1
        .           ; if ret2
        .
        .
    execute
    proc1
        .
        .
        .
proc1:  .           ; start of procedure
        .
        .
    mov  0,0  snr   ; if AC0 = 0 then
      ret0          ;    ret0    else
    inc# 0,0  snr   ; if AC0 = -1 then
      ret1          ;    ret1     else
      ret2          ;    ret2
```

The BASIC-system calls the user-coded subroutine by means of the EXECUTE-instruction. If return is made by means of RET0, this is interpreted as if an error has occured (i.e. the BASIC-program will be interrupted). Otherwise (RET1 or RET2) execution of the BASIC-program continues after the CALL-statement. 2.7

3. SURROUNDINGS OF THE SUBROUTINE.

The user-coded subroutines must be included in a MUS-process (see ref. [1]). This means, that the module containing the subroutines must be started with a programhead and concluded with a process-descriptor. The RC BASIC symbol tape, BAPAR (see ref. [3]) contains two macro-definitions which, when used, will make DOMAC assemble a program-head and a process descriptor respectively.

Besides the program-head, the macro PRDE1 also defines the following:

```
.title      UCA01
.nrel           ; relocatable binary output from DOMAC
.rdx   10       ; radix 10
.txtm  1        ; packed from left to right
.txtn  1        ; no null-bytes if even number
                ; of bytes
```

Furthermore the PRDE1 macro contains two instructions which will make the process stop when it is loaded.

The first word after the macro PRDE1 must be the first word of the subroutine-table (see section 2.1.).

The macro PRDE2 defines a process-descriptor which must be placed after the subroutines

```
ex.:  PRDE1         ; program head
      .             ; subroutine table
      .             ; and subroutines
      .
      PRDE2         ; process-descriptor and
                    ; .end-operation.
```

The name of the defined process is UCALL.

Appendix B contains an example showing a subroutine-sourcetext and a listing produced by DOMAC.

4. THE ASSEMBLY AND LOADING OF THE SUBROUTINES. 4.

When the programmer has prepared the module containing the source-text of the subroutines(s), this module can be assembled using the DOMAC-macro-assembler. Before doing this, the user must be sure, that the semi-permanent symbols and macroes defined in BAPAR (see ref. [3]) are 'known' by DOMAC.

The command
DOMAC BIN.BCALL LIST.$LPT ACALL
will assemble the sourcetext in ACALL. A listing will be produced on the lineprinter and the relocatable binary output will be stored in the file BCALL.

For further information about DOMAC, please see ref. [4] and ref. [5].

When the subroutines have been assembled, they can be loaded (in a moving-head-disc system) by means of the command LOAD BCALL.

The subroutines must always be loaded before the RC BASIC-interpreter (COPS). In a processor-expansion system the subroutines must be loaded in the same cpu as COPS.

The process-name of the module containing user coded subroutines is UCALL, i.e. the routines can be removed by means of the KILL UCALL-command. If a system contains subroutines coded by Regnecentralen, then the processname of these is RCALL.

If the subroutines should be included in a floppy-disc-system, they must be linked together with the other modules contained in such a system. In the link-command, the module must be placed before COPS.

This page is intentionally left blank.

APPENDIX A - REFERENCES

[1] MUS-SYSTEM INTRODUCTION (I) and MUS PROGRAMMER's GUIDE (II).

Keywords: Multiprogramming, monitor, device handling, I/O-utility, record I/O, operator communication, operating system.

Abstract: (I) This manual is intended as an introduction guide to the Multiprogramming Utility System.

(II) The manual is mainly intended for readers who are going to use the system. The user is assumed to be familiar with the general principles of the system as well as with the assembler language.

[2] RC BASIC, Operating Guide.

Keywords: RC BASIC, DOMUS, Logical Disc.

Abstract: This manual describes how to use the RC BASIC system under the DOMUS operating system. The creation and use of logical discs is shortly described.

[3] BAPAR, RC BASIC Symbol Tape.

Keywords: DOMAC, COPS, RC BASIC, RC3600/RC7000.

Abstract: Definition of symbols used, when the COPS/RC BASIC system is assembled by DOMAC.

[4] Introduction to DOMAC Assembler.

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

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

[5] DOMAC, Programmer's Reference Manual.

APPENDIX B - EXAMPLES

The following pages shows an example of a module containing the two subroutines PUSH and POP.

The example illustrates the use of the macroes PRDE1, PRDE2 and BCALL. Also the use of same of the fetch- and store-functions, local procedures and the return mechanism is shown.

The module is shown in two 'versions':

1) The source text.

2) The listing produced by DOMAC, when the module is assembled.

B.1 EXAMPLE, SOURCE TEXT.

