RCSL No: 31-D674

Edition: April 1982

Author: Jesper Andreas Tågholt

Title:

ALGOL Coroutine System
User's Guide

A/S REGNECENTRALEN
af 1979

Keywords:

RC8000, ALGOL8, activity, coroutines, semaphore.

Abstract:

This manual describes a coroutine system available to ALGOL8 programs.

(52 printed pages)

Printed by A/S Regnecentralen af 1979, Copenhagen

RCSL 42-i1905

TABLE OF CONTENTS PAGE

TABLE OF CONTENTS (continued) PAGE

APPENDICES:

1. INTRODUCTION 1.

This manual describes a coroutine system available to ALGOL8 programs.

The manual implies a knowledge of the activity concept of ALGOL8.

Various concepts from the RC8000 monitor are used in the description. To avoid confusion the prefix '8000' is used with these concepts. For a more detailed description, plese see ref. [1] and ref. [3].

Chapter 2 contains a brief description of the facilities made available by the system.

Chapter 3 describes a special set of data structures used in the system.

Chapter 4 describes a central logic procedure of the system.

Chapter 5 describes the test facilities of the system.

Chapter 6 describes the standard procedures and standard variables of the system.

2. THE COROUTINE SYSTEM 2.

The system, which is based on message semaphores, is implemented as a set of external procedures with the following facilities:

1. Coroutines scheduled by priority.

2. Time out on wait operations.

3. Messages communicated in order of priority.

4. Messages picked out according to message lock/key.

5. Full index control of messages and semaphores.

6. Easy programming:
   a. pre-compiled modules, no source code copying,
   b. readymade central logic, scheduling coroutines, communicating messages,
   c. separate compilation of coroutines.

7. Switch to dedicated central logic.

8. Creation of test records.

2.1 The Central Logic Procedure Centralogic 2.1

The system has a standard procedure, centra_logic, which may be used as central logic for schedulation of coroutines and communication of messages.

The procedure has the following facilities:

1. Start up of coroutines, queuing them up on proper start semaphores.

2. Restart of the highest priority coroutine ready to be started

3. Scheduling coroutines de-activated by implicit passivate or "stack busy" (virtual coroutines inhibited by a fellow coroutine waiting for some i/o operations to complete).

4. Queuing up 8000-messages on central logic semaphores where it can be fetched by means of wait operations.

5. Queuing up messages on specified semaphores on the arrival of certain 8000-answers to the event queue.

6. Time out on wait operations without CPU load.

7. Display of logging records, showing the operation flow and time consumption.

8. Display of semaphore tables at termination.

2.2 Standard Procedures 2.2

1. Procedures for System Initialization

Name:	Task:
coroutines	make stack reservations for the semaphore and coroutine description tables and initialize the tables.
allocate	make stack reservation for a message buffer and initialize it.
initref	initialize reference variables (see section 3.4).
set_priority	change the priority of calling coroutine.

2. Communication Procedures

Name:	Task:
signal	send a message (placed in a message buffer) to a semaphore queue.
wait	wait for a message in/fetch a message from a semaphore queue.

3. Test Procedure

Name:	Task:
prepare_test	create a user test record and insert it among system test records.

4. Auxiliary Procedures

Name:	Task:
schedule	start the next coroutine.
cor_to_sem	transfer a coroutine description to a semaphore queue and insert it according to its priority.

The auxiliary procedures are intended to facilitate the construction of alternative coroutine schedulers, replacing centra_logic.

2.3 Standard Variables 2.3

1. Standard Variables for Modification of the Next Wait Operation

Name:	Meaning:
wait_select	if zero, the wait operation waits for/fetches the first message in the semaphore queue, no matter its key. If not zero, wait_select is a lock, and the wait operation waits for/fetches the first message in the semaphore queue with a key that fits the lock.

Name:	Meaning:
wait_time	states the maximum waiting time accepted by the coroutine executing the wait operation. If wait_time = 0 there is no limitation to the waiting time. If wait_time < 0 no waiting time is accepted. If wait_time > 0 wait_time contains the maximum waiting time measured in tenth of a second.

The contents of these variables are stored in the calling coroutine description each time wait is called and they are reset to zero.

2. Standard Variable for Specification of Test Records

Name:	Meaning:
select_test	contains the test numbers wanted as a bit pattern.

3. Standard Variables Used by The Central Logic

Name:	Meaning:
co_time_base	8000-clock-time at the last test for timeout. Basis for time indication in the coroutine system. Must not be changed!
co_time	the time until the next possible timeout measured in tenth of a second from co_time_base.
co_8000_event	the number of unprocessed 8000-events found in the latest scan of the 8000-event queue.
co_own_base	the address of the first own variable in the coroutine system. Must not be changed.

2.4 Procedures in ALGOL Library 2.4

The following ALGOL library procedures are used by the system and the users of the system.

For a detailed description of the individual procedures, cf. ref. [1].

