RCSL No: 31-D673

Edition: June 1982

Author: Carl Henrik Dreyer

Title:

BOSS
Basic Internal Formats
Maintenance Manual

A/S REGNECENTRALEN af 1979

RCSL 42-i1905

TABLE OF CONTENTS (continued) PAGE

1. INTRODUCTION 1.

This book is a maintenance manual, and so it is a working document for the people maintaining BOSS: It is a collection of descriptions that are needed over and over again when programming for BOSS, and so a general understanding of the structure of the system is more or less presupposed.

The program text for the online system is contained in 10 files, compiled separately (but using the same option file). These files are referred to as MODULES.

The first MODULE, named 'bos' (or central) contains the basic elements:

- Central Logic procedures (or C.L.)
- Pager coroutine
- Initialization and loader/linker.

Besides this, the commonly used data structures are described. These descriptions are closely related to the method of referencing the structure in slang (suppose the absolute address of first word is contained in register W1):

+0:			first word, referred as "X1".
+2:			second word, as "X1+2".
+4:			third word, as "X1+4".
+6:		+7:	hw No 7, as "X1+6". hw No 8, as "X1+7".

2. COMMON DATA STRUCTURES 2.

The terms segment, section, virt. addr are described in section 5.

2.1 Common Resident Tables 2.1

The common data structures resident in BOSS core are placed in tables in the high-address end of BOSS core at run time.

In this survey, they are mentioned from the low address end:

- core table
- segment table

} Allocated after first loader pass.

- semaphore table — Allocated during first loader pass.
- terminal buffers
- sender table
- coroutine table
- file access table
- run test buffers

} Allocated at init (before first loader pass).

(- top address of BOSS)

2.1.1 Core Table 2.1.1

Core table holds one entry for every segment place in page core (see chapter 5). Coretable is indexed with core index: segment places counted from 0. Segment number (in + 3) is relative to first drum segment (cf. 5.6).

PAGE CORE

Core index: 0 1 2 3 4 5 - - - -

CORE TABLE

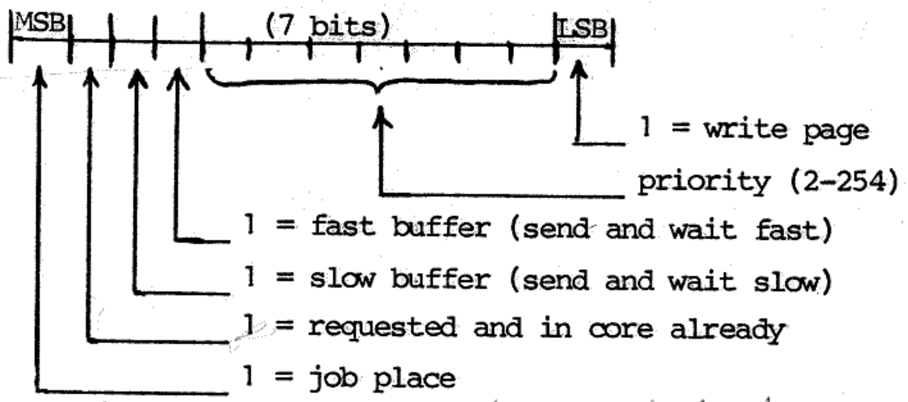
Entries: 0 1 2 3 4 5 - - - -

Format of a core table entry (size = 4 hw):

+0:	core index for first segment of section
+1:	priority (= 0 for later segments)
+2:	length of section
+3:	segm number of first segment of section

core index for first segm of section
forklar

Priority (12 bits):



priority: updated at each reference (forklar)

Length:

4 * number of segments for first segment in section, 0 for later segments, 4 for free segments and later segment places for a job.

Core table is terminated with a dummy top entry with the format:

first index = 0

priority = 4095

length = core table length - 4

segment No = 4095

~~segm number for first segm of section~~ forklar

2.1.2 Segment Table 2.1.2

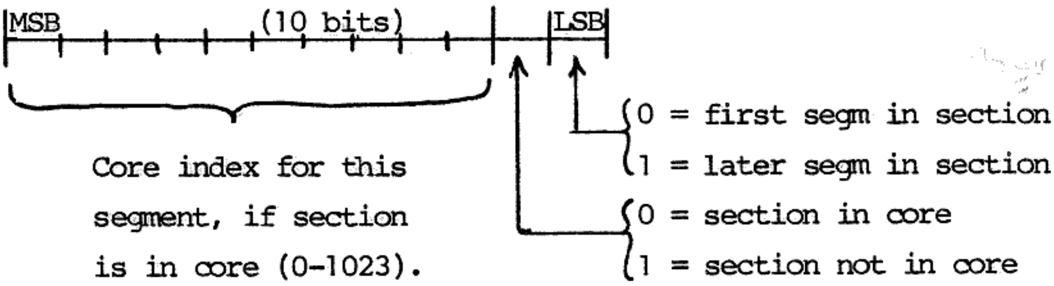
Segment table holds one entry for every segment in virtual core (drumcore/disccore - see chapter 5).

Segment table is indexed with segment No: first entry is segm No = (last addr of BOSS) //512+1, so only segment places of virtual core are contained in segment table.

særskilt afsnit om "virtual addresses" (sec. 5)

Format of a segment table entry:

Length = 1 hw, 12 bits:



Notice that when the two least significant bits are marked off, the segment table entry directly equals the relative address of the corresponding core table entry (because each core table entry has a length of 4 hw).

2.1.3 Semaphore Table 2.1.3

The semaphore table is allocated (and defined in size) during first loader pass, by reservations in external 19.

The semaphore table contains semaphore descriptions and also a diversity of private datastructures, not described here.

Format of a semaphore description:

SEMAPHORE ADDR →	+0: SEM. VALUE	length = 6 hw (option e13)
	+2: FIRST IN QUEUE	
	+4: LAST IN QUEUE	

Examples of contents of semaphore descriptions:

CORUNO is addresses of a coroutine description
VIRT.OP is virtual address of an operation.

- SIMPLE SEMAPHORE: (OPEN USED)

-2	-1	0	1	2
FIRST CORUNO	CORUNO	0	0	0
LAST CORUNO	CORUNO	0	0	0

- CHAINED SEMAPHORE: (OPEN CHAINED USED)

-2	-1	0	1	2
FIRST CORUNO	CORUNO	0	VIRT.OP	FIRST VIRT.OP
LAST CORUNO	CORUNO	0	VIRT.OP	LAST VIRT.OP

2.1.4 Terminal Buffers 2.1.4

Used for terminal input/output. Allocated at init. A buffer for each possible terminal supported. Number of buffers = option i4+1. Buffer length = option i3 hws.

2.1.5 Sender Table 2.1.5

Each entry holds a description of a (maybe unknown) process sending messages to BOSS, and a semaphore to open when it occurs. Allocated at init. Number of entries = option e15.

When C.L. detects a message arrived from another process, this event is signalled to a waiting coroutine by making open chained of an operation to a specific semaphore. The possible senders and related semaphores are found in sender table.

Format of sender table:

ONE TABLE ENTRY: (length = 10 hws)

+0: Process descr. addr of a known sender	
+2: Address of semaphore to signal	
+4: OPER:	+0: CHAIN
+6:	+2: free, op.kind (=0)
+8:	+4: Mess. buf. addr of received mess. or = 1: mess in queue, but no coroutine waiting or = 0: no message in queue

Total number of sender table entries:

el5: = commandios + unknown sender (2)
+ pseudojobs + operator display (1);

C.L. performs, when message is received:

1. find sender in sendertable (last entry = unknown sender)
2. if oper already describes a mess ready for coroutine or coroutine not waiting (sem. value > -1) then skip the message (in the event queue) else:
   insert mess buf addr in oper.
   simulate open chained (oper)
   page 2 (activated coroutine): = abs addr of oper.
   get event (mess. buf addr);

The coroutine performs:

1. lock chained (SEM).
   On SEM, many different operations may arrive, also an operation describing an arrived message (as address of SEM is inserted in a sender table entry).

   "Message-operations" are distinguished by having fouth hw (OPER+3) = 0 (op.kind = 0), by convention.

   Having received such an operation, proceed with:

2. Copy message (maybe).
3. Take actions on message.
4. Send answer and set OPER(+4):= 0
   OPER(+2):= 0 (= free), maybe. (hw).
5. Proceed with other tasks, maybe.

   Now, a new message may be handled by C.L., but only one.
6. Return to 1.

2.1.6 Coroutine Table

The coroutine table contains coroutine descriptions. Allocated at init. Number of entries = option e14 (= number of coroutines).

In the code of C.L., the variable named f1 (called coruno) holds the abs address of current coroutine description (the abs address of f1 is in ext (26)). And the variable named f2 (called coruno code) holds the abs address of current coroutine code page.

Format of an entry in coroutine table, (see details below, length = 16 hws, option e12):

+0	CHAIN	
+2	STATE	
+4	COROUT.IDENT, TEST, REL.RETURN	
+6	PAGE0	
+8	PAGE1	Virtual
+10	PAGE2	address
+12	PAGE3	
+14	PAGE4	

The virtual addresses of pages should be interpreted: if it is > max core addr of BOSS, it is a virtual address. Otherwise it is some core address (e.g. CL will sometimes use page 2 for a core address).

On exit to a corutine, CL promises that all pages having a real virtual address are in core.

Detailed format of a coroutine description:

length = 16 hws (option e12)

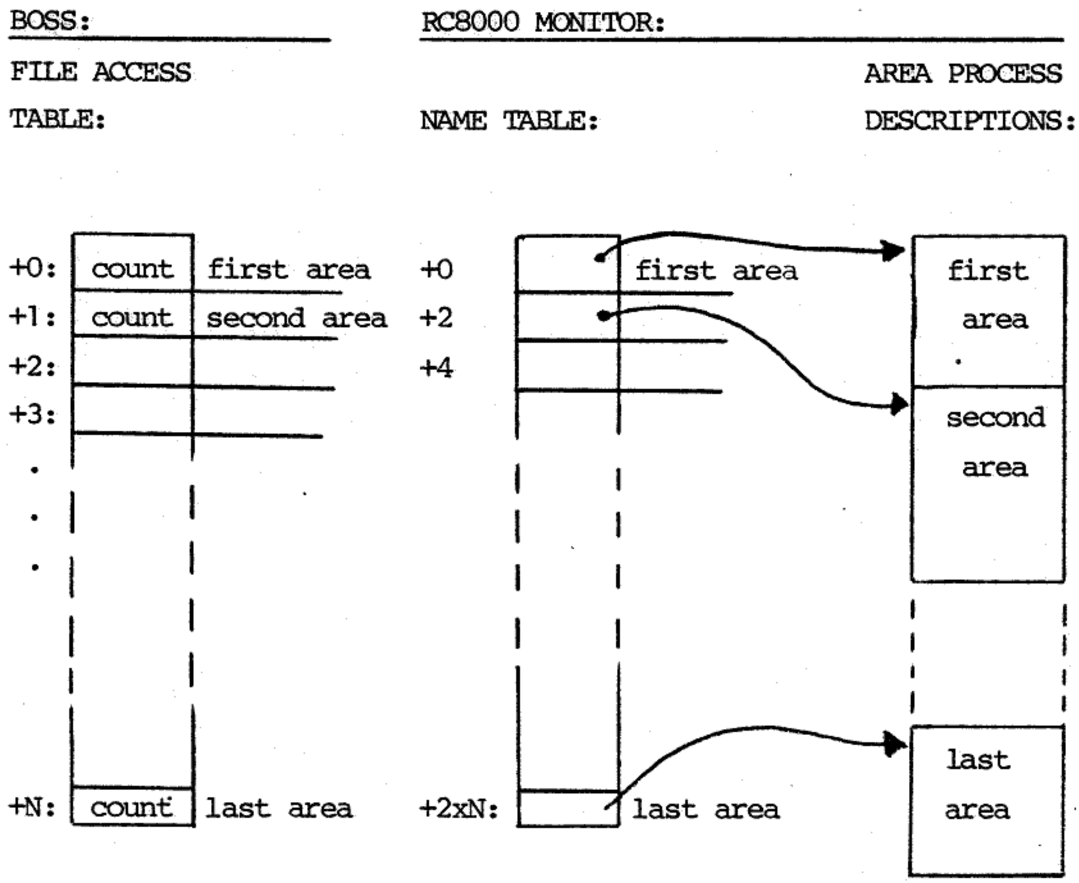
CORUNO-> (f1)

+0	CHAIN: - TO NEXT IN QUEUE - OR = 0 IF LAST IN QUEUE IF IN SOME QUEUE - OR UNDEF
+2	STATE: = 0 when running, and other cases = 1 after lock/lock chained, when SEM.VAL <= 0. (for testoutput of operation at exit). < FIRST of BOSS: waiting for answer (= message buffer addr). < LAST of BOSS: after lock/lock chained, when SEM VAL > 0 during check-pages. (= SEM ADDR) ~~>=LAST of BOSS:~~ ≤ 0 during openchained, waiting for virtual operation to be moved to core (to update the chains)
+4	COROUTINE IDENT <15 (No, 1-~~999~~ 511) + TEST PATTERN <12 (3 BITS, TESTOUTPUT) + REL. RETURN (The relative addr to return to in page 0, when exit is done to courutine)
+6	PAGE0: (VIRTUAL ADDR) CODE PAGE currently used by the coroutine
+8	PAGE1: (VIRTUAL ADDR) VARIABLE PAGE, variables used by the coroutine
+10	PAGE2: (VIRTUAL ADDR) SET BY CENTRAL LOGIC: = OPERATION after lockchained INTERNAL USE IN CENTRAL LOGIC: = VIRT: of last operation in queue during opench = 0 after send and wait/stop and wait = CORETABLE REF. OF BUFFER after send and wait fast-slow
+12	PAGE3: (VIRTUAL ADDR) SET BY COROUTINE
+14	PAGE4: (VIRTUAL ADDR) SET BY COROUTINE

2.1.7 File Access Table

The file access table is used to control the access from coroutines to bs files. It is closely coupled to the name table of the monitor.

Each entry in file access table is 1 hw, and corresponds to one area process description in the monitor (= number of entries). Allocated during init.



The file access table is indexed via the name table address (NTA), which is the absolute address of the word in name table pointing to the process description in question. NTA is set by C.L. procedure prepare access, which creates an area process.