```
;

PRDE1    ; MACRO: PROGRAM HEAD
```

```
;
; CALLING SEQUENCES:
;
;       PUSH:   <STN> CALL "PUSH",<MVAR>,<EXPR>
;
;       POP:    <STN> CALL "POP",<MVAR>,<NVAR>
;
;       WHERE:
;               <STN>   IS A STATEMENT NUMBER.
;
;               <MVAR>  IS A NUMERIC ARRAY TO BE USED
;                       AS A STACK.
;
;               <EXPR>  IS A NUMERIC EXPRESSION TO BE
;                       PLACED ON TOP OF THE STACK.
;
;               <NVAR>  IS A NUMERIC VARIABLE OR
;                       A NUMERIC ARRAY ELEMENT TO
;                       RECEIVE THE VALUE ON TOP
;                       OF THE STACK.
;
;       THE FIRST ELEMENT OF <MVAR> MUST BE INITIALIZED
;       TO 0.
;
;       IF      0010 LOWBOUND=1
;               0020 DIM A(N)
;       THEN:
;               0100 CALL "PUSH",A,X+Y
;
;       CORRESPONDS TO
;
;               0100 LET A(1)=A(1)+1
;               0110 IF A(1)>N THEN STOP <* ERROR 31 *>
;               0120 LET A(A(1))=X+Y
;       AND
;               0200 CALL "POP",A,Z
;
;       CORRESPONDS TO
;
;               0200 IF A(1)=0 THEN STOP <* ERROR 31 *>
;               0210 LET Z=A(A(1)); A(1)=A(1)-1
;
```

```
                        ; SUBROUTINE TABLE
PUSH                    ; STARTING ADDRESS OF 'PUSH'
.TXT    "PUSH<0><0><0><0>"   ; NAME: 8 BYTES
POP                     ; STARTING ADDRESS OF 'POP'
.TXT    "POP<0><0><0><0><0>"   ; NAME: 8 BYTES
0                       ; TERMINATE TABLE WITH ZERO
;
;
;
;       UPON ENTRY TO PUSH THE COREAREA POINTED OUT BY
;       U.STK LOOKS AS FOLLOWS:
;
;               STACK + 0: X
;                     + 1: Y
;               X:    + 2: ADDRESS OF <MVAR>
;                     + 3: NUMBER OF ROWS
;                     + 4: NUMBER OF COLOUMNS
;               Y:    + 5: <EXPR> (FIRST WORD)
;                     + 6:        (SECOND WORD)
;
;
;       AND UPON ENTRY TO POP:
;
;               STACK + 0: X
;                     + 1: Y
;               X:    + 2: ADDRESS OF <MVAR>
;                     + 3: NUMBER OF ROWS
;                     + 4: NUMBER OF COLOUMNS
;               Y:    + 5: ADDRESS OF <NVAR>
;
```

```
PUSH:     2                    ;PROCEDURE PUSH
          ARRAY+REAL           ;( VAR A: ARRAY OF REAL;
          REAL                 ;       X: REAL);
          SUBZL  1,1           ;BEGIN
          EXECUTE              ;
          PSPOP                ;  ADJUST(1,ADDRESS);
          RETO                 ;  IF ERROR THEN RETURN0;
          STA    1   PSH01     ;
          LDA    0   +3,2      ;
          LDA    1   +4,2      ;
          LDA    2   CUR       ;
          A.PDOUBLE            ;  A(A(1)):=X
          1                    ;
PSH01:    0                    ;
          RET1                 ;END;

POP:      2                    ;PROCEDURE POP
          ARRAY+REAL           ;( VAR A: ARRAY OF REAL;
          REFERENCE+REAL       ;  VAR X: REAL);
          ADC    1,1           ;BEGIN
          EXECUTE              ;  ADJUST(-1,ADR);
          PSPOP                ;
          RETO                 ;  IF ERROR THEN RETURN0;
          INC    1,1           ;
          INC    1,1           ;
          SUBZL  0,0           ;
          LDA    2   CUR       ;
          A.GDOUBLE            ;  VALUE:=A(A(1)+1);
          LDA    3   U.STK,3   ;
          LDA@   3   +1,3      ;  ADDR:=ADDRESS(X);
          STA    3   POP01     ;
          A.PDOUBLE            ;  X:=VALUE
          1                    ;
POP01:    0                    ;
          RET1                 ;END;
```

```
PSPOP:  LDA     2   +0,2        ;PROCEDURE ADJUST(ADD,
                                ;                 ADDRESS);