Name:	Function:
activity	Used to create a number of empty activity descriptors before calling procedure coroutines.
new_activity	Initiates an empty activity with a procedure and starts the activity. Must be called after procedure coroutines and before any other coroutine procedure.
activate	Activates a non-empty activity in its restart-point. Used by procedure schedule. Must not be used in coroutines scheduled by this procedure.
passivate	De-activates the executing activity, establishing its restart point (waiting point). Used in procedure wait. May be used in the coroutines, but does not change the location in any semaphore queue of executing coroutine.

3. DATA STRUCTURES 3.

The system introduces 3 data structures and a new type of variable in connection with the coroutine system.

The data structures are:

a) A semaphore table containing chain fields for all semaphores.

b) A coroutine description table containing information about the priority of each coroutine, the maximum waiting time, what the coroutine is waiting for and chain fields.

c) Message buffers each consisting of a head and a data part. The head contains priority, length and chain fields.

The new type of variable is a reference type used to refer to the messages.

3.1 Semaphore Tables 3.1

Semaphores in the system are always message semaphores. They are identified by number.

There are 3 types of semaphores:

a)	user semaphores	numbered 1 to max-semaphore,
b)	central logic semaphores	numbered -5 to 0,
c)	system semaphores	numbered -9 to -6.

The semaphore description consists of 8 halfwords per semaphore.

3.1.1 User Semaphores 3.1.1

The number of user semaphores is stated when the system is started up calling procedure coroutines. The semaphores are used for signal and wait operations.

3.1.2 Central Logic Semaphores 3.1.2

These semaphores, defined by the central logic, are used in connection with the facilities made available by the central logic.

The semaphores can be used by means of procedures cor_to_sem, signal and wait.

3.1.3 System Semaphores 3.1.3

System semaphores are used by the coroutine procedures.

The system semaphores are:

No:	Name:	Semaphore queue contains:
-9	ready semaphore	coroutines ready to be activated.
-8	implicit passivate semaphore	coroutines implicitly passivated.
-7		not used.
-6	free semaphore	coroutines not used at present.

3.2 Coroutine Description 3.2

The procedure coroutines will create a coroutine description for each coroutine. The description consists of 16 halfwords per coroutine. It must not be confused with the activity description created by procedure activity, consisting of 20 halfwords per activity.

Apart from chain fields, the coroutine description contains coroutine priority, maximum waiting time accepted and a "message lock", which is the value of wait_select the last time procedure wait was called by the coroutine.

The coroutine description will always be queued up on a semaphore in order of priority. When the system is started up, all coroutine descriptions are queued up on the free semaphore.

3.3 Message Buffers 3.3

Communication between coroutines takes place by means of messages placed in message buffers. These buffers are allocated in the stack by means of procedure allocate.

Apart from the data part, a message buffer contains a protected head with chain fields, the length of the data part and the priority of the message buffer.

A message buffer occupies 8 halfwords + the length of the data part, which must be at least 6 halfwords.

The system considers the first two words of the data part to be a message key. If "the message lock" in a coroutine description queued up on a sempahore is not zero, it will be compared with the key words in all message buffers arriving at the semaphore, and only if the key fits will the message be transferred to the coroutine.

In this way several pairs of coroutines can communicate messages via the same semaphore, or two coroutines can communicate different types of messages.

3.4 Reference Variable 3.4

A new type of variable, a reference variable, is used to refer to messages in the message buffers.

A reference variable can be regarded as an array, whose length and location in the stack changes according to the state of the variable.

The state can be one of the following:

'array'

The reference variable refers to a message buffer.

In this state the reference variable functions as an array containing the message.

The array has the same length as the data part of the message buffer and has lower index = 1.

Fielding and indexing are made in the same way as with a normal array.

The type of array equals the type declared for the reference variable, i.e. boolean, integer, real or long.

'nil'

The reference variable does not refer to a message buffer, and it has the length 0.

Fielding and indexing in this state will cause a field/index alarm.

Reference variables are declared in the following way:

```
⎡ boolean ⎤ 1
⎢ integer ⎥
⎨         ⎬   array <name> (1:1)  {,<name> (1:1)} *  ;
⎢ long    ⎥                                       0
⎣ real    ⎦ 0
```

A reference variable <u>must</u> be initialized before being used. This is done by calling the procedure:

```
initref (<name>);
```

and the state will be 'nil'.

<u>Note</u>:

1) The array bound (1:1) must be stated in the declaration for <u>each</u> reference variable separately.

2) A parameter to an ALGOL procedure will be a reference variable only if:

   a) the parameter on the call side is an unfielded reference variable,
   b) the parameter is not fielded or indexed in the procedure itself.

   If fielding or indexing of a reference variable is necessary in a procedure, this part of the procedure must be transformed into one or more procedures called with the reference variable as actual parameter.

3) At the end of the block in which a reference variable has been declared, it should be in the 'nil' state, otherwise the message buffer referred to will be lost.

4. CENTRA LOGIC 4.

The procedure centra_logic may be used as central logic to schedule coroutines and communicate messages. It does not return until a corutine terminates, either via its final end or because of an alarm.

The procedure can be called again.

Call: centra_logic (log);

centra_logic (return value, long). The result of the last call of schedule.

log (call value, integer). Log is interpreted as a bit pattern specifying the log records wanted. (cf. section 4.3).