"count" is counting number of simultaneous "openings" to the file described by the area process.

An "opening" is understood as the time between call of prepare access and terminate access.

A negative count will lead to bossfault 2. If count > 1, more than one coroutine (e.g. terminals) have opened to this specific file, and it must not be destroyed or overwritten.

2.1.8 Run Test Buffers 2.1.8

Allocated during init. 2 buffers, maybe only one (if option e7 = -1).

Each buffer is 512 hws (+ one dummy word). Used by C.L. procedures for testoutput. When a buffer is full (no room for next testoutput record), it is sent to the device.

The device may be a bs file name "bosstest" or a magtape station named "bosstest". In the latter case, option e6 = +1 is required.

The selection of magtape/bs file is done on basis of "reserve process". If this succeeds, it must be a peripheral process already existing (magtape assumed). Otherwise an area process is created.

2.2 Internal Queues in Central Logic 2.2

C.L. works on 3 queues, used in administration of the coroutines.

The queues are operated like semaphores and so the format is similar.

CORUNO denotes a coroutine description address.

- Active queue. Chains of coroutines ready to activate (internal name: f4).

```
f4 -> (-2 : no value of the queue)

      +0: FIRST CORUNO IN QUEUE (or = 0)
      +2: LAST CORUNO IN QUEUE (or = 0)
```

- Answer queue. Chain of coroutines waiting for an answer to a message (internal name: f3).

```
f3 -> (-2: no value of the queue)
      +0: FIRST CORUNO IN QUEUE (or = 0)
      +2: LAST CORUNO IN QUEUE (or = 0)
```

- Pager queue. Chain of coroutines, whose pages are not in core, and whose pages must be moved to core in order to proceed with some operation (internal name: f5).

```
f5 -> +0: VALUE
      +2: FIRST CORUNO IN QUEUE (or = 0)
      +4: LAST CORUNO IN QUEUE (or = 0)
```

2.3 Datastructures in Virtual Core 2.3

The datastructures not resident in BOSS core are placed in pages in the virtual core (drumcore/disccore), and moved to (and maybe from) the core by the pager coroutine (see chapter 5).

The datastructures placed on pages, are:

- Code pages. Used as page0 in coroutines.
- Variable pages. Used as page1 in coroutines.
- Operations. Chained to a semaphore, or in no chain (if delivered to a coroutine).
- Job file pages.
- Job description pages.
- Work pages, no general formats.

Pages are created by the initialization code in each module file (coroutine file), when activated by the loader.

2.3.1 Code Pages 2.3.1

Code pages contain code.

Format of a code page (values are set at EXIT to coroutine):

```
ABS.        ┌──────────────────────────────────────────────┐
PAGE0 ->    │ +0:  ABS. PAGE0 ADDR. (=ITSELF)              │
ADDR        │ +2:  ABS. PAGE1 ADDR. (VARIABLES)            │
(f2.)       │ +4:  ABS. PAGE2 ADDR. (USED BY C.L.)         │
            │ +6:  ABS. PAGE3 ADDR                         │
            │ +8:  ABS. PAGE4 ADDR                         │
            │+10:  CODEPAGE IDENT     (hw)                 │
            │+11:  REL. RETURN at exit from CL (hw)        │
            │        i.e.: rel.addr of the instruction     │
            │               to which C.L. jumped at exit.  │
            │                                              │
            │              (the rest contains code)        │
            └──────────────────────────────────────────────┘
```

Before exit to a coroutine, C.L. assigns all the fields mentioned, except CODEPAGE IDENT.

All code pages are created as "real" pages. Except for the pager, which is resident in core. But the head of the pager code has the same format.

2.3.2 Variable Pages 2.3.2

Contain variables, mostly dedicated to a specific coroutine. Used as page1.

There exist no conventions about format. Note that C.L. procedure w1-call stores virtual return address on page1:

```
      Page1:
     ┌───────────────────────────┐
+0   │ Virtual page0 addr        │
+2   │ Relative return in page0  │
     │                           │
     │                           │
     └───────────────────────────┘
```

2.3.3 Operations

Common conventions for format:

+0	chain (if chained)		
+2	"Free"-mark/banker op.code	HW	defines use and
+3	Op.kind	HW	contents of operation
+4	Answer Semafor addr (abs) (if sending coroutine expects an answer)		
	(defined by 2nd word)		

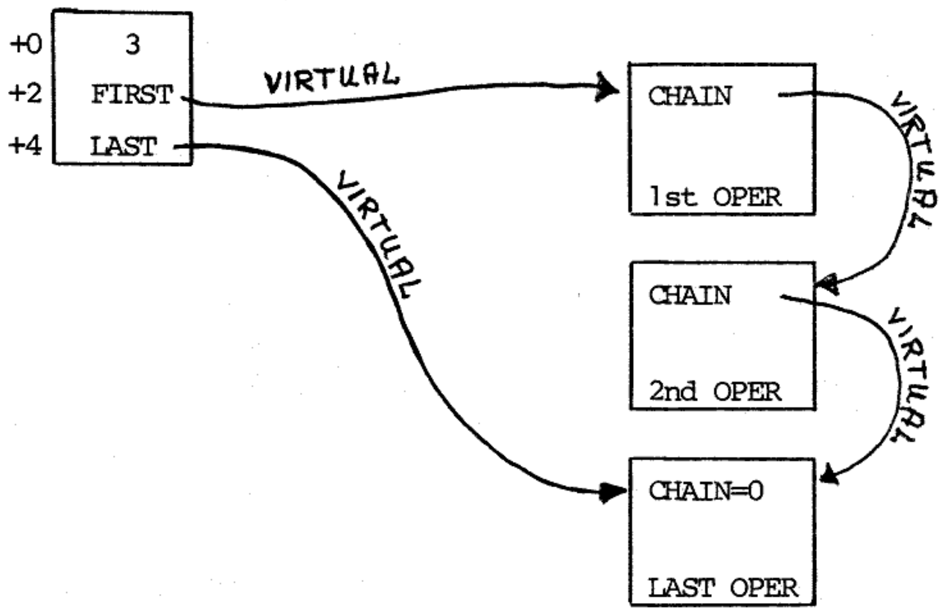
In an operation, C.L. only uses 1st word, and only when the operation is chained to a semaphore. The chain is a virtual address of the next in chain.

Note that the length of the operation is not part of the datastructure. It is defined at creation and not signalled later on.

Examples of semafor states:

- Semaphore with value 3 and chain (operations queued):

SEM.DESCR. (VIRTUAL) CORE:



- Semaphore with value 3 and no chain:

SEM.DESCR:

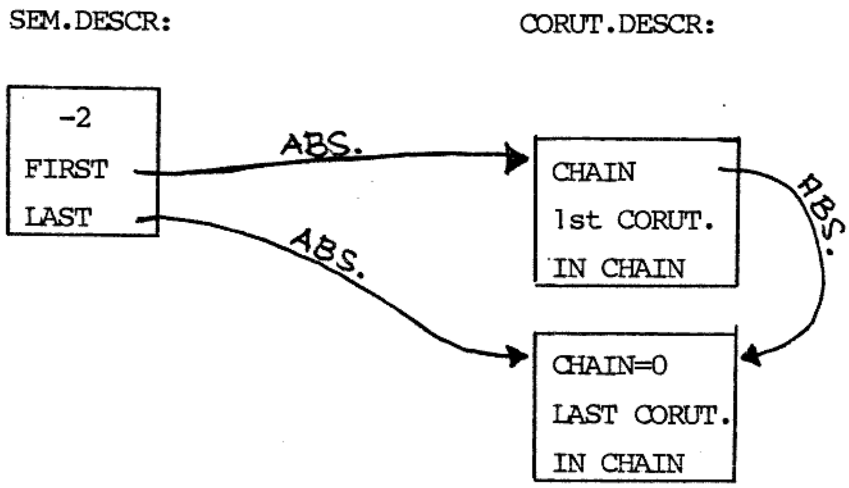
```
3
0
0
```

- Semaphore with value 0:

SEM.DESCR:

```
0
0
0
```

- Semaphore with value -2 (2 coroutines chained):



2.3.4 Job File Pages 2.3.4

This page is not a C.L. datastructure. It is created and used in the module terml.

One page for each commandio coroutine.

Layout of JOB FILE PAGE:

```
; sl 14.6.71            job file page           term1    ...2...

g0=k-4-i46                  ; begin page 1, job file page
                            ;     return virt and rel for commandio procs
d7=4, d8=4                  ;     param list and line buffer (i46 hw)
                            ;     also used as working area by command actions.
d34=k-2-g0, d35=d8-2        ;     last of line, base line

; variables for next block and attention:
d36=k-g0, 0                 ;     ext 177     : job descr page
d38=k-g0, g63: 53           ;     ext 221     : terminal reserve. binary semafore.
d39=k-g0, g64:              ;+2   ext 223     : attention semafore.
h. e13*i4+e13, 19   w.      ;
d40=k-g0, g65:              ;     ext 222     : terminal sender descr
h. 10*i4+10,   20   w.      ;
d41=k-g0, 0, r.5            ;     ext 217&300: terminal name, nta
                            ;                     <* the two externals are equivalent *>
d42=k-g0, g66: 150          ;     ext 178     : first addr of terminal buffer.

c. d42+2-e11-1
m.*** e11 too small
z.

; variables for next char and central actions:
d1=k-g0,  0                 ;     partial word
d2=k-g0, -1                 ;     ext 289     : special char. logout cause
d3=k-g0,  g20               ;+2   state. initially: start state, testing only.
d4=k-g0,  0                 ;     curr word. abs address in input buffer.
d5=k-g0,  0                 ;     top but.    -     -     -     -     .
d6=k-g0, 29<12+0            ;     ext 179     : console state < 12 + jobstate
; console state:    8 + ready*4 + user*2 + operator*1 + unlogged*16
; job state:   defined in tjob, io page
d37=k-g0, 0                 ;+2   conversational * 8
; d10 is used to count buffers returned with timer status

; variables for edit actions:
d9=k-g0,  d14               ;     top corr list. rel address within page 1.
d10=k-g0, 0                 ;+2   work. line number. timer count(from command input).
d11=k-g0, 0                 ;     work. return rel inside page.
d12=k-g0, d15               ;     curr corr segm. rel address within page 1.
d15=k-g0, 1000 001          ;                 first corr segm:   list of corr segments each
          r. 2*144          ;     ext 228         described by two words, thus:
; first word:    top corr list when the corr segment became full, large
; until it is full. second word:    virt address of corr segment.
d86=k-g0, 0               ;     ext 135     : start line for autoline
d87=k-g0, 0               ;     ext 136     : linedistance in autoline

; variables for next line and init line:
d16=k-g0, 0                 ;     ext 189     : work page. buffer for basis file.
d18=k-g0, 0                 ;     ext 188     : abs or page. relative on bulk file page.
                            ;                     (page 0-4 are indicated by value 0-8,
                            ;                      abs addr by: ext 155)
d27=k-g0, 0                 ;-2               last store rel. used by next line.
d19=k-g0, 0                 ;+0   ext 187     : store rel.             -     -      .
d20=k-g0, 0                 ;+2               return rel.              -     -      .
d21=k-g0, 0                 ;     ext 231     : segment number in basis file.
d22=k-g0, 0, r.5            ;     ext 218     : basis file name, nta
d26=k-g0                    ;                 logical status at alarm.
d33=k-g0, 0                 ;     ext 190     : line got. used by next line.
d28=k-g0, 0                 ;     next corr line
d29=k-g0, 0                 ;+2   next corr index
d30=k-g0, 0                 ;     used already from corr line.
d31=k-g0, 0                 ;                 next file line
d32=k-g0, 0                 ;+2   ext 230     : next file index
d25=k+2-g0, 0, 0            ;     ext 257     : return virt and rel from nextline etc.,
                            ;                     and from print catalog entry
```

Layout of JOB FILE PAGE (continued):

```
; sl 12.11.71              job file page          term1    ...3...

; variables for accounting and stdinterval
d43=k-g0,    0             ;-2   ext 494    : time logged in (seconds)
d44=k-g0,    0             ;+2   ext 226    : operations performed
d45=k+2-g0, 0, 0           ;     ext 227    : stdinterval

; variables for psjob, primary io:
d50=k-g0, g67:             ;     ext 175    : psjobque, chained semafor.
h. e13*i45,     19 w.      ;
d51=k-g0, g68:             ;     ext 176    : job in core, binary semafor.
h. e13*i45,     19 w.      ;
d52=k-g0, 0                ;     ext 180    : exhaust count.

                           ; job controlled printer:
d53=k-g0, 0                ;     ext 181    : rest room on page
d54=k-g0, 0                ;+2   ext 182    : job buffer index.
>>d55=k-g0, 0              ;+4   ext 183    : get current virt buffer addr
d56=k-g0, 0                ;     ext 184    : message addr
d57=k-g0, 0                ;     ext 185    : first of job (even)
d58=k-g0, 0                ;     ext 186    : top of job (even)

                           ; primary output:
d59=k-g0, 0                ;     ext 191    : rest room on page
d60=k-g0, 0                ;+2   ext 192    : job buffer index
d61=k-g0, 0                ;+4   ext 193    : virt buf
d62=k-g0, 0                ;     ext 194    : first buf byte
d63=k-g0, 0, r.5           ;     ext 195    : primary output file: name, nta
d64=k-g0, 0                ;     ext 196    :    -      -       - : length (segments)
d65=k-g0, g69: 54          ;     ext 197    : terminal full, simple semafor

                           ; card reader:
d66=k-g0, 0                ;     ext 198,450: prog state, start rel
d67=k-g0, 0                ;     ext 199    : curr virt address of card buffer
d68=k-g0, 0                ;     ext 200    : last received status of card reader
d69=k-g0, 0                ;     ext 201    : card length
d82=k-g0, 0                ;-2   ext 367    : free buffer semafore
d83=k-g0, 0                ;     ext 368    : full buffer semafore
d84=k-g0, 0                ;-2   ext 369    : virt first buffer addr
d85=k-g0, 0                ;     ext 370    : virt top buffer addr

                           ; terminal output:
d71=k-g0, 0                ;     ext 299    : state
d72=k-g0, 0,0              ;     ext 301,302: next byte, buf byte
d73=k-g0, 0,0              ;     ext 303,304: virt buf, appetite
d74=k-g0, 0                ;     ext 305    : out bytes

                           ; job controlled tape reader:
d75=k-g0, 0                ;     ext 446    : prog state
d76=d75+1                  ;     ext      447:    start rel
d77=k-g0, 0                ;     ext 448    : curr virt address of tape buffer
d78=k-g0, 0                ;     ext 449    : last received status of tape reader
d90=k-g0, 0                ;     ext 518    : job reader free
d91=k-g0, 0                ;     ext 519    : job reader full
d92=k-g0, 0                ;     ext 520    : first virt buffer
d93=k-g0, 0                ;     ext 521    : top virt buffer

                           ; card reader and reader:
d81=k-g0, 0                ;     ext 429    : next store in job input area

                           ; job controlled printer:
d79=k-g0, 0                ;     ext 474    : current paper type

d80=k-g0, g140: 81         ;     ext 306    : terminal input semafore

; variables for various commands
d88=k-g0, 0                ;     ext 134    : terminal user rights (privilege)
d70=k-g0, g100:            ;     ext 236    : commandio answer semafore
h. e13*i4+e13, 19 w.       ;
```