        STA     1   U.S00,3     ;BEGIN
        LDA     1   +0,2        ;  S00:=ADD;
        SUBZL   0,0             ;
        LDA     2   CUR         ;
          A.GDOUBLE             ;  VALUE:=A(1);
        BCALL       FIX         ;  VALUE:=FIX(VALUE);
        LDA     0   U.S00,3     ;
        ADD     0,1             ;  VALUE:=VALUE+ADD;
        LDA@    2   U.STK,3     ;
        LDA     0   +1,2        ;
        SGE     1,0             ;  IF (VALUE>=A.01) OR
        MOVZL   1,0 SZC         ;     (VALUE<0) THEN
        JMP         ER31        ;        ERROR(31);
        LDA     2   +0,2        ;        ! INDEX ERROR !
        ADD     2,0             ;  ADDRESS:=A.ADR+VALUE*2;
        STA     0   U.S00,3     ;
        STA     2   PSP01       ;
        SUB     0,0             ;
        BCALL       FLOAT       ;  VALUE:=FLOT(VALUE);
        LDA     2   CUR         ;
          A.PDOUBLE             ;  A(1):=VALUE
          1                     ;
PSP01:    0                     ;
        LDA@    2   U.STK,3     ;
        LDA     1   U.S00,3     ;
          RET1                  ;END;

ER31:     ERROR                 ;ERROR:  SET ERRORCODE;
          31.                   ;        RETURN0;
```

```
PRDE2                   ; MACRO: PROCESS-DESCRIPTOR
```

B.2 EXAMPLE, DOMAC-LISTING.

```
 0001 UCA01 DOMUS MACRO ASSEMBLER REV 01.05
01                      ;
02
03                              PRDE1   ; MACRO: PROGRAM HEAD
04
                        .TITL   UCA01   ; USER-CODED SUBROUTINES   78.05.01
06                      .NREL
07              000012  .RDX     10      ; RADIX 10
08              000001  .TXTM    1       ; PACKED FROM LEFT TO RIGHT
09              000001  .TXTN    1       ; NO NULL-BYTES IF EVEN NUMBER OF BYTES
10
11                      PP00:                   ; PROGRAM START
12  00000'100001                1B0+1B15        ; DESCRIPTOR
13  00001'000007'               PP05            ; START
14  00002'000000                0               ; CHAIN
15  00003'000125                PP10-PP00       ; SIZE
16  00004'052503                .TXT    .UCALL. ; NAME
17              040514
18              046000
19
20                      PP05:                   ;
21  00007'006013                STOPPROCESS     ;
22  00010'000777                JMP     PP05    ;
23
```

```
!0002 UCA01
01                ;
02
03                ; CALLING SEQUENCES:
04                ;
05                ;       PUSH:    <STN> CALL "PUSH",<MVAR>,<EXPR>
06                ;
07                ;       POP:     <STN> CALL "POP",<MVAR>,<NVAR>
08                ;
09                ;       WHERE:
10                ;                <STN>    IS A STATEMENT NUMBER.
11                ;
12                ;                <MVAR>   IS A NUMERIC ARRAY TO BE USED
 3                ;                         AS A STACK.
 4                ;
15                ;                <EXPR>   IS A NUMERIC EXPRESSION TO BE
16                ;                         PLACED ON TOP OF THE STACK.
17                ;
18                ;                <NVAR>   IS A NUMERIC VARIABLE OR
19                ;                         A NUMERIC ARRAY ELEMENT TO
20                ;                         RECEIVE THE VALUE ON TOP
21                ;                         OF THE STACK.
22                ;
23                ;
24                ;       THE FIRST ELEMENT OF <MVAR> MUST BE INITIALIZED
25                ;       TO 0.
26                ;
27                ;
28                ;       IF       0010 LOWBOUND=1
29                ;                0020 DIM A(N)
30                ;       THEN:
31                ;                0100 CALL "PUSH",A,X+Y
 2                ;
 3                ;       CORRESPONDS TO
34                ;
35                ;                0100 LET A(1)=A(1)+1
36                ;                0110 IF A(1)>N THEN STOP <* ERROR 31 *>
37                ;                0120 LET A(A(1))=X+Y
38                ;       AND
39                ;                0200 CALL "POP",A,Z
40                ;
41                ;       CORRESPONDS TO
42                ;
43                ;                0200 IF A(1)=0 THEN STOP <* ERROR 31 *>
44                ;                0210 LET Z=A(A(1)); A(1)=A(1)-1
45                ;
```