Program mode: At call, the program mode must be cor_monitor (cf. appendix C).

Procedure centra_logic uses the central logic semaphores in the following way:

No:	Name:	Use:
0	wait_message_pool	(cf. section 4.1)
-1	wait_message	(cf. section 4.1)
-2	wait_answer_pool	(cf. section 4.2)
-3	delay	may be used freely
-4		not used
-5	virtual_error	cf. point 4 below

Procedure centra_logic performs the following functions:

1) Starts up coroutines queuing them up on the proper start semaphore. (This is due to the fact that the central logic does not get the result from the call of new_activity).

2) Restarts the highest priority coroutine ready to be started. (Procedure schedule selects the coroutine).

3) Coroutines deactivated with an implicit passivate statement are queued up on the implicit passivate semaphore. They are restarted in order of priority when their 8000-answers are found in the 8000-event queue.

4) Coroutines which cannot be restarted because of "stack busy" (activate result -2) are queued up on the virtual_error semaphore. When the blocking coroutine later passivates explicitly, the blocked coroutine will be queued up on the ready semaphore.

   Note: There must be no call, explicitly or implicitly of passivate between 8000-send-message and 8000-wait-answer in virtual coroutines. Consequenly virtual coroutines cannot send 8000-messages and receive 8000-answers, in multibuffered implicit-passivate-zones.

5) resets wait_time and wait_select before each coroutine is activated.

6) Communicates incoming 8000-messages (cf. section 4.1).

7) Communicates on request 8000-answers via messages sent to specified semaphores (cf. section 4.2).

8) Generates timeout on 8000-answers (cf. Section 4.2).

9) If no coroutines are ready to be started, the 8000-event queue is scanned for external events.
   One event to be found is the answer from the 8000 clock process to a delay operation sent by centra_logic itself with the purpose of being timed out from the event queue to restart timed-out coroutines.

10) Displays logging records showing operation flow and time consumption.

4.1 Processing of 8000-messages 4.1

The semaphores 0 and -1 are used in connection with communication of 8000-messages.

If a coroutine wishes to receive an 8000-message there must be a message queued up on the semaphore wait_message_pool.

When, scanning the event queue, an 8000-message is found, centra_logic will examine whether there is a message queued up on the wait_message_pool semaphore. If a message is found and a coroutine is queued up on the wait_message semaphore waiting for the 8000-message, the 8000-message buffer will be copied to the message, which is signalled to the wait_message semaphore:

field addr.	
+2	sender process description addr.
+4	receiver process description addr.
+6	8000-message buffer addr.
+8	
.	
.	8000-message
.	
+22	

If the message is less than 22 halfwords, only the part of 8000-message for which there is room will be copied. If the message is greater than 22 halfwords the rest of the message will be undefined.

By means of wait_select it is possible for a coroutine to wait for a message from a particular process or/and a particular pseudo process.

4.2 Conversion of 8000-answers into Messages 4.2

In connection with conversion of an 8000-answer into a message, semaphore -2 (wait_answer_pool) is used.

This facility can be used if:

1) A coroutine has to wait for one or more 8000-answers.

2) A coroutine has to wait for the first of either a message or an 8000-answer.

3) A coroutine wishes to regret an 8000-message, if no 8000-answer is received within a certain time (timeout).

4) A coroutine has sent an 8000-message via a global zone, and another coroutine has to wait for the 8000-answer.

The facility is used in the following way:

When an 8000-message has been sent by means of monitor (16 <*send_message*>, ...) a message with the following format is signalled to wait_answer_pool:

field addr.	
+2	not used
+4	8000-message-buffer-address
+6	answer semaphore

Whenever procedure centra_logic finds an 8000-answer in the 8000-event queue, it is examined whether a message with this 8000-message-buffer-address is queued up on wait_answer_pool. If so, the message is communicated to the semaphore stated in the 3rd word of the message.

Procedure centra_logic does not change the message, which only need to consist of 6 halfwords.

In case 3 where the coroutine does not wait for the 8000-answer, the coroutine must withdraw its message from wait_answer_pool before calling 8000-regret-message.

4.3 Demand for Log Display 4.3

The log display wanted by procedure centra_logic is stated by the parameter log. The log display is printed on current output.

The parameter log must contain the sum of the numbers of the printouts wanted:

No	Meaning
1	Various counters.
2	The semaphore table and associated records at normal coroutine termination.
4	The semaphore table and associated records at coroutine termination with alarm.
8	The semaphore table and associated records at start up.

5. TEST FACILITIES IN THE COROUTINE SYSTEM 5.

The coroutine procedures contain a possibility of creation of test records in a zone belonging to the coroutine system.

The zone, which must be declared and opened in the user program, is the second parameter to the procedure coroutines. The creation of test records is demanded by means of the standard integer variable select_test. The different test record types have a number, which is a power of two and so select_test must have a value corresponding to the sum of the numbers of the test record types wanted.