Layout of JOB FILE PAGE (continued):

```
; sl 14.6.71            job file page          term1    ...4...

; line table, correction list, related g-names.

d17=k-g0, 1000-000      ;     line table. contains one word for each
; segment in the basis file, specifying the line number for the first
; character on the segment. if the file is not terminated by an em-
; character, the top segment is considered containing the em-character.
; the table is terminated by one more word containing a large value.

d14=d17+2+2*i47         ;     corr list. one word for each correction
                        ; specifying the line number corrected.

g2 =d14+2+2*i48         ; end page 1, job file page. g2 = length

g12=g2, g13=510-i46     ;     max top corr list, max last on corr page
g14=d15+2*i44-2         ;     max current corr segm.
g15=8+4                 ;     required edit bits: conversational, ready.
g16=i46, g19=i3         ;     max line length, length terminal buffer

; the corrected lines are stored on correction segments, each
; segment composed like this:
; 0:     last on corr page (1 word, specifying last used on this page)
; 2:     corrected line (possibly some words containing the characters,
;                        one word containing line number,
;                        one word containing -length of corrected line)
;        corrected line, ...
```

2.3.5 Job Description Pages 2.3.5

This page is not a C.L. datastructure. It is described in a separate module text, and used in the modules: job, jobstart and procs.

It is created (as a page) in the module JOBSTART

One page for each pseudo-job.

Layout of JOB DESCRIPTION PAGE:

```
; pm 16.12.71                        boss 2, tpsjobdescr, ...1...

; definition of psjob descr page common for psjob start,
; lookup usercat and psjob end.
; versionid: 78 07 14,20

b. d160,   w.                    ;

f0:              0,0             ;    return point
d11=-f0.,        0, r.8*i49      ;+4  param
d144=-f0.,    0                 ; rel return 'set reader device no'
d146=-f0.,f9:h. r99*i45,19 w.;     abs ref permanent core for rb messages
d97=-f0.                         ;    top param
d12=-f0.,        0               ;    switch
d0=-f0.,   f6:   0               ;+2  jobfile
d3=-f0.,         0               ;+4  virt line buffer

d91=-f0., f1:   63               ;    psjob que
d93=-f0.,        0               ;    request line
d110=-f0.,f5:   64               ;    job in core
d96=-f0., f7:    0               ;    psjobindex * e13
d108=-f0.,f2: h. i45*10,20 w.;    sender table addr

d15=-f0.,        0               ;    tail:   size
                 <:disc:>        ;+2          docname
d105=-f0.,       0, r.7          ;+10         zeroes      d105: zero for set catbase (tail+6)

d87=-f0.,        0               ;+0  semafore operation:   chain
f8:          h.  2*e13, 19 w. ;+2  opcode:   skip 2 psjob buf sem
d82=-f0., f4:h.i45*e13,19 w.;+4  psjob buf
d14=-f0.,        0               ;+6  banker operation:   chain
f3:              0               ;+2: opcode, psjobnr
                                 ;+4: all, stations   ext 349: newjob operation (28 bytes)
                                 ;+6: disc
                                 ;+8: reader
                                 ;+10: converts, accounts
                                 ;+12: bufs,      areas
                                 ;+14: internals, suspends
                                 ;+16: drum
                                 ;+18: device
                                 ;+20:    -
                                 ;+22: size
                                 ;+24: keys,      priority
                                 ;+26: gross time
                                 ;+28: net time
                                 ;+30: arrival
                                 ;+32: max wait

                                 ;    d89   -   d89+18   is used for lookup entry
                                                         in convert from commandio

                                 ;    these variables are located in banker operation
d89=d14+10                       ;+0  message:   code
                                 ;+2  first
                                 ;+4  last
d86=d89+6                        ;+6  segment
d1=d86+2                         ;+0  name addr in cat proc
                                 ;+2  interval   -    -    -
                                 ;+4     -       -    -    -
                                 ;+6  length, function
                                 ;+8  rel return
```

Layout of JOB DESCRIPTION PAGE (continued):

```
; pm 7.12.71                          boss 2, tpsjobdescr, ...2...

d9=  d14 +34            ;     text shift, state save
d6=  d9   +2            ;+2   return from next line and set claims
d7=  d6   +2            ;     count
d8=  d7   +2            ;     addr
d10= d8   +2            ;     text addr
d5=  d10  +2            ;     work
d13= d5   +2            ;     command state (c15 job command must be
                                   read now, c38 claims not set,
                        ;     c16 claims set: only load command admitted
d16= d13  +2            ;     job param base
; d9 through d13 are used as scratch during convert from commandio too

; job parameters

d17= d16                ;     time of arrival of job * 2**(-13)
d23= d17 +4             ;-10  user name
d25= d23 +8             ;-2   user index
d20= d25 +2             ;     project no
d26= d20 + 4            ;     project base used in connection whith convert
                        ;     operations on netprinters described in catalog
                        ;     entries. not equal to maxbase in case of a userpool
d19= d26 +2             ;     job name

d18= d19 +8             ;     prog name, name table entry
d63= d18 +2             ;
d140=d18 +10            ;     no of conversational input lines
d70= d140+1             ;     tape table entries
d143=d70 +1             ;     terminal user rights
d61= d143+2             ;     usercat ref post address of user
d62= d61 +2             ;     no of useable private kits
d22= d62 +2             ;     usercat refdiscrest claim key 3
d60= d22 +2             ;     usercat ref drumrest claim key 3

d58= d60 +2             ;     discrest key 1
d69= d58 +2             ;                  2
d55= d69 +2             ;                  3
d59= d55 +2             ;     drumrest key 1
d67= d59 +2             ;                  2
d56= d67 +2             ;                  3

d28= d56 +2             ;     first of private kit table
;        +0             ;     kit name
d29=     +8             ;     usercat ref rest claim key 3
d57=     +10            ;     job claims
d113=     12            ;     slice length
d80= d28 +14*i30+2      ;     top private kit table
```

Layout of JOB DESCRIPTION PAGE (continued):

```
; pm 1.11.71                      boss 2, tpsjobdescr, ...3...

d75=d80                 ;     conversation
d102=d75 +1             ;      degree of information wanted:
                        ;       0  maximal information  <:no:>
                        ;       1  minimal information  <:yes:>
d77= d102+1             ;      core lock time
d132=d77 +2             ;      priority
d134=d132+1             ;      preserve catalog  0 catalog is cleaned <:no:>
                        ;                        1 catalog is preserved <:yes:>
d138=d134+1             ;      waiting time deliberately swopped out
d148=d138 +2            ;      link

d41= d148+2             ;      keys
d49= d41 +1             ;      stations
d43= d49 +1             ;      mounts
d51= d43 +1             ;      tapes
d68= d51 +1             ;      converts
d31= d68 +1             ;      accounts
d45= d31 +1             ;      output slices
d71= d45 +1             ;      suspendings
d53= d71 +1             ;      time * 2**(-13)   first storage address
d47= d53 +2             ;      size              top      -     -
d35= d47 +2             ;      bufs
d33= d35 +1             ;      areas
d39= d33 +1             ;      internals
d37= d39 +1             ;      func
d66= d37 +1             ;      max wait * 2**(-13)   16 protection reg
;    d66 +1             ;                            17 protection key
d21= d66 +4             ;      max
d24= d21 +4             ;      std
                        ;      user interval
d30= d24 +6             ;      device
d81= d30 +4 - d80        ;.
d76= d81 +d75           ;      max conversation
d103=d81 +d102          ;      max degree
d78= d81 +d77           ;      max core lock time
d133=d81 +d132          ;      max priority
d135=d81 +d134          ;      max preserve catalog
d139=d81 +d138          ;      max waiting time deliberately swopped out
d147=d81 +d148          ;     max link
d42= d81 +d41           ;      max keys
d50= d81 +d49           ;      max stations
d44= d81 +d43           ;      max mounts
d52= d81 +d51           ;      max tapes
d72= d81 +d68           ;      max converts
d32= d81 +d31           ;      max accounts
d46= d81 +d45           ;      max output
d73= d81 +d71           ;      max suspendings
d54= d81 +d53           ;      max time
d48= d81 +d47           ;      max size
d36= d81 +d35           ;      max bufs
d34= d81 +d33           ;      max areas
d40= d81 +d39           ;      max internals
d38= d81 +d37           ;      max func
d27= d81 +d66           ;      respite
d65= d27 +2             ;      max devices
```

Layout of JOB DESCRIPTION PAGE (continued):

```
; pm 27.12.72                              boss 2, tpsjobdescr, ...4...

d88= d65 +6            ;    psjob status:  1<0  reader reserved
                       ;                   1<1  change of jobname after replace
                       ;                   1<2  operator message when enrolled
                       ;                   1<3  online job
                       ;                   1<4  error message pending
                       ;                   1<5  card job
                       ;                   1<6  internal
                       ;                   1<7  convert claim
                       ;                   1<8  offline job
                       ;                   1<9  convert from commio
                       ;                   1<10 replace job
d90= d88 +1            ;    output length
d92= d90 +1            ;    paper tape count
d95= d92 +1            ;    swop area length (slices)
d116=d95 +1            ;    card file no

d84= d116+2            ;    expected finis time (sec after midnht)
d119=d84 +2            ;    hostident
d120=d119+2            ;    dh. linkno < 12 + dh. hostno
d122=d119+4            ;    dh. hostid
d121=d122+2+2          ;    name of primout printer (last of dbword)
                       ;      first word = 0  => std printer
                       ;        -        = 1  => some rb-printer
d123=d121+2            ;    device kind
d131=d123+2            ;    device number of reader
d112=d131+2            ;    primary output name and name table addr
d100=d112+10           ;    job file name and name table addr
d101=d100+10           ;     -    -    -   interval
d2  =d101+4            ;    total net time (0.8 sec)
d79 =d2  +2            ;    cputime used(0.8 sec)
d74 =d79 +2            ;    convert claim
d83= d74 +2            ;-2  length of input area (segments)
d114=d83 +2            ;    op buf addr for answer commandio
d64 =d114+4            ;    returnpoint for load and pagejump to set claims
d118=d64 +2            ;    cardfilecount
                      ;+2: card block count (segments)
                      ;+4: card reader queue sem
                      ;+6:  -      -    first virt buffer
                      ;+8:  -      -    top     -     -
                      ;+10: -      -    free  sem
                      ;+12: -      -    full    -
d141=d118+12           ;    start of reader table:
                       ;+2  tape reader segment count
                       ;+4    -     -     queue sem
                       ;+6    -     -     first virt buffer
                       ;+8    -     -     top virt buffer
                       ;+10   -     -     free  sem
                       ;+12   -     -     full sem
d142=d141-d87
d85 =d141+14           ;    temp drum, disc

d98= d85 +2            ;    rest mounts
d99= d98 +1            ;    rest suspends
d111=d99 +1            ;    rest converts, rest accounts
```

Layout of JOB DESCRIPTION PAGE (continued):

```
; re 5.2.75                          boss 2, tpsjobdescr   ...5...

d104=d111+2          ;     finis cause
d115=d104+2          ;+0:  net run left at finis
                     ;+2:  n r l at run time exceeded
                     ;+4:  n r l at dump
                     ;+6:  n r l at provoke before dump
                     ;+8:  n r l at high priority exceeded
                     ;+10: n r l at  corelock exceeded
d136=d115+12         ;-2  summa waiting times
d137=d136+2          ;    actual waiting time.rel return in low/high prio start
d106=d137+2          ;     proc descr addr of job
                     ;d107-6: harderror swopin op: chain
                     ;d107-4:                     free, opcode=16
                     ;d107-2:                     logical status
d107=d106+8          ;    swop area spec:    name
                     ;+2                      -
                     ;+4                      -
                     ;+6                      -
                     ;+8  name table addr
                     ;+10 timer op:    chain
                     ;+12              free, opcode
                     ;+14              net run time left
d4=d107+16           ;    line huffer base
d94=d4+i46           ;    line buffer last
d117=d94+2           ;    save array for go parameters
d109=d117+d88-d18    ; ,  top jobdescr

i.
t.
```

2.4 Time Representations 2.4

Time representations are always based on the monitor clock in address 110 (and 108).

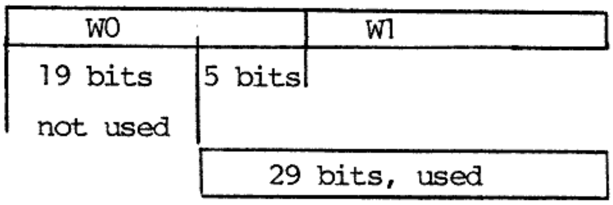
It is loaded like this: dl W1 110

The registers now will contain:

W0	W1
24 bits	24 bits
48 bits real time clock in units of 0.1 msec.	

Time in testoutput header

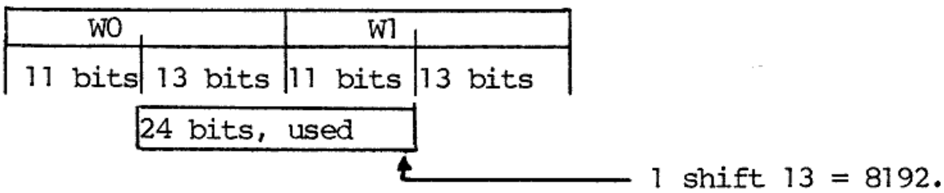
Algorithm in pseudo-algol
second header-word: = time:=
((montime - bossstarttime) extract 29) //100;
After subtract of boss start time:



time is in units of 0.01 sec.
Max. value is 5368709, stored in 24 bits.