```
!0003 UCA01
01                  ;
02
03                                      ; SUBROUTINE TABLE
04  00011'000024'           PUSH        ; STARTING ADDRESS OF 'PUSH'
05  00012'050125            .TXT    "PUSH<0><0><0><0>"  ; NAME: 8 BYTES
06          051510
07          000000
08          000000
09  00016'000043'           POP         ; STARTING ADDRESS OF 'POP'
10  00017'050117            .TXT    "POP<0><0><0><0><0>"  ; NAME: 8 BYTES
11          050000
12          000000
13          000000
14  00023'000000            0           ; TERMINATE TABLE WITH ZERO
15                  ;
16                  ;
17                  ;
18                  ;       UPON ENTRY TO PUSH THE COREAREA POINTED OUT BY
19                  ;       U.STK LOOKS AS FOLLOWS:
20                  ;
21                  ;               STACK + 0: X
22                  ;                     + 1: Y
23                  ;               X:    + 2: ADDRESS OF <MVAR>
24                  ;                     + 3: NUMBER OF ROWS
25                  ;                     + 4: NUMBER OF COLOUMNS
26                  ;               Y:    + 5: <EXPR> (FIRST WORD)
27                  ;                     + 6:        (SECOND WORD)
28                  ;
29                  ;
30                  ;       AND UPON ENTRY TO POP:
31                  ;
32                  ;               STACK + 0: X
33                  ;                     + 1: Y
34                  ;               X:    + 2: ADDRESS OF <MVAR>
35                  ;                     + 3: NUMBER OF ROWS
36                  ;                     + 4: NUMBER OF COLOUMNS
37                  ;               Y:    + 5: ADDRESS OF <NVAR>
38                  ;
```

```
!0004 UCA01
01                ;
02
03 00024'000002 PUSH:     2                    ;PROCEDURE PUSH
04 00025'000100            ARRAY+REAL           ;( VAR A: ARRAY OF REAL;
05 00026'000000            REAL                 ;       X: REAL);
06 00027'126520          SUBZL  1,1             ;BEGIN
07 00030'002240            EXECUTE              ;
08 00031'000066'           PSPOP                ;  ADJUST(1,ADDRESS);
09 00032'002241            RETO                 ;  IF ERROR THEN RETURNO;
10 00033'044406          STA    1   PSH01       ;
11 00034'021003          LDA    0   +3,2        ;
12 00035'025004          LDA    1   +4,2        ;
13 00036'030040          LDA    2   CUR         ;
14 00037'007105            A.PDOUBLE            ;  A(A(1)):=X
15 00040'000001            1                    ;
16 00041'000000 PSH01:     0                    ;
17 00042'002242            RET1                 ;END;
18
19 00043'000002 POP:       2                    ;PROCEDURE POP
20 00044'000100            ARRAY+REAL           ;( VAR A: ARRAY OF REAL;
21 00045'100000            REFERENCE+REAL       ;  VAR X: REAL);
22 00046'126000          ADC    1,1             ;BEGIN
23 00047'002240            EXECUTE              ;  ADJUST(-1,ADR);
24 00050'000066'           PSPOP                ;
25 00051'002241            RETO                 ;  IF ERROR THEN RETURNO;
26 00052'125400          INC    1,1             ;
27 00053'125400          INC    1,1             ;
28 00054'102520          SUBZL  0,0             ;
29 00055'030040          LDA    2   CUR         ;
30 00056'007102            A.GDOUBLE            ;  VALUE:=A(A(1)+1);
31 00057'035463          LDA    3   U.STK,3     ;
32 00060'037401          LDA@   3   +1,3        ;  ADDR:=ADDRESS(X);
33 00061'054403          STA    3   POP01       ;
34 00062'007105            A.PDOUBLE            ;  X:=VALUE
35 00063'000001            1                    ;
36 00064'000000 POP01:     0                    ;
37 00065'002242            RET1                 ;END;
```