The following test record types exist:

1: the first part of message at call of procedure signal
2: the first part of message at return from procedure wait
4: call of procedure signal
8: call of procedure wait
16: return from procedure wait
32: not used
64: transfer of coroutine to another semaphore queue
128: start up of coroutine

The system contains a procedure prepare_test, which creates a test record containing test type (= 1024), coroutine number, the hour and zeroes in the remaining fields.

When prepare_test has been called, the test record is the current zone_record and may be changed freely.

6. PROCEDURE DESCRIPTION 6.

The following procedure description, which describes standard procedures as well as standard variables, is divided into 3 sections: user procedures, test procedures and central logic procedures.

6.1 User Procedures 6.1

6.1.1 Allocate 6.1.1

This standard procedure is used to allocate stack space for a message buffer, initialize it with priority and a message, which in the second word contains the message buffer length and otherwise is zerofilled. This message is signalled to the semaphore. The procedure must be called after the procedure coroutines and at the same block level.

Call: allocate (sem, message_size, prio);

sem	(call value, integer). The number of the semaphore to which the message is to be signalled. 0 <= sem <= max_semaphore.
message_size	(call value, integer). The length in halfwords of the message buffer to be reserved. 6 <= message_size.
prio	(call value, integer). The priority to be assigned to the message buffer. -2048 <= prio <= 2047. (2047 is the highest priority).

Program mode: The program mode must be cor_monitor.

6.1.2 Coroutines 6.1.2

This standard procedure is used to allocate stack space for semaphore and coroutine descriptions. The coroutine descriptions are queued up on the free semaphore with the priority 0.

The procedure activity must be called before procedure coroutines and at the same block level.

Procedure coroutines must be called before procedure new_activity and before all other coroutine procedures.

Call: coroutines (max_semaphore, test_zone)

max_semaphore	(call value, integer). The number of user semaphores. These semaphores are numbered from 1 to max_semaphore.
test_zone	(call value, zone). The zone in which test records are created. The zone must be open and ready for record output when the creation of test records is enabled (cf. prepare_test and select_test).

Program mode: At call the program mode must be act_monitor, at return it will be cor_monitor.

6.1.3 Initref 6.1.3

This standard procedure is used to initialize a reference variable.

Call: initref (ref);

ref	(call and return value, boolean array, integer array, long array or real array). The array to be transformed into a reference variable. At return the state of the reference variable will be 'nil'.

Program mode: The program mode must be one of the cor_modes (cf. appendix C).

The array ref must be declared in the following way:

```
 ┌         ┐ 1
 │ boolean │
 │ integer │                                             *
<          >   array <name> (1:1)  { ,<name> (1:1) }   ;
 │ real    │                                             0
 │ long    │
 └         ┘ 0
```

6.1.4 Set Priority 6.1.4

This standard procedure changes the priority of a coroutine. When the change has been made, the coroutine is queued up on the ready semaphore in order of its new priority and is passivated.

Call: set_priority (prio);

prio	(call value, integer). The new priority. -2048 <= prio <= 2047. (2047 is the highest priority).

Program mode: The program mode must be cor_activity.

As the procedure queues up the coroutine on the ready semaphore after all coroutines with the same or higher priority, the call:

```
set_priority (act_prio);
```

act_prio being the priority of the coroutine, will queue up the coroutine behind all coroutines with the same priority.

6.1.5 Signal 6.1.5

This boolean standard procedure signals a message to a semaphore.

Call: signal (sem, ref);

signal (return value, boolean). True if the message key fits the lock of a waiting coroutine (i.e. a coroutine is transferred to the ready semaphore, queue), false otherwise.

sem (call value, integer). The number of the semaphore to which the referenced message is signalled.
-5 <= sem <= max_semaphore.

ref (call and return value, reference variable). ref is a reference variable which refers to the message. At call the state of ref must be 'array', at return the state is 'nil'.

Function:

Procedure signal proceeds in the following steps:

1) If one or more coroutines are queued up on the specified semaphore, they are examined, in order of priority, to have a message lock in their coroutine description to which the message key fits. (see subsection 6.1.7).

2) If a coroutine is found, the message buffer address will be inserted in the coroutine description, and the coroutine is queued up on the ready semaphore in order of priority.

3) If no coroutine is found, the message is queued up on the semaphore message queue in order of priority.

4) The procedure returns.

Program mode: The program mode must be one of the cor_modes (cf. appendix C).

Even if the procedure transfers a coroutine with a higher priority than that of calling coroutine to the ready semaphore, the procedure will not de-activate calling coroutine.

6.1.6 Wait

This integer standard procedure waits for a message at a semaphore.

Call: wait (sem, ref);

wait	(return value, integer). = 0: timeout, no message received > 0: length in halfwords of the data part of message received (the data part is always greater than five halfwords).
sem	(call value, integer). The number of the semaphore where to wait for the message. -5 <= sem <= max_semaphore.
ref	(call and return value, reference variable). At call, the state of ref must be 'nil'. At return the state of ref is 'array' and the value of ref will refer to the message received, if one is received.

Function:

Procedure wait proceeds in the following steps:

1) Current values of the standard variables wait_select and wait_time are inserted in the coroutine description.