Units of 0.8192 seconds

Algorithm in pseudo-algol:
time:= (montime shift (-13)) extract 24;

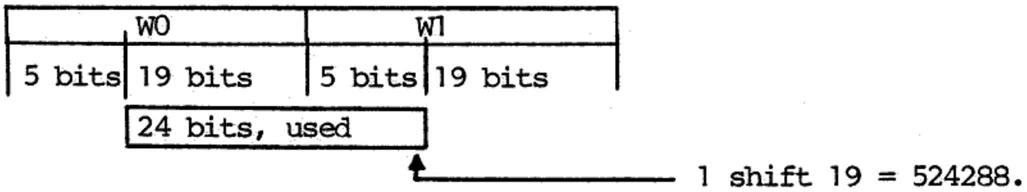


time unit is: 8192 x 0.0001 = 0.8192 sec.

<u>Shortclock in catalog entry</u>

Algorithm in pseudo-algol:

time: = (montime shift(-19)) extract 24;



time unit is: 524288 x 0.0001 = 52.4288 sec.

3. CENTRAL LOGIC PROCEDURES 3.

The functions of C.L. procedures are described below. The return from the C.L. procedure to the coroutine is in most cases done as an "exit to coroutine". Thereby it is checked that all pages stated in the page list of the coroutine description are present in core (page core). So, before call, the coroutine may change its page list, and in the same call get some other pages (note that this is not working in procedures marked with no delay).

At "exit to coroutine", C.L. promises that all pages, mentioned in the coroutine description, are in core. And on page0 itself, the corresponding abs. addresses of the pages is placed (see subsection 2.3.1).

Note that if an operation is delivered as page2, this may be a page (comming from virtual core), but it could also point to e.g. sender table (if it signals that a message has arrived).

3.1 Survey of Central Logic Procedures 3.1

NAME	EXTERNAL	CALL W0	CALL W1	CALL W2	CALL W3	RETURN W0	RETURN W1	RETURN W2	RETURN W3	RETURN PAGE2	DELAY
OPEN	4			SEM	RETURN	UNCH.	UNCH	Ø	PAGE1	UNCH	N
OPEN CHAINED	6		OP	SEM	RETURN	Ø	Ø	Ø	PAGE1	0	M
LOCK	3			SEM	RETURN	Ø	Ø	Ø	PAGE1	UNCH	M
LOCK CHAINED	5			SEM	RETURN	Ø	OP	Ø	PAGE1	OP.PAGE	M
GET PAGES	7				RETURN	Ø	Ø	Ø	PAGE1	UNCH	M
PAGE JUMP	8			PAGE0	REL	Ø	Ø	Ø	PAGE1	UNCH	M
(W1-) CALL	25		RETURN	PAGE0	REL	Ø	Ø	Ø	PAGE1	UNCH	M
W0 - CALL	34	STORE	RETURN	PAGE0	REL	Ø	Ø	Ø	PAGE1	UNCH	M
WAIT ANSWER	35			MESSBUF	RETURN	STATUS	ANSW	Ø	PAGE1	0	Y
SEND AND WAIT	1		MESS	NAME	RETURN	STATUS	ANSW	Ø	PAGE1	0	Y
SEND AND WAIT FAST	2		MESS	NAME	RETURN	STATUS	ANSW	Ø	PAGE1	BUFPAGE	Y
SEND AND WAIT SLOW	9		MESS	NAME	RETURN	STATUS	ANSW	Ø	PAGE1	BUFPAGE	Y
STOP AND WAIT	30			NAME	RETURN	STATUS	ANSW	Ø	PAGE1	0	Y
CLEAR CORE	11		PLACES		RETURN	Ø	Ø	Ø	PAGE1	UNCH	Y
PREPARE ACCESS	32	LOW.BASE	UP.BASE	NAME	RETURN	LOW.AREA	UP.AREA	ACCESS	PAGE1	UNCH	N
TERMINATE ACCESS	33			NAME	RETURN	LOW.AREA	UP.AREA	ACCESS	PAGE1	UNCH	N
PRIV-OUT	21	HEAD1	TAIL		RETURN	Ø	Ø	Ø	PAGE1	UNCH	N
JD-1	-	W0	W1	W2	W3	UNCH	UNCH	UNCH	UNCH	UNCH	N
JD-XXX	-	LENGTH	TAIL			UNCH	UNCH	UNCH	UNCH	UNCH	N

candrey

LEGEND TO SURVEY OF C.L. PROCEDURES

SEM - Abs address of semaphore (in semaphore table)
RETURN - Abs address of return-jump. Must be inside current page0.
PAGE1 - Abs address of first word of page1
OP - Abs address of operation
MESS - Abs address of 8 words to send as message
NAME - Abs address of 5 words: name of receiver + NTA of receiver
ANSW - Abs address of 8 words containing the answer
MESSBUF - Abs address of message buffer sent
STORE - Abs address of word where virtual return is stored.
UNCH - Unchanged
Ø - Undefined
STATUS - Logical status of answer: if result = 1 then (2+answ(1)) else 1 shift result

BUFPAGE - Address of the page used as buffer (virt/abs)
OP.PAGE - Address of the page containing the operation (virt/abs)
PAGE2 - Page 2 address: in coroutine descr (virt) and on page0 (abs)
PAGE0 - VIRT address of the wanted new page0
REL - Relative adddress to exit to on the wanted new page0
LOW.BASE - Lower limit of catalog base to lookup the area
UP.BASE - Upper limit of catalog base to lookup the area
LOW.AREA - Lower limit of the found area
UP.AREA - Upper limit of the found area
ACCESS - Total no of accesses to the area (accesscount)
HEAD1 - First word of head: length < 6+kind
TAIL - Abs address of the words to move to tail of record
EXTERNAL - No of the external variable holding address of the procedure
DELAY - Tells if the C.L. delays the coroutine. Y=yes, N=no, M=maybe
XXX - An integer 201 <= XXX <= 263. Testoutputkind will be XXX-200
PLACES - Core places to be cleared; first<12 + top

3.2 Procedure Descriptions 3.2

3.2.1 Open 3.2.1

Release a semaphore without chain (operation):

```
sem:= sem+1;
if sem <= 0 then activate another coroutine,
                 waiting for the semaphore.
                 (it is chained to active queue).
```

Returns immediately (no delay).

3.2.2 Open Chained 3.2.2

Releases a semaphore with a chained queue of operations, i.e. the operation pointed out is turned over to (chained to) the semaphore, whose value is now increased by one.

The operation is chained as the last of the operations queued at the semaphore.

The operation must be in core (pointed out by an abs addr), and must be part of a virtual page.

In the operation itself,

- first word (addr+0) is used by C.L. for chain. (Virtual addr of next in chain)
- fourth hw (addr+3) should be <> 0 (as 0 means a message).

Calling coroutine may be delayed. At least the chain from a preceding operation in the queue must be altered, and this requires the corresponding page in core.

Function:

The semaphore address is temporarily saved in the chain of the operation. The address of the operation is transformed from absolute to virtual, a writing bit is added and this virtual address is saved in the state of the coroutine. Page2 of the coroutine is set to the last operation (or coroutine) chained to the semaphore.

Now the pages are checked and afterwards the page pointed to by the state, the operation, is checked. If any coroutine is waiting for the semaphore (value < 0) the first coroutine is taken out of the chain, the operation is inserted as page2 of this coroutine description and this coroutine is put in the active queue. The semaphore value is incremented by one and the state and page2 of the delivery coroutine are cleared.

If no coroutine is waiting it is checked that no other coroutine has succeeded in sending an operation to the same semaphore during a paging, if any, ordered by check pages. In this case the last-pointer of the semaphore differs from page2 of the coroutine, the page2 is set to the new last-pointer and the algorithm is repeated from the point of page-checking.

If no other coroutine has interfered, then the operation is inserted into the chain of the semaphore, the value of which is increased by one and the state of the coroutine is cleared.

3.2.3 Lock, Lockchained 3.2.3

These procedures are identical, as the same entry in C.L. is used for both.

Purpose:

- lock: wait for a semaphore without chain (operation):

  sem: = sem -1;
  if sem < 0 then wait for activation

- lock chained: wait for an operation chained to the semaphore:

  sem: = sem -1;
  if sem < 0 then wait for activation
  else deliver first operation on semaphore.

Description:

if sem <= 0 then wait for activation
else (sem is >= 1)
    sem:= sem -1;
    if any operations are chained to the semaphore
    then deliver first operation in queue;
    (even in this case the calling coroutine maybe delayed).

Note that the type of the semaphore is determined by the open operations performed on the semaphore:

- open makes the semaphore unchained (simple)
- open chained makes the semaphore chained.
- open (simple) leaves the semaphore as it is (chained or unchained) i.e. if you use open (simple) on a chained semaphore the chain fields are not touched)

In an operation delivered,

- first word (addr+0) contains the new semaphore value.
- second word (addr+2) contains by convention

  free, op.kind

  op.kind = 0 if the operation signals a message arrived.

Function:

If the semaphore is not available (value <= 0) the state of the coroutine is cleared, the semaphore value is decreased and the coroutine is put into the semaphore chain.

If the semaphore is available (value >= 1) then the first operation of the semaphore chain (empty if not chained) is placed as page2 of the coroutine and the semaphore address as state.

Then the pages are checked and after this possibly time consuming operation the semaphore is checked again; if it is not available any longer or if the first operation in the chain now differs from the one which was found before the pages were checked, then some other coroutine has locked the semaphore and snatched away the operation during the paging which must have taken place and the algorithm is repeated from the beginning.

When the pages are ok and no other coroutine has interfered, the semaphore value is decreased and the operation is taken out of the semaphore chain (only if chained).

3.2.4 Get Pages 3.2.4

The pages (page0-page4) described in coroutine description (coroutine table), are moved to core (if not already present), i.e.: check that all the pages of the coroutine are in core. The coroutine continues when this is the case.

3.2.5 Page Jump 3.2.5

The calling coroutine wants to jump to another code page (page0), as indicated in W2-W3.

Note that no return address in the old code page is saved.

The page0 addr is renewed, and the coroutine continues when its pages are ready (in core).

3.2.6 W1-Call (or just Call) 3.2.6

The calling coroutine wants to jump to another code page (page0), as indicated in W2-W3.

The return address (in W1) is stored in the first two words of (current) page1, thus:

page 1 addr+0: virt (current) page0
page 1 addr+2: rel return in page0 (W1 - abs addr of page0)

Now page0 addr is renewed, and the coroutine continues when its pages are ready.

3.2.7 W0-Call 3.2.7

The calling coroutine wants to jump to another code page (page0), as indicated in W2-W3.

The return address (in W1) is stored in the abs address pointed out by W0:

addr(W0) -2: virt (current) page0
addr(W0) +0: rel return in page0 (W1 - abs addr of page0)

Now page0 addr is renewed, and the coroutine continues when its pages are ready.

3.2.8 Wait Answer 3.2.8

The coroutine asks to wait until answer arrives to a message already sent by the coroutine in the message buffer pointed out by W2 (MESSBUF).

Note that the message sent <u>must not</u> - if it is an I/O message - point out a data buffer in some of the pages (virtual core).
(in that case use 3.2.10 instead)

If I/O, the data buffer must be in permanent core.
Used by transmit.
The coroutine is queued to wait for the answer.

3.2.9 Send and Wait 3.2.9

The coroutine wants to send a message and wait for the answer.

The message to send is in addr (W1).

- if the message is an I/O mess, it <u>must not</u> point out a data buffer in some of the pages (virtual core). (Use 3.2.10 instead)
- If I/O, the data buffer must be in permanent core.
- page2:= 0;

- send message;
- the coroutine is queued to wait for the answer;

3.2.10 Send and Wait Fast, Send and Wait Slow 3.2.10

The coroutine wants to send a message and wait for the answer. Data are on a virtual page.

The message to send is in addr (W1).
- The message will normally be an I/O mess, and it contains abs. addresses pointing out a data buffer in some of the pages (virtual core).
- The page in question must be in core at call time, and now this page is marked (must stay in core until answer arrives).
- page2:= virtual addr of page containing data buffer.
- send message;
- the coroutine is queued to wait for the answer;

Function:
The procedures are used in connection with input/output to or from a virtual buffer. The corresponding bufferbit is set in the core table entry for the buffer, otherwise it works as send and wait (note input/output to or from a core store buffer is accomplished by means of the normal send and wait). The difference between fast and slow is solely reflected in the order in which the pager selects its victims (i.e. core places to be cancelled).

3.2.11 Stop and Wait 3.2.11

The coroutine wants to perform stop process (a job process).

- page2:= 0
- stop process (monitor 60)
- the coroutine is queued to wait for the answer.

3.2.12 Clear Core 3.2.12

The purpose is to reserve a part of core store before swop-in of a job process.

Works as get pages but ~~the coroutine has in advance replaced its page4-addr in coroutine description, thus:~~ in addition the segment places pointed out by w1 will be cleared, and their description in core table will be changed to describe a job place.

~~page4: +0: - first segm place (hw)~~

~~+1: + top segm place (hw)~~

w1: first segm place < 12 + top segm place

Used by banker. Must not be used by other coroutines

The pager is activated (put in active queue).

The calling coroutine is put in the pager queue, and the pager will clear the pages occupying the core places pointed out.

3.2.13 Prepare Access 3.2.13

The calling coroutine wants to access (read and/or write) a bs file. W0, W1 contains the "lookup-base" from where to find the file and create area process. Calling coroutine is not delayed.

2 return jumps exist:

1) return to W3 (from call):

    No entry visible, the registers are:
    W0:= result of create area process
    W1, W2:= undefined
    W3:= page1 addr
    page2:= unchanged.
    In name area (addr(W2)), result of create area process is stored instead of name table address.

2) return to W3+2:

    Entry found, area process created.
    Registers set as described in the survey.
    In name area (addr(W2)), name table address is set (in addr(W2)+8).

The catalog base (of BOSS) is set to the base of the area process.
Access count is increased by 1 (in access table).
(if new value is < 0, bossfault 2 occurs).