```
!0005 UCA01
01                ;
02
03 00066'031000 PSPOP:  LDA     2   +0,2       ;PROCEDURE ADJUST(ADD,
04                                               ;              ADDRESS);
05
06 00067'045464          STA     1   U.S00,3    ;BEGIN
07 00070'025000          LDA     1   +0,2       ;  S00:=ADD;
08 00071'102520          SUBZL   0,0            ;
09 00072'030040          LDA     2   CUR        ;
10 00073'007102            A.GDOUBLE            ;  VALUE:=A(1);
11                       BCALL       FIX        ;  VALUE:=FIX(VALUE);
12 00074'035511          LDA     3   U.S21,3
13 00075'007400          JSR@        +0,3
14 00076'021464          LDA     0   U.S00,3    ;
15 00077'107000          ADD     0,1            ;  VALUE:=VALUE+ADD;
16 00100'033463          LDA@    2   U.STK,3    ;
17 00101'021001          LDA     0   +1,2       ;
18 00102'122032          SGE     1,0            ;  IF (VALUE>=A.D1) OR
19 00103'121122          MOVZL   1,0 SZC        ;     (VALUE<0) THEN
20 00104'000417          JMP         ER31       ;        ERROR(31);
21 00105'031000          LDA     2   +0,2       ;     =     ! INDEX ERROR !
22 00106'143000          ADD     2,0            ;  ADDRESS:=A.ADR+VALUE*2;
23 00107'041464          STA     0   U.S00,3    ;
24 00110'050407          STA     2   PSP01      ;
25 00111'102400          SUB     0,0            ;
26                       BCALL       FLOAT      ;  VALUE:=FLOT(VALUE);
27 00112'035511          LDA     3   U.S21,3
28 00113'007401          JSR@        +1,3
29 00114'030040          LDA     2   CUR        ;
30 00115'007105            A.PDOUBLE            ;  A(1):=VALUE
31 00116'000001            1                    ;
32 00117'000000 PSP01:     0                    ;
33 00120'033463          LDA@    2   U.STK,3    ;
34 00121'025464          LDA     1   U.S00,3    ;
35 00122'002242            RET1                 ;END;
36
37 00123'006244 ER31:      ERROR                ;ERROR:  SET ERRORCODE;
38 00124'000037            31.                  ;        RETURN;
```

```
!0006 UCA01
01                  ;
02
03                          PRDE2               ; MACRO: PROCESS-DESCRIPTOR
04
05                  PP10:                       ; PROCESSDESCRIPTOR:
06  00125'000000            0                   ; NEXT
07  00126'000000            0                   ; PREV
08  00127'000000            0                   ; CHAIN
09  00130'000025            PP15-PP10           ; SIZE
10  00131'052503            .TXT    .UCALL.     ; NAME
11          040514
12          046000
13  00134'000134'           .+0                 ; FIRST EVENT
14  00135'000134'           .-1                 ; LAST EVENT
15  00136'000000            0                   ; BUFFE
16  00137'000000'           PP00                ; PROGRAM
17  00140'000000            0                   ; STATE
18  00141'000000            0                   ; TIMER
19  00142'000001            1                   ; PRIORITY
20  00143'000007'           PP05                ; BREAK
21  00144'000125'           PP10                ; AC0
22  00145'000000            0                   ; AC1
23  00146'000125'           PP10                ; AC2
24  00147'000000            0                   ; AC3
25  00150'000016"           PP05*2              ; PSW
26  00151'000000            0                   ; SAVE
27
28                  PP15:                       ;
29                                      .END    PP10

0000 SOURCE LINES IN ERROR
```

```
  0007 UCA01

ALLAS 007106
ALLOC 007074
ALSIZ 000012
BCALL 000000   MC     5/11    5/26
CILAS 007137
CISIZ 000003
COMUS 007134
ER31  000123'         5/20    5/37
FADD  177775          5/12    5/14    5/27    5/29
FDIV  177772          5/12    5/14    5/27    5/29
FILAS 007130
FILER 007106
FISIZ 000022
FIX   177777          5/12    5/14    5/27
FLOAT 177776          5/12    5/14    5/27    5/29
FMPY  177773          5/12    5/14    5/27    5/29
FSUB  177774          5/12    5/14    5/27    5/29
IDIV  177767          5/12    5/14    5/27    5/29
IMPY  177771          5/12    5/14    5/27    5/29
IMPYA 177770          5/12    5/14    5/27    5/29
IOERR 177766          5/12    5/14    5/27    5/29
MAINC 007137
MCALL 007000
MCLAS 007150
MCSIZ 000011
POP   000043'         3/09    4/19
POP01 000064'         4/33    4/36
PP00  000000'         1/11    1/15    6/16
PP05  000007'         1/13    1/20    1/22    6/20    6/25
PP10  000125'         1/15    6/05    6/09    6/21    6/23    6/29
PP15  000152'         6/09    6/28
PRDE1 000211   MC     1/03
PRDE2 000276   MC     6/03
PSH01 000041'         4/10    4/16
PSP01 000117'         5/24    5/32
PSPOP 000066'         4/08    4/24    5/03
PUSH  000024'         3/04    4/03
TILAS 007134
TIMIN 007130
TISIZ 000004
```