2) If one or more messages are queued up on the semaphore specified, they are examined, in order of priority, to have a message key that fits the message lock stored in the coroutine description (cf. subsection 6.1.7).

   If more messages of the same priority are found, the one which has been queued up for the longest time is taken.

3. If program mode is cor_monitor or cor_disable, or if wait_time < 0, the procedure proceeds at point 7. with message found or timeout if no message was found.

4. If no message was found, the coroutine is queued up on the semaphore specified.

   If a message was found the coroutine stays in the ready semaphore queue.

5. The procedure examines whether there is timeout for any of the other coroutines in the system or, if any higher priority coroutine is implicitly passivated, whether there is an 8000-answer for any of the implicitly passivated coroutines. If so, they are queued up on the ready semaphore in order of priority.

6. If a message was found, and calling coroutine is the first in the ready semaphore queue, the procedure proceeds at 7., else the coroutine is passivated.

7. This is a re-start point with a delay in case the message waited for is received or in case of timeout. It is the continue point with no delay, whether or not a message is received in case the procedure is called in cor_disable or cor_monitor mode or if the coroutine will accept no delay.

   If a message is received, the reference variable specified is changed to refer it, and its state is changed to 'array'.

8. If a message is received, wait returns with the length of the datapart, else with the value 0.

Program mode: The program mode must be cor_modes (cf. appendix C).

6.1.7 Wait_select 6.1.7

This long standard variable affects the progress of the next wait.

Procedure wait starts by inserting the value as a message lock in the coroutine description, whereupon wait_select is zeroed.

A coroutine must not passivate, explicitly or implicitly, between an assignment to wait_select and a call of wait, because wait_select is zeroed by procedure centra_logic.

A reactivated coroutine will always find wait_select = 0.

The value wait_select is inserted as a message lock in the coroutine description, which must be fitted by a message key found in the first two words of a message.

The lock/key comparison made by procedure wait as well as by procedure signal both use the value of wait_select inserted in the coroutine description by procedure wait.

Two integer arrays, message_key and message_lock, containing the first two words of the message as a message key and the message lock from the coroutine description respectively, are used in the comparison algorithm.

The comparison is made according to the following algorithm:

```
accept:= true;

for i:= 1,2 do
if message_lock(i) <> 0 then
begin
  if message_lock(i) > 0 then
  begin
    if message_lock(i) <> message_key(i) then accept:= false
  end else
  begin <* message_lock(i) < 0 *>
    if logand (message_lock(i), message_key(i)) = 0 then
    accept:= false
  end
end;
```

If accept is still true, the message key fits the lock.

6.1.8 Wait_time

This integer standard variable affects the progress of the next wait.

Procedure wait starts by inserting the value as maximal delay accepted in the coroutine description, whereupon wait_time is zeroed.

A coroutine must not be passivated, explicitly or implicitly, between an assignment to wait_time and a call of wait because wait_time is zeroed by procedure centra_logic.

A re-activated coroutine will always find wait_time = 0.

The value of wait_time is inserted as maximal delay accepted waiting for a message.

wait_time > 0: the maximum waiting time accepted in tenth of seconds. If no message is received within this period of time, procedure wait returns without a message.

wait_time = 0: no limits to the waiting time.

wait_time < 0: procedure wait accepts no delay and returns at once.

If a message is received it is returned, else wait returns without a message. The coroutine is not passivated even if higher priority coroutines are queued up on the ready semaphore.

Example 1:

Coroutine that prints out the time every 5 minutes.

```
procedure write_time (z);
zone z;
begin
  real time;
  integer delay_sem;
  integer array dummy (1:1);
  initref (dummy);
  delay_sem:= -3;
  repeat
    systime (5, 0.0, time);
    write (z, <:<'nl'>time:>, <<dd dd>, (entier time)//100);
    setposition(z,0,0);

    wait_time:= 5 * 60 / 0.1024; <* 5 min *>
    wait(delay_sem, dummy);
  until false
end;
```

6.2 Test Procedures 6.2

6.2.1 Prepare_test 6.2.1

This standard procedure creates a test record in the test zone buffer. The test record, which consists of 16 halfwords, is created by means of outrec6.

The record is initialized with test type, current coroutine and hour, cf. appendix D.

After return from the procedure, further specification may be inserted in the test record, which is available in the test zone buffer.

Call: prepare_test

At call the zone state for the test zone (2nd parameter in the call of procedure coroutines) must be 0 or 6 (ready for record output).

6.2.2 Select_test 6.2.2

This integer standard variable selects the test records to be created. The value of the variable can be changed dynamically during program execution if only the test zone is open and ready for record output (zone state 0 or 6), before select_test is given a value different from 0.

Standard procedure coroutines zeroes select_test.

The value of select_test must be the sum of the numbers of the test record types wanted:

Test record type	Meaning
1	Message buffer at call of procedure signal
2	Message buffer at return from procedure wait
4	Call of procedure signal
8	Call of procedure wait
16	Return from procedure wait
64	Transfer of coroutine to another semaphore queue
128	Start up of coroutine.