3.2.14 Terminate Access 3.2.14

The calling coroutine wants to terminate (a previously prepared) access to a bs file.

Access count is decreased by 1 (in access table).
(if new value is < 0, bossfault 2 occurs).
The catalog base (of BOSS) is set to the base of the area process.
If the (new) access count = 0, remove process is called.

The coroutine is not delayed.

3.2.15 Priv-Out 3.2.15

Private testoutput from the coroutine.
The C.L. procedure coruno output is called.
The coroutine is not delayed.
The testoutput record has the format:

```
+0   | length<6+kind    |      length: length of tail in hw.
+2   | time             |              (max 502 hw)
+4   | coroutine ident  |
     |------------------|
+6   |                  |  }   tail: 'length' hw,
     |                  |  }         copied from addr(W1) and on.
     |__________________|  }
```

3.2.16 JD-1 3.2.16

The coroutine wants testoutput of registers.

The coroutine executes the (illegal) instruction jd -1, which provokes a 'break0'.

The interrupt routine checks, that this was the case, generates at testoutput record containing registers and instruction counter, and continues with re-established registers in the instruction following the jd-1.

The coroutine is not delayed.

jd-2xx
jd-3xx

3.2.17 JD-XXX 3.2.17

The coroutine wants to generate private testoutput (as PRIV-OUT). This facility is just a simpler way of implementing it.

The registers (W0, W1) are assigned in advance, with W0 = tail length, W1 = first (tail). XXX must be an integer, 201 <= XXX <= 263. Testoutput kind will be XXX-200.

The coroutine executes the (illegal) instruction jd-XXX, which provokes a 'break0'.

The interrupt routine checks that this was the case, sets W0 = length < 6 + kind, W1 = first (tail), calls C.L. procedure coruno output and continues with re-established registers in the instruction following the jd-200.

The coroutine is not delayed.

4. LOADER

The loader in BOSS is a 2-pass loader/linker.

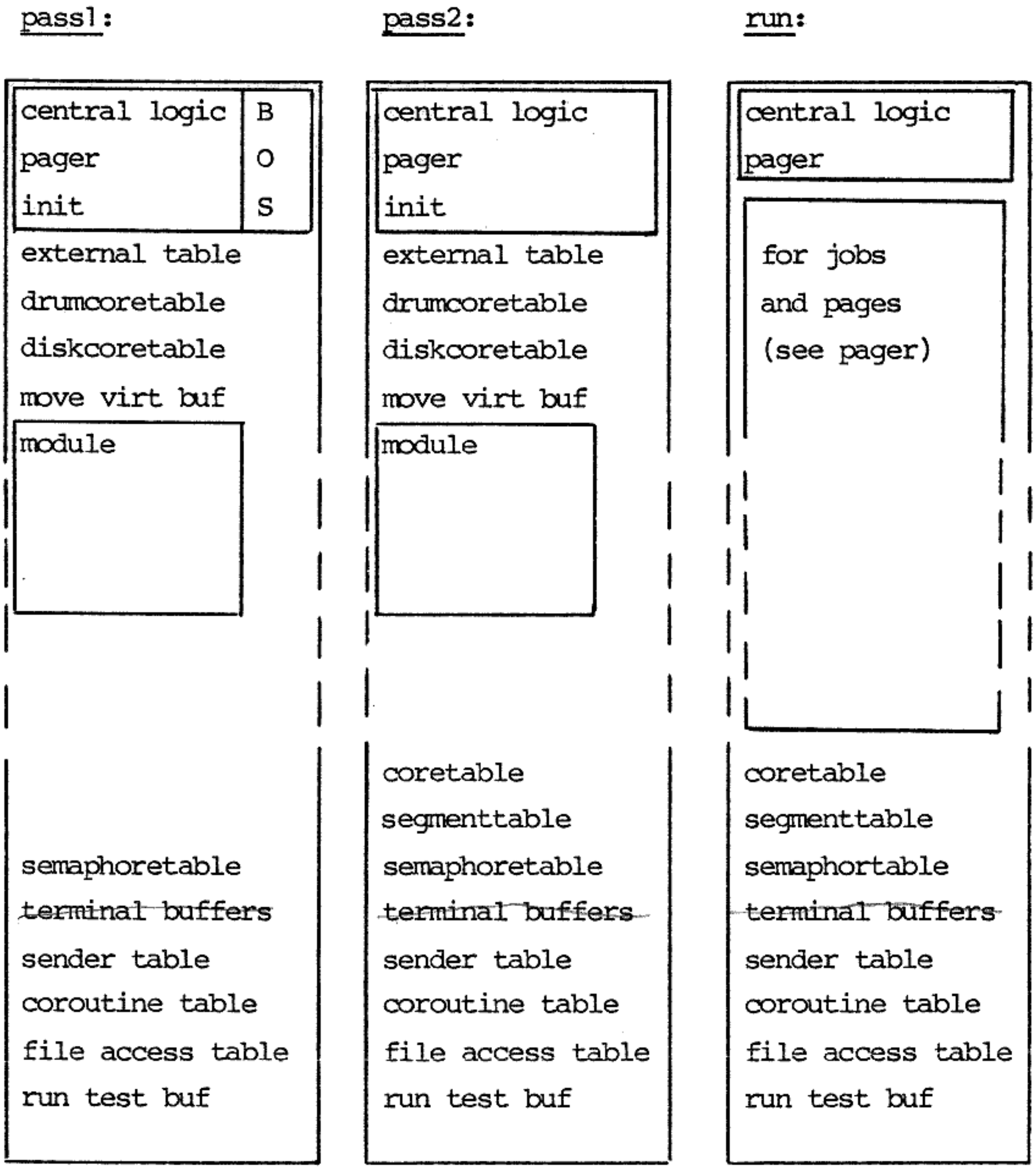
All the module files are loaded twice and executed. This execution is the initialization of the module, and it transforms the module into one or several code pages, variable pages, creates coroutines, semaphores, operations etc.

Information used in several modules is passed from module to module via the external table. It contains abs addresses (e.g. of C.L. procedures) virtual addresses etc. All values are initially zero and only a few are defined before first loader pass.

The majority of the externals are defined by the modules. They are in turn used by other modules that may have been loaded already. This is the ~~basis~~ main reason for the 2 passes.

4.1 Core Layouts 4.1

The figure shows the core utilization during the two loader passes and during run.



4.2 Init and Load, Step by Step 4.2

The following description covers all the initialization performed in the module bos, including the two loader passes.

step 1. If FP is present in the BOSS core, all the code is moved, so the FP code is overwritten.

step 2. Already existing internal processes with BOSS as parent are left unchanged i.e. the last address of BOSS core is set below the smallest of these core processes.

step 3. Allocate run test buffer(s).

step 4. Get name of main console.

step 5. Remove all area processes.

step 6. Connect testoutput medium: bosstest (bs file or magtape).

step 7. Zero-fill all BOSS core after the bos module.

step 8. Overwrite bosstest to zero-fill all segments (if bs file).

step 9. Allocate core used during loader:
- external table
- drumcore table
- disccore table
- buffer for move to virtual.

step 10. Allocate (some of) common resident tables:
- file access table
- coroutine table
- sender table
- terminal buffers
- top of semaphoretable.

step 11. Assign externals (external 13, 14, 24 dummy)

step 12. First testoutput: - installation name record
- bos load record.

step 13. "Next pass": Assign externals (to initial values)
18 = post record of coroutine table
19 = post record of semaphor table
20 = post record of sender table.

step 14. Load 9 module files, one by one:
step 14.1 load next module file
step 14.2 testoutput of load record
step 14.3 link the module (see subsection 4.3.1)
step 14.4 Jump to initialization code of module file (loader procedures called from module: see section 4.3).

step 15. Assign externals, that will be active in pass2:

13 = move to virtual

14 = simulate lock

24 = put in active queue.

step 16. Adjust all virtual disc addresses in external table ~~(all externals < 0),~~

and virtual addresses in coroutine table.

step 17. Allocate and initialize - segment table

- core table.

step 18. Create and overwrite drumcore and disccore.

step 19. Proceed with pass2: Step 13 and 14.

step 20. Output external table to testoutput.

step 21. In file access table, access count is set = 1 for the files:
- bosstest
- drumcore
- disccore
- usercat
- catalog.

step 22. If BOSS is started in monitor mode and monitor mode is not wanted (option e16) and the machine is RC4000 (option e80) then the protection key is changed.

step 23. On RC8000: Ensure, that BOSS has only write access to its own core (mode0 removed).

step 24. Jump to C.L. entry, where first coroutine in active queue is activated.

4.3 Loader Procedures 4.3

The initialization code in a module calls some procedures to handle the common datastructures.

4.3.1 Algorithm for Linking a Module 4.3.1

Before exit to a module file, the link algorithm transfers the current value of externals to places in the module itself.

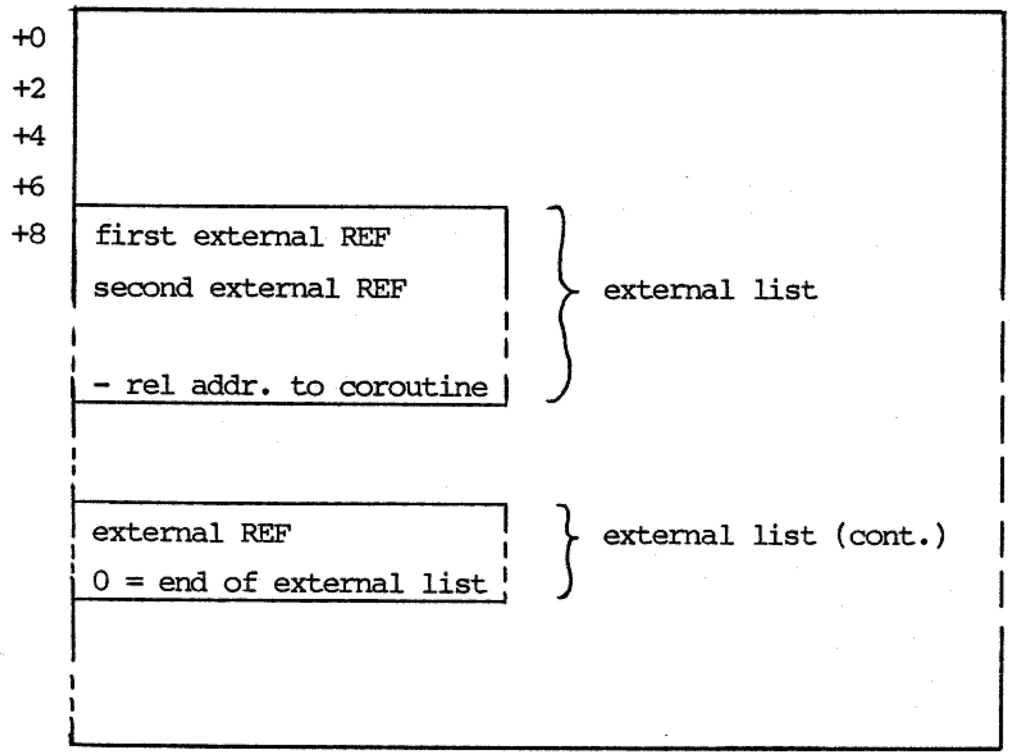
The linking is also used for reserving core in semaphore-, coroutine- and sendertable.

The module contains an external list. This is a list of relative addresses (REF) meaning:

"in the address relative to this is found an external number".

The list starts in module start +8 (see section 4.4)

MODULE:



A negative value means: continue in rel.addr - ref.
A zero means: end of external list.

Now, the list is scanned, and each positive ref. is interpreted, thus:

- REF is odd:
  In the hw addressed (will always be the right half of a word), the external number is found. Current value of this external is inserted in the hw.

	hw	hw		hw	hw
if found:	//////	305	it becomes	//////	ext(305)

- REF is even, and the word addressed contains the value 10:

  Reserve in semaphoretable (usually more than 4095 hws).

  In the word preceding the addressed word is found the number of hws to reserve.

  Ext(19) is decreased by number of hws and stored in the word addressed.

  (meaning: first address of just reserved core).

  Ext(10) is increased by number of hws.

if found:	-2	50	, 50 hws are reserved:
	+0	10	ext(10): = ext(10) +50;
			ext(19): = ext(19) -50;

it becomes:

	-2	~~50~~
	+0	~~ext(19)~~

- REF is even:

  In the word addressed, the external number is found. Current value of this external is inserted in the word.

if found	305	it becomes	ext(305)
if found	-10	it becomes	ext(-10)

- Special use of external 18, 19, 20:

  The left hw of the word addressed is used as "number of hws to reserve", and the external (number in right hw) is decreased before stored in the word addressed. So this value means: first address of just reserved core.

	hw	hw	
if found:	300	18	300 hws are reserved: ext(18): = ext(18) - 300,

and it becomes:

ext(18)

ext(18) reserves in coroutine table
ext(19) reserves in semaphore table
ext(20) reserves in sender table.

4.3.2 Set Externals 4.3.2

The abs addr of entry to the procedure is in ext(40).
The procedure is used to assign new values to externals. Called from the initialization code of the module.

Call:	Return:
W0 = irr	W0: = unchanged
W1 = irr	W1: = unchanged
W2 = irr	W2: = unchanged
W3 = -2 + abs.addr. of start of list	W3: = undefined
	Return jump is made to the address just after the list.

The list consists of a number of entries. Each entry is 2 words, the first is the new value, the second is external number. The last entry must be 0, -1000.

Example of call:

```
   external list:
   g13., g14., ...
   ....
   jl. w3 (2)          ; set externals:
g13:         40         ; here abs addr of entry is placed.
   100 ,     304        ; means set ext(304): = 100
    e2 ,      -7        ; means set ext(-7): = option e2
g14: 26<12+19           ; here is placed abs addr of first word of
                        ; the 26 hws which are reserved in the
                        ; semaphore table.
             335        ; and then set external will assign this
                        ; value to ext(335)
        0   -1000       ; end of set external list
    (code)              ; set externals returns to this addr.
```

4.3.3 Reserve Virtual 4.3.3

The procedure is used to reserve (find room) in the virtual store for a number of hws, and returns the virtual address of the first word reserved.

The abs addr of entry to the procedure is in ext(12).

The reservation must be specified to either drumcore or disccore.