6.3 Auxiliary Central Logic Procedures 6.3

These standard procedures and standard variables are used in the central logic procedure of the system (centra_logic).

The procedures are thus only interesting if an alternative central logic is to be made, or if one wishes to know how centra_logic works.

6.3.1 Co_8000_event 6.3.1

This integer standard variable holds information about unprocessed 8000-events, if any, from the last scan of the 8000-event queue.

If the value of co_8000_event differs from 0 at entry to procedure centra_logic the procedure will scan the 8000_event queue before starting up the next coroutine.

co_8000_event is assigned by procedures schedule and centra_logic after each scan of the 8000_event queue.

6.3.2 Co_own_base 6.3.2

This standard integer contains the start address of the own core of the coroutine system. See appendix B.

6.3.3 Cor_to_sem 6.3.3

This standard procedure is called by the central logic to transfer coroutines from one semaphore queue to another. A coroutine is always inserted in order of priority.

Call: cor_to_sem (sem, cor);

sem (call value, integer). The number of the semaphore where the coroutine is queued up. -9 <= sem <= -1

cor (call value, integer). The number of the coroutine.

Program mode: The program mode must be cor_monitor (cf. appendix C).

Alarm: The procedure must not be called as a formal procedure or with formal expressions.

6.3.4 Co_time 6.3.4

This integer standard variable controls - together with the variable co_time_base - when the next possible timeout can take place.

The time of the next possible timeout is

```
co_time_base + (extend co_time shift 10);
```

6.3.5 Co_time_base 6.3.5

This long standard variable contains the basis of time measurements in the coroutine system.

The value of the variable must not be changed.

co_time and remaining wait_time in the coroutine descriptions are measured in units of 0.1024 sec and are used relative to co_time_base.

6.3.6 Schedule 6.3.6

This long standard procedure is used by the central logic instead of procedure activate.

The procedure starts by searching for the next coroutine to be started. If one is found, schedule calls activate. In this case the result value is the same as the result value from activate, and the return value of the parameter is the number of the coroutine activated/attempted activated.

If no coroutine is found, the procedure returns with the result value 0 and the parameter value 0.

<u>Call</u>: schedule (cor);

schedule	(return value, long). If activate has been called, the return value of procedure activate, otherwise 0.
cor	(return value, integer). If activate has been called, the number of the coroutine activated, otherwise 0.

<u>Program mode</u>: The program mode must be cor_monitor (cf. appendix C).

Alarm: The procedure gives parameter error if called with expression or a constant as parameter.

<u>Function</u>:

1. If co_8000_event is negative, the 8000-event queue is scanned.

   At a scan of the 8000-event-queue, coroutines queued up on the implicit-passivate semaphore are transferred to the ready queue when the 8000-answers waited for are found in the queue.

   At the same time, the number of events in the 8000-wait-queue, which are not answers to coroutines implicitly passivated, are counted in the variable co_8000_event.

2. If the ready semaphore is empty, coroutines which have exceeded their maximum waiting time are transferred to the ready semaphore queue.

   If no coroutines are transferred to the ready semaphore queue, the 8000-event queue will be scanned as described in 1.

3. If the ready semaphore queue is still empty, procedure schedule returns, else the first coroutine on the ready semaphore queue is activated.

A. REFERENCES

[1] RCSL No 42-i1278:
ALGOL8, User's Guide, Part 2

[2] RCSL No 31-D476:
RC8000 MONITOR, Part 1, System Design

[3] RCSL No 31-D477:
RC8000 MONITOR, Part 2, Reference Manual

B. INTERNAL TABLES B.

This appendix contains the formats of the internal tables.

The addresses of the fields are relative addresses measured in halfwords relative to the base address of the table.

Section B.1 contains a description of the own core of the coroutine system.

Section B.2 contains a description of chain elements.

Section B.3 contains a description of the semaphore table.

Section B.4 contains a description of the coroutine descriptions.

Section B.5 contains a description of the message buffers.

Section B.6 contains procedures for displaying the tables.

B.1 The Own Core of the Coroutine System B.1

External procedure coroutines own core.

rel. addr.	
+0	"max_semaphore"
+2	"base address of the semaphore table"
+4	"the address of the last coroutine"
+6	"the address of the coroutine 0"
+8	co_8000_event
+10	co_time
+12 } +14 }	co_time_base
+16	wait_time
+18 } +20 }	wait_select
+22	co_own_base "base address of this table"
+24	"the length of activity's description per coroutine"

+26 +28	"testzone formals"
+30	select_test
+32 +34 +36 +38 +40	"testrecord"

B.2 Chain Elements B.2

Semaphore tables, coroutine descriptions and message buffers contain chain elements. These chain elements consist of the address of the next element and the address of the previous element.

A chain element looks like this:

rel. addr.	
-2	the address of the next element (towards lower priority)
+0	the address of the previous element (towards higher priority)

If a chain element has no references it points at itself (i.e. at the field with the relative address 0).

B.3 Semaphore Table B.3

The table is placed as an array from -9 to max_semaphore of semaphore elements each consisting of 8 halfwords.

Each semaphore element consists of a message chain element and a coroutine chain element.

"Base address of the semaphore table" (see B.1) contains the address of the coroutine chain element of semaphore 0.

A semaphore element has the following fields:

rel. addr. within sem.	
-6	the address of the first message (highest priority)
-4	the address of the last message (lowest priority)
-2	the address of the first coroutine (highest priority)
+0	the address of the last coroutine (lowest priority)

NB: When coroutines/messages are queued-up on a semaphore queue in order of priority, the first address of the previous chain element is used as priority. This implies that no signals are sent to the lowest semaphore (the ready semaphore) and that addresses always exceed 2047, which is in fact the case.

B.4 The Coroutine Description Table B.4

The table is placed as an array from 0 to no_of_coroutines of coroutine description elements each consisting of 16 halfwords. An element has the following fields:

rel. addr.	
-6	the priority of the coroutine
-4	message buffer address if a message has been found
-2	the address of the next coroutine (towards lower priority)
0	the address of the previous coroutine (towards higher priority)
+2	wait_select shift (-24)
+4	wait_select extract 24
+6	remaining wait_time in relation to co_time base
+8	coroutine number

The coroutine descriptions can be found either by means of the semaphore tables or by means of "the address of coroutine 0".

B.5 Message Buffers B.5

A message buffer has a head of 8 halfwords and a data part of at least 6 halfwords. A message buffer can only be found by means of the address, as it must be either queued up on a semaphore or referred to by a reference variable.

The message buffer has the following format:

rel. addr.	
-6	the priority of the message buffer
-4	the length of the message buffer data field in halfwords
-2	the address of the next message buffer (towards lower priority)
0	the address of the previous message buffer (towards higher priority)
+2	message key 1 message
+4	message key 2 message
+6	. message
.	.
.	.
.	.

B.6 Procedures used to Display the Tables B.6

```
own integer max_sem, sem_basis, cor_basis, max_cor;

procedure initowns;
begin
  integer array own_core(1:4);
  system(5, co_own_base, own_core);
  max_sem   := own_core(1);
  sem_basis := own_core(2);
  cor_basis := own_core(4);
  max_cor   := (own_core(3) - cor_basis) shift (-4)
end;
```

```
procedure writesem(sem);
integer sem;
begin
  write(out, <:sem:>, <<-ddd  >, sem,
<* pos *> if sem>0 then <:user:> else
          case sem+10 of(
<*  -9 *> <:ready:>,
<*  -8 *> <:impl. pass.:>,
<*  -7 *> <::>,<*not used*>
<*  -6 *> <:free:>,
<*  -5 *> <:virt. error:>,
<*  -4 *> <::>,<*not used*>
<*  -3 *> <:delay:>,
<*  -2 *> <:wait answ. pool:>,
<*  -1 *> <:wait mess.:>,
<*   0 *> <:wait mess. pool:>))
end;

integer procedure where(cor);
value cor;
integer cor;
begin
  <* the procedure returns the number of the semaphore on which
     the coroutine is queued up.
  *>
  integer addr;
  integer array chain_field(1:4);
  initowns;
  addr:=cor shift 4 + cor_basis;
  for addr:=addr, chain_field(4) while chain_field(1)<2048 do
  begin
    where:=(addr-sem_basis)//8;
    system(5, addr-6, chain_field)
  end
end;
```

```
procedure printsemtable;
begin
  <* for all semaphores the procedure will display the
     coroutines/messages in the queue.
  *>
  integer array sem_descr(1:3), cor_descr(1:8), mess_head(1:9);
  integer addr, sem;

  procedure printsem(semaddr);
  value semaddr;
  integer semaddr;
  begin
    integer addr;
    procedure printcor;
    begin
      system(5, addr-6, cor_descr);
      write(out, <:<'nl'>  cor:>, <<ddd >, cor_descr(8),
        <<-dddd  >, <:prio=:>, cor_descr(1),
        <:ident:>, <<-dddddd>, cor_descr(5), cor_descr(6));

      if cor_descr(7) > 0 then
      writeint(out, <: wait_time:>, << d.d>, cor_descr(7));
      addr:=cor_descr(3)
    end;

    procedure printmess;
    begin
      integer i, size;
      system(5, addr-6, mess_head);
      write(out, <:<'nl'>   mess   prio=:>, <<-dddd  >,
        mess_head(1), <:size:>, mess_head(2));
      size:=if mess_head(2) > 10 then 5 else mess_head(2)//2;
      for i:=1 step 1 until size do write(out, mess_head(4+i));
      addr:=mess_head(3)
    end;
```

```
    system(5, semaddr-6, sem_descr);
    if sem_descr(1)<>semaddr-4 or sem_descr(3)<>semaddr then
    begin
      write(out, <:<'nl'><'nl'>:>);
      writesem(sem);

      addr:=sem_descr(3);
      while addr<>semaddr do printcor;

      addr:=sem_descr(1);
      while addr<>semaddr-4 do printmess;
    end
  end;
  initowns;
  for sem:=-9 step 1 until max_sem do printsem(sem*8 + sembasis)
end;

procedure printcorsem;
begin
  <* for all coroutines the procedure displays the number and
     possible name of the semaphore on which the coroutines are
     queued up.
  *>
  integer cor;
  initowns;
  for cor:= 1 step 1 until max_cor do
  begin
    write(out, <:<'nl'>cor:>, <<dddd>, cor, <: on :>);
    writesem(where(cor))
  end
end;
```

C. PROGRAM MODE C.

The coroutine system defines certain program modes not used by activity.