Call:	Return:
W0 = drum or disc	W0: = unchanged
(W0 extract 1 = 0 means drum)	
W1 = length of reservation, in hws	W1: = unchanged
W2 = irr.	W2: = virtual address
W3 = return addr	W3: = undefined.

- Successive calls of reserve virtual with same value of W0 extract 1 and W1 >= 512, will make reservations on consecutive segments.

- if W1 >= 512, the reserved room will start at a segment start, and an unused part, if any, of last segment will not be used in later reservations.
- if W1 < 512, the procedure will find the smallest suitable "hole" (if any), left over from previous calls with W1 < 512

4.3.4 Move to Virtual 4.3.4

The procedure is used to move a just initialized page to virtual core. Note that it is dummy in pass1.

The abs addr of entry to the procedure is in ext(13).

Call:	Return:
W0 = first addr to move	W0: = unchanged
W1 = length in hws	W1: = unchanged
W2 = virtual address to move to (W2 extract 1 = writing)	W2: = unchanged
W3 = return addr.	W3: = unchanged

4.3.5 Put in Active Queue 4.3.5

The procedure is used to chain a coroutine to the active queue, so it will be activated when the load is completed.

The abs addr of entry to the procedure is in ext(24). Note that the procedure is dummy in pass1.

Call:	Return:
W0 = irr.	W0: = undefined
W1 = coroutine description addr (in coroutine table)	W1: = unchanged
W2 = irr.	W2: = abs addr of active queue.
W3 = return addr.	W3: = unchanged

4.3.6 Simulate Lock 4.3.6

The procedure is used to chain a coroutine to a semaphore, so it will be waiting for this semaphore to be opened when the load is completed.

The abs addr of entry to the procedure is in ext(14). Note that the procedure is dummy in pass1.

Call:	Return:
W0 = irr	W0: = undefined
W1 = coroutine descr. addr (in coroutine table)	W1: = unchanged
W2 = semaphore descr. addr (in semaphore table)	W2: = unchanged
W3 = return addr.	W3: = undefined.

4.4 Format of a Module File

Output from slang compiler must have the following format:

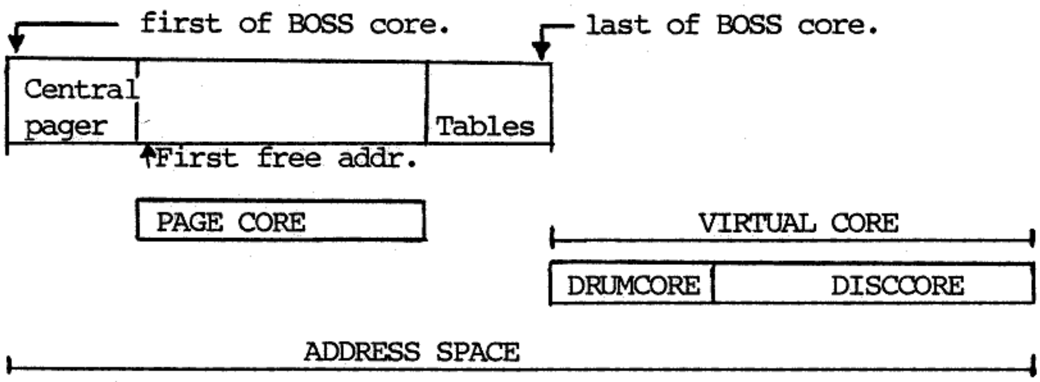
+0	from slang: sum of chars (S0)
+2	from slang: double sum of chars (S1)
+4	date of version (e.g.: 820220)
+6	ident of version (e.g: 75)
+8	first external REF — External list:
+10	second external REF
	. . .
	(module code)
	(maybe continued external list)
	last external ref
	0 (end external list)
ENTRY POINT ->	First instruction of initialization code
	(module code)

The module files are named:

bos	loaded by FP or s. Contains Central Logic, pager and loader.
bterm2 bterm1 bjobstart bjob bmount bread bprinter bprocs bbanker	Loaded by the loader, in the order mentioned.

5. PAGER, VIRTUAL CORE 5.

The virtual core is defined as follows:



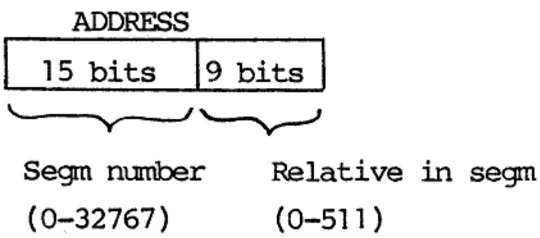
The addressing is based on core addresses. The process description of BOSS gives the addresses for first and last of BOSS core.

The virtual core is addressed as a continuous extension of the BOSS core.

The address space has the limits:

- first addr = 0 (= register W0).
- last addr = last addr of disccore.

The address space is split into segments, so an address may be decoded, thus:



Segmentsæt til 0-4095 pga coretable format

5.1 A Virtual Address 5.1

An address pointing in the virtual core, i.e. a "core" address > last of BOSS core. But the term may also be used for an address which may be both in BOSS core and virtual core, e.g. the page addresses in the coroutine description.

5.2 A Page

Initially a page is a number of consecutive hws, residing somewhere in the virtual core and moved to and from page core by the pager. The hws of a page will always remain together, regardless of where it is placed in page core.

A page is created during initialization by call of "reserve virtual" (and output to virtual core by "move to virtual").

The length of a page is not maintained after reservation, but is implicit known to the code.

The page is identified by the virtual address of first word. This is often stored in an external, to pass the address to other modules.

In this way is created
- codepages
- variable pages
- some of the internal operations (e.g. convert operations)
- work pages

After the creation of a page, it might be used and addressed freely.

For example, a variable page is created during initialization. Somewhere on this page, an operation may be set up. It is signalled to a semaphore by the open chained procedure (the abs address of the operation is set at call, and this is converted to a virtual address by C.L.).

When another coroutine receives this operation, the address is placed as page2 of the coroutine (so now only the operation is handled as a page). The receiving coroutine may have no knowledge or interest in the start address of the original page.

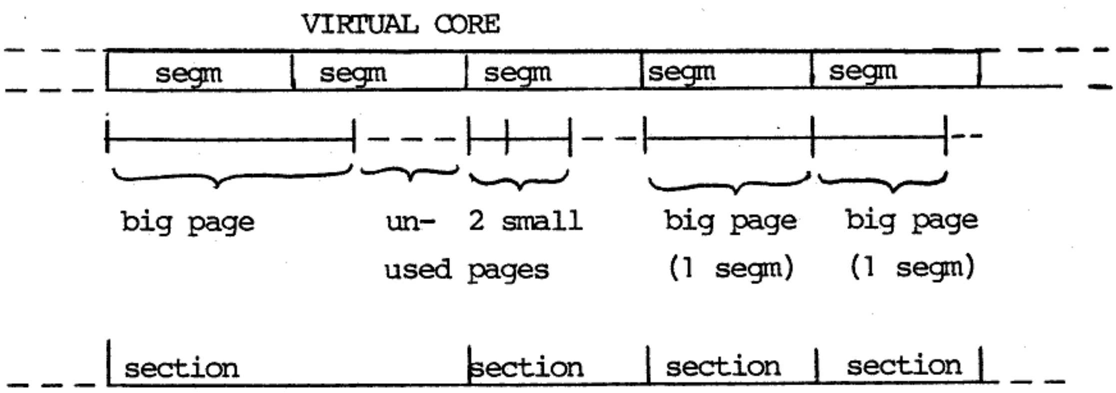
5.3 A Section 5.3

A section is a number of consecutive segments in virtual core. Sections are what the pager is actually paging (and sometimes the term page is also used for a section).

A section contains an integral number of segments in virtual core. The segments of a section remain consecutive when moved to page core.

A section contains one or several pages as defined to "reserve virtual"). A "big" page (size >= 512 hws), will occupy one section alone, but otherwise it will probably be placed on a section together with other "small" pages.

Example:



5.4 Use of BOSS Core 5.4

The core store of BOSS is divided into the following parts:

1) A part containing the code for the central logic and the pager coroutine. This part is of fixed size and resident in core.

2) A part containing buffers, coroutine descriptions, semaphores and tables of various kinds.
   This part is resident in core but the size of it varies with the installation.

3) A part containing the processes created by BOSS and the currently and most recently used pages from the virtual core of BOSS. The size of this part is determined by the size of the process in which BOSS is operating. The contents of this part of the core will change dynamically.

This gives the following layout of the core store of BOSS (cf. section 4.1):

1)	Central Logic and Pager Coroutine
3)	Job Processes and Pages from the Virtual Store
2)	Coroutine Descriptions Semaphores Buffers Tables etc.

The jobs running under BOSS are divided into two groups: a-jobs and b-jobs. The jobs may have different size and the core place is determined by the fact that a-jobs always start at the lower end and b-jobs always end at the higher end of the exchangeable part of BOSS (3).

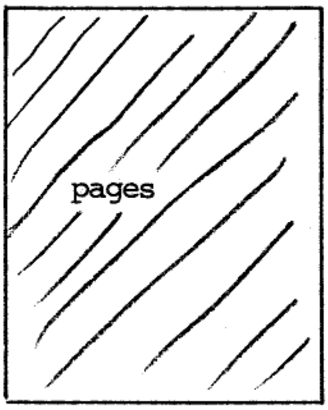
This exchangeable part may at different times have the following contents:

- no jobs in it
- one a-job only
- one b-job only
- one a-job and one b-job.

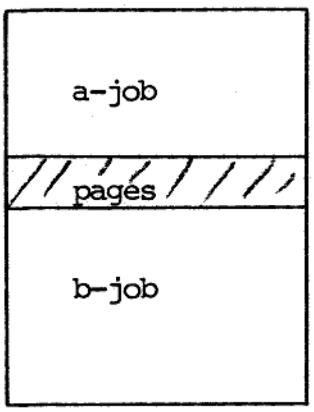
In any case the remaining core place will be used for pages from the virtual core of BOSS.

This leads to four different layouts:

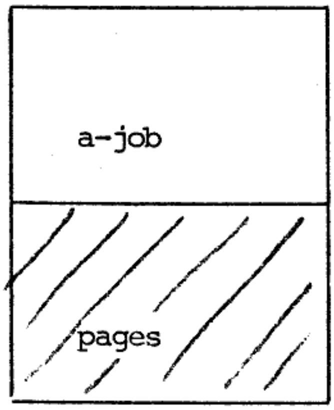
no jobs in core



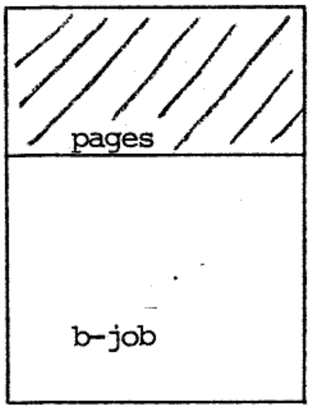
two jobs in core



one a-job only



one b-job only



5.5 Working Cycle of the Pager 5.5

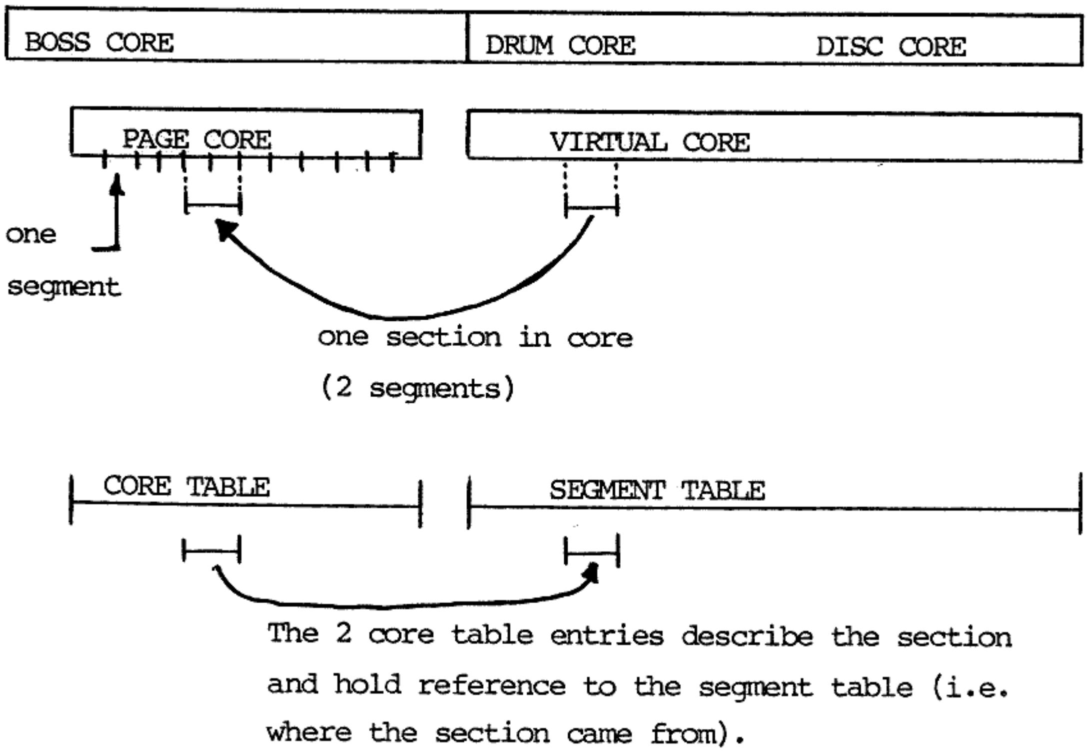
The pager receives operations in the pager queue. An operation to the pager consists of a coroutine description for a coroutine which wants to be put in the active queue but needs one or more pages from the virtual core.

Coroutines are put into the pager queue by the central logic which checks the presence of all specified pages of a coroutine each time it requests to operate on them (e.g. before exit to a coroutine, at lock, lock chained and open chained operations).

The pager builds up a page information list and two chains of pages, i.e. pages_in_core and pages_not_in_core.

5.6 Relations to Core Table and Segment Table

The relations can be illustrated by this example:



See descriptions of coretable and segment table in subsections 2.1.1 and 2.1.2.

6. TESTOUTPUT FORMATS 6.

The following pages give a quick survey of record types, format and contents.

In most modules, the term "kind" is used for what is called "record type" here.

SURVEY OF TESTOUTPUT RECORD TYPES

TYPE	TEST BIT	SOURCE: CONTENTS, DESCRIPTION
1 SEND	1	CL: MESS, at send and wait etc.
2 LOCK	1	CL: SEM, at lock (simple or chained).
3 OPCH	1	CL: SEM+OP, at open chained.
4 OPEN	1	CL: SEM, at open simple.
5 EXIT	1	CL: PAGE-ADDR+REL.EXIT, at exit to code page. After: all except open.
6 MESS	1	CL: MESSAGE, after: message ready for coroutine.
7 ANSW	1	CL: ANSWER, after: answer arrived to coroutine.
8 JD-1	2	CL: W0, W1, W2, W3, at jd-1 call.
9 STOP	(-)	CL: W0-W3, IC, BF etc, at BOSSFAULT + CLOSE.
10 OPER	1	CL: SEM+OP, after: lock chained (just before exit).
11	4	BANKER: PSJOB+FREE RESS, REC1: psjob a+b, REC2-REC5: free ress.
12		JOBSTART: jobfile page, jobdescr page PSJOB FINIS: timer values.
13 LOAD	(-)	START UP: REC1:install.name, REC2: startup params, REC3-REC20: from loader.
14 EXTN	(-)	START UP: External table, after second loaderpass.
15 LINE	2	COMMAND/JOBSTART: line, from terminal or jobfile.
16	2	BS ADJUST: BS CLAIMS.
17		JOB TERM/LOGOUT: CAT.ENTRY, when it is removed.
18		MOUNTER: STATION TABLE, at each activation of mounter.
19		MOUNTER: TAPE TABLE, at each activation of mounter.
20		PREP./TERM. ACCESS: ENTRY, ACCESS COUNT, output at call.
21		: AREA PROCESS.
22 DUMP		CL: CODE DUMP. Primary storage of code leading to interrupt. At BOSSFAULT+CLOSE.
23 REQU		REQ.DISPL: REQUEST OPERATION, output when received.
24		START UP: LOAD RESULT. End of loader, end of fixed part of testoutput.
25 PSJ1		BANKER: PSJOB descr. part 1.
26 PSJ2		BANKER: PSJOB descr. part 2.
27 PSJ3		BANKER: PSJOB descr. part 3.
28 PSJ4		BANKER: PSJOB descr. part 4.

SURVEY OF TESTOUTPUT RECORD TYPES (continued)

TYPE	TEST BIT	SOURCE: CONTENTS, DESCRIPTION
29		LOOKUP HOST: call params.
30		LOOKUP HOST: return params.
31		LOOKUP DEVICE, LINK-UP REMOTE: call params.
32		LOOKUP DEVICE, LINK-UP REMOTE: message to host.
33		LOOKUP DEVICE, LINK-UP REMOTE: data to host.
34		ALL HOST PROCEDURES: answer from host.
35		ALL HOST PROCEDURES: data from host.
36		LOOKUP DEVICE, LINK-UP REMOTE: return params.

Format of testoutput records

Types: 1-2-3-4-5-6-7-8

HEAD (WORD1, WORD2, w3) — TAIL (w4, w5, w6 →)

WORD1	WORD2	w3	w4	w5	w6	w7	w8	w9	w10	w11	w12	w13
1 SEND	TIME	NTA of receiver	MESS1	MESS2	MESS3	MESS4						
2 LOCK	TIME	SEM-ADDR	SEM-VALUE (before)									
3 OPCH	TIME	SEM-ADDR	SEM-VALUE (before)	OPER 1	OPER 2	OPER 3						
4 OPEN	TIME	SEM-ADDR	SEM-VALUE (before)									
5 EXIT	TIME	CORUT. IDENT	PAGE0	PAGE1	PAGE2	PAGE3	PAGE4	CODE PAGE IDENT <12+ REL. EXIT IN PAGE0				
6 MESS	TIME	CORUT. IDENT	÷ PDA of receiver	PDA of sender	MESS1	MESS2	MESS3	MESS4	MESS5	MESS6	MESS7	MESS8
7 ANSW	TIME	CORUT. IDENT	RESULT (of wait answer)	PDA of Sender (=BOSS)	ANSW1	ANSW2	ANSW3	ANSW4				
8 JD-1	TIME	CORUT. IDENT	w0	w1	w2	w3	IC (REL)					

Type 5 (EXIT), PAGE0–PAGE4: ABS adresses (or: virtual addresses, option e8=-1)

Type 6 (MESS), MESS1–MESS8: CONTENTS OF MESSAGE

Type 7 (ANSW), ANSW1–ANSW4: FIRST PART OF ANSWER

1 SEND — Call: - send and wait (-, fast, slow)
- stop and wait
- wait answ

} MESS undefined

2 LOCK — Call: - lock (simple)
- lock chained

3 OPCH — Call: - open chained

4 OPEN — Call: - open (simple)

5 EXIT — Return after:
- all wait op.
- lock, lock ch
- opench (not: open)
- get pages
- page jump
- clear core.

} ALL CL. procedures, except those with "no delay"

6 MESS — After: Message received (from job or terminal) and coroutine expecting it.

7 ANSW — After: Answer received to a message from coroutine.

8 JD-1 — "Call": - jd-1, testoutput of registers.

IC (REL): { IC - PAGE0ABS, if IC > PAGE0
else IC

FORMAT OF RECORD HEAD

WORD1	WORD2	WORD3	- - -
TAIL (HW) <6 + TYPE	TIME = Units of 0.01 secs after start	- - - Depend on TYPE	TAIL (HW)

WORD1=0 MEANS: REST OF SEGM UNUSED

Max value is 5 368 709 (appt. 14 hours, 55 min)

Format of testoutput records (continued)

Types: 9–10

HEAD: WORD1, WORD2, w3 — TAIL: w4 – …

WORD1	WORD2	w3	w4	w5	w6	w7	w8	w9	w10	w11	w12	w13	w14	w15
9 STOP	TIME	CORUT. IDENT	w0	w1	w2	w3	STATUS	IC	CAUSE	ADDR	344064	÷ BF (BOSS FAULT)	PAGE 0 (address)	CODEPAGE IDENT <12+ EXIT REL

After: - internal interrupt (close down)

CORUT. IDENT: or coroutine descr. addr, if this is nonsense.

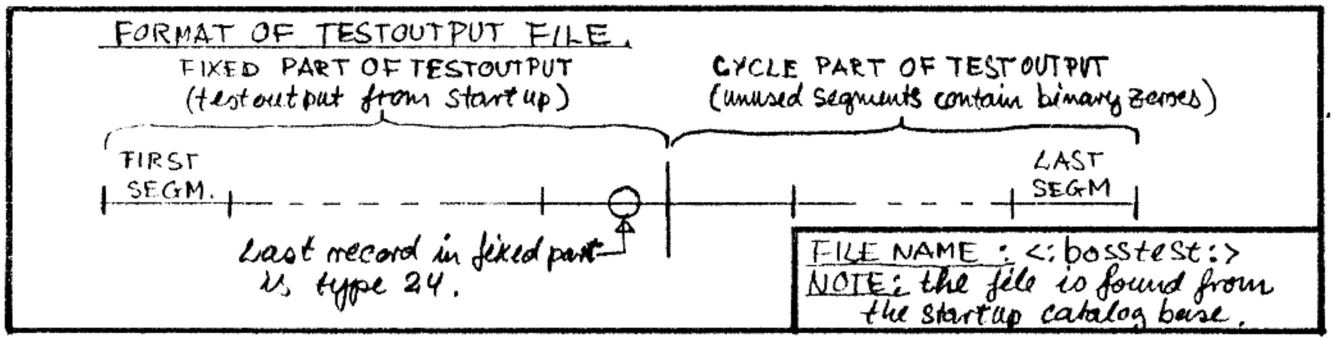
CODEPAGE IDENT: or page0 address, if this is nonsense

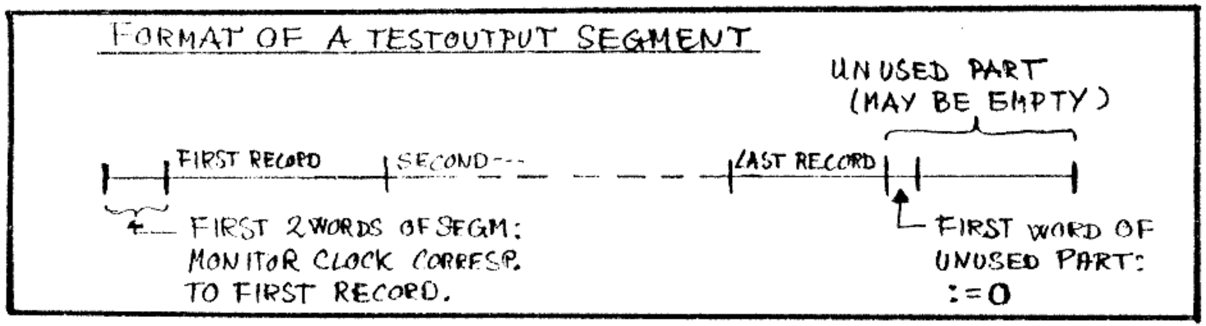
WORD1	WORD2	w3	w4	w5	w6	w7
10 OPER	TIME	CORUT. IDENT	NEW SEM. VALUE	OP1	OP2	OP3

OP1–OP3: Part of operation (or irr. if length < 3w)

After: - lock chained, when operation is delivered.

NEW SEM. VALUE: or: SEM-ADDR (if coroutine has been waiting)





Format of testoutput records (continued)

Types: 11-12

HEAD: word 1 – word 3; TAIL: w4 – w18

	word 1	word 2	word 3	w4	w5	w6	w7	w8	w9	w10	w11	w12	w13	w14	w15	w16	w17	w18
FIRST	11	TIME	CORUT. IDENT	PSJOB REL. A-JOB (or = 0)	PSJOB REL. C-JOB (or = 0)													
				FREE DYNAMIC RESOURCES:					FREE STATIC RESOURCES:									
NEXT 4 REC.	11	TIME	CORUT. IDENT	STATIONS	ENTRIES (MAINCAT) — TEMP DISC	SLICES — TEMP DISC	STD READERS	REMOTE READERS	CONVERT OPER.S	ACCOUNT OPER.S	MESS BUFS	AREA PROCS	INTERNALS	SUSPENDS	ENTRIES (DUMMY) — TEMP DRUM	SLICES — TEMP DRUM	FREE DEVICES 1	FREE DEVICES 2
	12	TIME	CORUT. IDENT	PART OF JOB DESCRIPTION PAGE (10 WORDS)														
	12	TIME	CORUT. IDENT	PART OF JOB FILE PAGE (10 WORDS)														
	12	TIME	CORUT. IDENT	PART OF JOB DESCRIPTION PAGE (6 words) (d115+0 to d115+10)														

Type 11, first record:
- PSJOB, FREE RESOURCES
Produced by the Banker
5 consecutive records

Type 11, next 4 records: ← 4 records (job class 0,1,2,3). All values one too high

Type 12 (job description page, 10 words):
- JOBSTART:
Job description page
on 23 consecutive records.

Type 12 (job file page, 10 words):
- JOBSTART:
Job filepage on
23 consecutive records.

Type 12 (job description page, 6 words):
- PSJOB finis
Part of job description page
containing inf on runtime left.

Format of testoutput records (continued)

Types: 13-14-15-16

HEAD: word1, word2, word3 — TAIL: w4, w5, w6, w7, w8, w9, w10, w11, w12, w13, w14, w15, w16, ...

INSTALLATION IDENT. First startup record. Fixed part.

13	TIME	0	INSTALLATION NAME (options page 1: e0 length e29 hw)

INSTALLATION PARAMS. Second startup rec. Fixed part.

13 LOAD	TIME	0	BOS (name of first coroutine file)	Monitor clock at startup time	LOAD ADDR OF COROUTINE FILES	SUM	DOUBLE SUM	VER-DATE	VER NO
						FROM SLANG		VERSION-ID	

w15, w16, w17, w18, w19, w20, w21, w22, w23, w24, w25, w26

FIRST ADDR	LAST ADDR	TOTAL NO OF STORAGE HW'S	MONITOR RELEASE <12+ VERSION	BOSS RELEASE <12+ VERSION	MONITOR RELEASE	NO OF TERMINALS	NO OF DRUM+ DISC DEVICES	NO OF EXCH. DISCS	NO OF STD. PRINTERS	NO OF REMOTE PRINTERS	NO OF PS.-JOBS
OF BOSS PROCESS		CORE (116)	CORE (64)	OPT. e78	OPT. e79	OPT. i4	OPT. i29	OPT. e23	OPT. i71	OPT. i190	OPT. i45

MODULE LOAD. Fixed part. From loader.

13 LOAD	TIME	0	NAME of coroutine file	FIRST OF COROUTINE TABLE	FIRST OF SEMAPHORE TABLE	FIRST OF SENDER TABLE	SUM	DOUBLE SUM	VER DATE	VER NO
							FROM SLANG		VERSION ID	

EXTERNAL TABLE. Output after second loader pass. Fixed part.

14 EXTN	TIME	N (=start external)	EXT N	EXT N+1	EXT N+2	ETC. (RECORD LENGTH=53 WORDS)	EXT N+49

N = -50, 0, 50, ... 550

After INPUT OF LINE output by command or jobstart.