The following program modes exist:

mode value	program mode of the coroutine system	activity program mode
8	neutral	neutral
17	act_monitor	monitor
18	act_activity	activity
20	act_disable	disable
33	cor_monitor (cor_modes)	monitor
34	cor_activity (cor_modes)	activity
36	cor_disable (cor_modes)	disable

The mode values stated will be displayed in the alarm caused by a procedure called in an illegal program mode.

Survey of the use of program modes:

	program mode	
procedure	before the call	after the call
allocate	33	unchanged
coroutines	17	33
initref	33, 34, 36	unchanged
set priority	34	unchanged
signal	33, 34, 36	unchanged
wait	33, 34, 36	unchanged
cor to sem	33	unchanged
schedule	33	33

D. TEST RECORDS

The individual test records consist of 16 halfwords and are created by means of outrec6. The format of the different test records are shown in fig. 1.

Comments to the table:

re [1]: The first 7 words of the message. If the message is shorter, the rest is undefined.

re [2]: If s-data is requested, the value is 5 and the following record an s-data record, otherwise the value is 4.

re [3]: The coroutine from which the procedure is called. If the procedure is called in the disable mode, the value is the negative value of coroutine number, and if it is called in the monitor mode, the value is 0.

re [4]: If w-data is requested, the value is 18, and the following record is a w-data record, otherwise the value is 16.

re [5]: This record is created by means of the procedure prepare_test. The fields can be changed freely after the call, only the test number must be outside the interval 0 - 1023.

re [6]: The hour contains 8000_time (measured in 0.1 msec). The value can be printed out in readable form:

```
begin
  real date, time;
  long field hour;
  hour:= 16;
  date:= systime(4, z.hour/10000, time);
  write(out, << dd dd dd>, date, time)
end;
```

TEXT \ WORD	1 type	2	3	4	5	6	7–8
s-data	1	←		message [1]			→
w-data	2	←		message [1]			→
signal	4/5 [2]	m.buf. addr.	m.buf. length	m.priority	semaphor No	act.cor.No [3]	hour [6]
wait	8	wait_select		wait_time	– –	–	–
wait-exit	16/18 [4]	m.buf. addr.	m.buf. length	c.priority	– –	–	–
cor_to_sem	64	corout.No	undefined	undefined	– –	0	–
activate	128	–	undefined	undefined	undefined	0	–
user	1024 [5]	0	0	0	0	act.cor.No [3]	–

Fig. 1.

After end of execution the test records can be displayed by means of print. It can be done by means of the following call, in which each test record is printed out in a separate line:

```
print testfile word words.8
```

E. ALARMS FROM THE COROUTINE SYSTEM E.

alarm text	procedure	explanation
c-level coroutines	coroutines allocate	The procedure has not been called at the same block level as activity.
index <j> coroutines	coroutines	The parameter max_semaphore has the illegal value <j>.
	allocate	The parameter 'sem' has the illegal value <j>.
index <j> schedule	cor_to_sem	The parameter 'cor' has the illegal value <j>.
index <j> signal/wait	signal wait	The parameter 'sem' has the illegal value <j>
not ref. signal/wait	signal wait	2nd parameter in the procedure call is not a reference variable.
p-mode <j> coroutines	coroutines allocate	The procedure has been called in a wrong program mode <j>. Cf. appendix C.
p-mode <j> schedule	initref cor_to_sem schedule set_priority	The procedure has been called in a wrong program mode <j>. Cf. appendix C.
param schedule	initref	first parameter in the call has not been declared as a reference variable.
	cor_to_sem schedule	the procedure has been called either as a formal procedure or by means of a formal expression.

alarm text	procedure	explanation
param signal/wait	signal wait	The procedure has not been called in cor_modes.
ref.arr signal/wait	wait	The reference variable has the state 'array' at call.
ref.nil signal/wait	signal	The reference variable has the state 'nil' at call.

RETURN LETTER

Title: ALGOL Coroutine System
User's Guide

RCSL No.: 31-D674

A/S Regnecentralen af 1979/RC Computer A/S maintains a continual effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback, your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability:

____

____

____

____

Do you find errors in this manual? If so, specify by page.

____

____

____

____

How can this manual be improved?

____

____

____

____

Other comments?

____

____

____

____

____

Name: ____________________ Title: ____________________

Company: ____________________________________

Address: ____________________________________

Date: __________

Thank you

42-i 1288

. . . . Fold here . . . .

. . . . Do not tear - Fold here and staple . . . .

Affix postage here

A/S REGNECENTRALEN
af 1979
Information Department
Lautrupbjerg 1
DK-2750 Ballerup
Denmark