15 LINE	TIME	CORUT. IDENT	CONTENTS OF INPUT LINE (TEXT)

BS CLAIMS from bs-adjust one record for each bs-device

16	TIME	CORUT IDENT	ENTRY <12+ SLICE (KEY0)	ENTRY <12+ SLICE (KEY1)	ENTRY <12+ SLICE (KEY2)	ENTRY <12+ SLICE (KEY3)	(BS DEV. NAME) (DOC. INDEX)

Format of testoutput records (continued)

Types: 17-18-19

HEAD (word 1 – word 3), TAIL (word 4 – …)

17 – CATALOG ENTRY

Output when a catalog entry is removed. (joboterminaiton or logout)

word 1	word 2	word 3	word 4	w5	w6	w7	w8	w9	w10	w11
17	TIME	CORUT. IDENT	FIRST SLICE <12+ NAMEKEY <3+ PERMKEY	LOWER BASE	UPPER BASE	ENTRY NAME				CONT
			ENTRY HEAD							

w11	w12	w13	w14	w15	w16	w17	w18	w19	w20
SIZE (>=0) / 1 <23+ MODE <12+ KIND	DOCUMENT NAME				TAIL 6	TAIL 7	TAIL 8	TAIL 9	TAIL 10
ENTRY TAIL									

18 – STATION TABLE ENTRY

Output each time the mounter is activated

word 1	word 2	word 3	word 4	w5	w6	w7	w8	w9	w10	w11	w12
18	TIME	CORUT. IDENT	DEVICE <12+ TAPE REF	LABEL MODE <18+ STATE <12+ TAPE KIND	PROJECT NUMBER	DESCR. ADDR. FOR USER	REL. SEM <12+ FREE AND STANDARD	OPE-RATION: CHAIN	REMOTER BUF		

19 – TAPE TABLE ENTRY

Output each time the mounter is activated

word 1	word 2	word 3	word 4	w5	w6	w7	w8	w9	w10	w11	w12
19	TIME	CORUT. IDENT	STATE <12+ STATION REF	JOB NR.	REQUEST LINE ADDR OR=0	ABS DEVICE <12+ PSJOB REF	TAPE NAME				NAME TABLE ADDR

Format of testoutput records (continued)

Types: 20-21-22

HEAD / TAIL

Type	WORD1	WORD2	WORD3	w4	w5	w6	w7	w8	w9	w10	w11	w12	w13	w14	w15	w16	w17
20	TIME	CORUT. IDENT	NTA OF AREA PROCESS	1 = PREPARE −1 = TERMINATE	ACCESS COUNT	LOWER BASE, ENTRY	UPPER BASE, ENTRY	NAME OF ENTRY									
21	TIME	CORUT. IDENT	LOWER BASE, AREA	UPPER BASE, AREA	KIND	NAME OF AREA PROCESS					CONT						

- Prepare access or
- Terminate access

Output at return.

- AREA PROCESS description

w11	w12	w13	w14	w15	w16	w17	w18	w19	w20	w21	w22
PDA OF PERIPH. PROCESS	RESERVER (BIT)	USERS (BITS)	FIRST SLICE	NO OF SEGM.	(DOCUMENT NAME				NO OF TIMES WRITTEN	NO OF TIMES READ	SHORT-CLOCK, LAST ACCESS)

Type	WORD1	WORD2	w3	w4	w5	w6	w7	w8	w9	w10	w11	w12
22 DUMP	TIME	CORUT. IDENT	(IC-20)	(IC-18)	(IC-16)	(IC-14)	(IC-12)	(IC-10)	(IC-8)	(IC-6)	(IC-4)	(IC-2)
			CONTENTS OF PRIMARY STORAGE.					INSTRUCTION CAUSING INTERRUPT ↑				

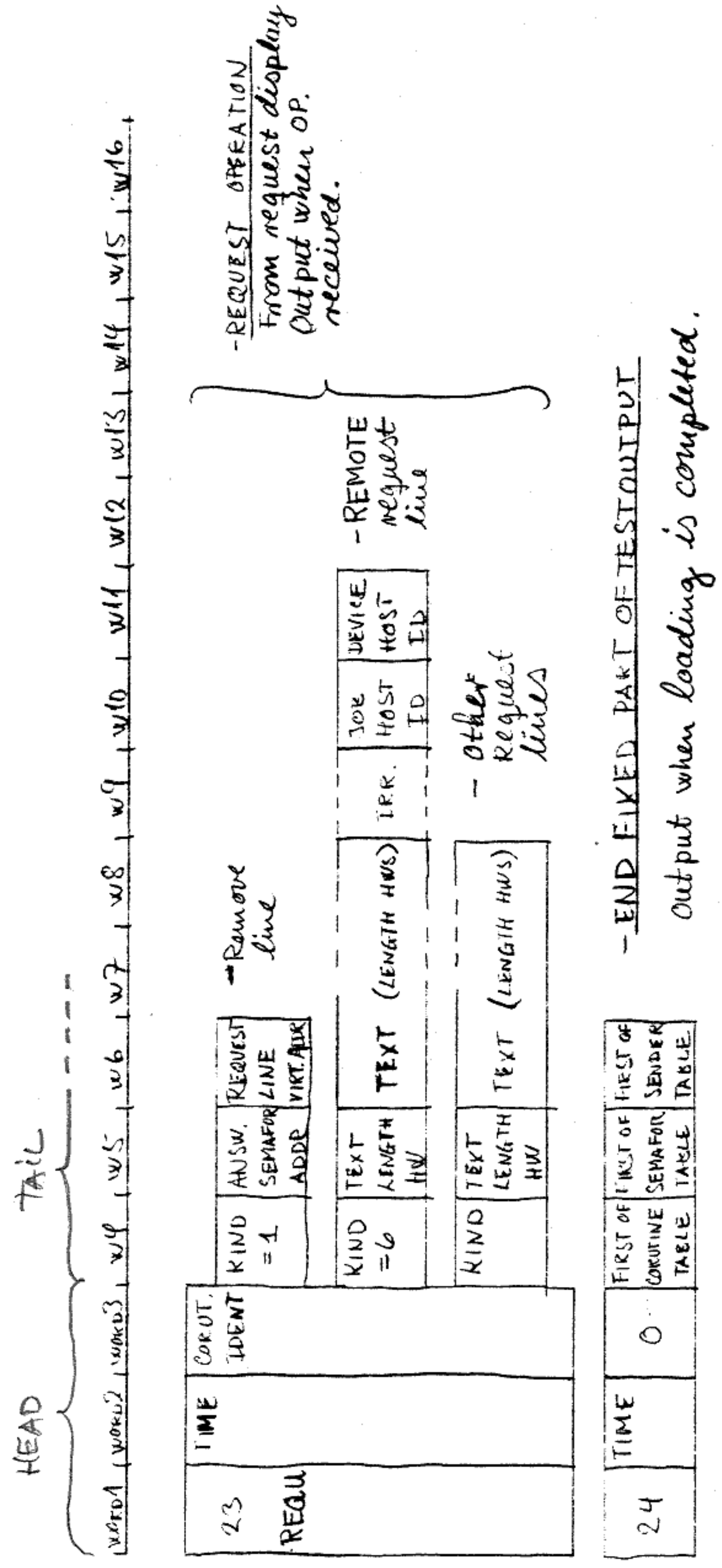
- CORE DUMP output just before the stop record.

Dump length = i205 hw

min 20, max 502 hw

Format of testoutput records (continued)

Types: 23-24



Format of testoutput records (continued)

Types: 25-26-27-28

HEAD: word1–word3; TAIL: w4 onwards

word1	word2	word3	w4	w5	w6	w7	w8	w9	w10	w11	w12	w13	w14	w15	w16	
25 PSJ1	TIME	CORUT. IDENT	JOB NAME				VIRT. ADDR. (ANSW. OR DUMP NAME)	STATE <12+ TIME CLASS <1+ CORELOCK	REL TIME TO TIMEOUT	REL TIME TO WAIT SWOPPED OUT	ABS EXPECT. FINISH TIME					FIRST
									UNITS OF 0.8192 S.							
26 PSJ2	TIME	CORUT. IDENT	COREPLACE FIRST SEGM <12+ TOP SEGM	255 ÷ PROT. REG <12+ PRIO FACT	REL GROSS RUN LEFT (+2)	REL NET RUN LEFT (+2)	ABS ARRIVAL TIME (+2)	REL MAX WAITING TIME								SECOND
					UNITS OF 0.8192 SECS											
27 PSJ3	TIME	CORUT. IDENT	PRIO OR IDLE CHAIN <12+ RESO. OR DEADLY CH.	REST CLAIMS: ALL *) <12+ STATIONS	ENTRIES <12+ SLICES, TEMP DISC	STD. READ's <12+ REMOTE READERS	STATIC RESOURCES: CONVERTS <12+ ACCOUNTS	MESS. BUFS <12 AREA'S	INTERNALS <12+ SUSPENDS	ENTRIES <12+ SLICES, TEMP DRUM	DEVICE WORD 1	DEVICE WORD 2				THIRD
											BIT WORDS					
28 PSJ4	TIME	CORUT. IDENT	RESERVED RESOURCES: ALL *) <12+ STATIONS	ENTRIES <12+ SLICES, TEMP DISC	STD. READ's <12 REMOTE READERS	WANTED RESOURCES: ALL *) <12+ STATIONS	ENTRIES <12+ SLICES, TEMP DISC	STD. READ's <12+ REMOTE READERS								FOURTH

— PSJOB DESCRIPTION

From Banker. 4 consecutive records.

*) ALL: Shows whether the STATIC RESOURCES should be included in the resource set or not (bits).
:= (converts included) < 2 +
(accounts included) < 1 +
(all other static incl.);

Format of testoutput records (continued)

Types: 29-30-31-32

HEAD: WORD 1, WORD 2, WORD 3; TAIL: w4, w5, w6, w7, w8, w9, w10, w11, w12, w13, w14

WORD 1	WORD 2	WORD 3	w4	w5	w6	w7	w8	w9	w10	w11	w12	w13	w14
29	TIME	CORUT. IDENT	ABS. REF, DATA	PROC. DESCR. ADDR									
30	TIME	CORUT. IDENT	LINK TYPE	PROC. DESCR. ADDR	SUB-HOST NO.	DEV. HOST IDENT							
31	TIME	CORUT. IDENT	ABS. REF, DATA	1 (DEVICE DESCRIPTOR)	2 (DEVICE DESCRIPTOR)	3 (DEVICE DESCRIPTOR)	4 (DEVICE DESCRIPTOR)	SUB-HOST NO.	DEV. HOST IDENT	MODE <12 + KIND	TIME-OUT (MIN)	LOWER BASE	UPPER BASE
32	TIME	CORUT. IDENT	OPER <12 + FUNCT. <1 + ADR.MODE (MESSAGE (8 WORDS))	FIRST ADDR OF DATA (MESSAGE)	LAST ADDR OF DATA (MESSAGE)	SUB-HOST NO (MESSAGE)	DEV. HOST IDENT (MESSAGE)	1 IRRELEVANT (MESSAGE)	2 IRRELEVANT (MESSAGE)	3 IRRELEVANT (MESSAGE)			

29: *LOOKUP HOST:* CALL PARAMETERS. From PROCS

30: *LOOKUP HOST:* RETURN PARAMETERS. From PROCS

LINK TYPE:
- 0 = unknown, error (subhostno = dev.host ident = 0)
- 1 = temp (remote) link
- 2 = perm (local) link (subhostno = dev.host.ident = 0)

31: *LOOKUP DEVICE, LINKUP REMOTE:* CALL PARAMETERS. From PROCS.

DEVICE DESCRIPTOR(1) = 1 means "remote by default" printer.
- otherwise: remote cat. entry name or name in device host

TIME-OUT (MIN):
- 0: no wait
- 128: wait forever

32: *LOOKUP DEVICE, LINKUP REMOTE:* MESSAGE TO HOST. From PROCS

OPER <12 + FUNCT. <1 + ADR.MODE:
- 1<12 + 6: Lookup device
- 1<12 + 12: Linkup remote

Format of testoutput records (continued)

Types: 33-34

HEAD: WORD1–WORD3; TAIL: w4–w14

WORD1	WORD2	WORD3	w4	w5	w6	w7	w8	w9	w10	w11	w12	w13	w14
33	TIME	CORUT. IDENT	MODE <12+ KIND	TIMEOUT <12+ BUFFERS	LINK BUFSIZE HWS	1 DEVICE NAME IN DEVICE HOST	2	3	4	1 IRRELEVANT	2	3	
34	TIME	CORUT. IDENT	STATUS	HWS TRANSF (DATA)	CHARS TRANSF (DATA)	DEV.HOST LINKNO <12+ SUB-HOSTNO	DEV. HOST IDENT	1 IRRELEVANT	2	3			
			ANSWER (8 WORDS)										

Type 33: LOOKUP DEVICE, LINKUP REMOTE: DATA TO HOST. From PROCS.

Type 34: ALL HOST MESSAGES: ANSWER FROM HOST. From PROCS.

STATUS:

DEVICE STATUS (8bits) <16 + LINK DESCR. (4 bit number) <12 + FUNCTION RESULT (12 bit signed number)

DEVICE STATUS:

BIT 0: (MSB) device unknown
1: device closed
2:
3:
4:
5: device driver not loaded
6: device reserved by other process
7: reservation rejected from A.P. (only GAC-interfaces)

LINK DESCR.:

0: no link
1: temp (remote) link
2: perm (local) link

FUNCTION RESULT:

-1: sender stopped
0: OK, function executed
1: trouble, see device status
2: device reserved by other jobhost
3: no ressources at jobhost
4: no ressources at devicehost
5: time out
6: device requested with higher priority
7: some link was present, see LINK DESCR.
8: device host unknown.

Format of testoutput records (continued)

Types: 35–36

HEAD: WORD1, WORD2, WORD3; TAIL: W4, W5, W6, W7, W8, W9, W10, W11, W12, W13, W14

Type	TIME	CORUT. IDENT	KIND	MAX BUF-FERS	LINK BUFSIZE CHARS	1	2	3	4	JOBHOST LINKNO =DEVNO	JOBHOST HOST IDENT.	IRR.	P.D.A. LINK SUBPROC	
35						DEVICE NAME IN DEVICE HOST								ALL HOST MESSAGES (if answ.result = 1): DATA FROM HOST. From PROCS.

Type	TIME	CORUT. IDENT	N.T.A.	1	2	3	4	SUB-HOST NO	DEV. HOST IDENT	MODE <12 + KIND	DEVNO	LINK BUFSIZE HWS	LINK BUFSIZE CHARS	
36				DEVICE NAME IN DEVICE HOST										LOOKUP DEVICE, LINKUP REMOTE: RETURN PARAMETERS From PROCS.

NTA and DEVNO:

- After LOOKUP DEVICE: both are = 0
- If result = OK: NTA = Name Table Address of sub-process
  DEVNO = device number of sub-process
- If result = NOTOK: NTA = Modified Function Result (MFR)
  DEVNO = STATUS < -12 (cfr. type 34) if MFR < 40
  = 0 if MFR > 40

MODIFIED FUNCTION RESULT (MFR):

1: trouble, see device status
2: device reserved by other jobhost
3: no ressources at jobhost
4: no ressources at devicehost
5: timeout
6: device requested with higher priority
* 7: device has permanent link (cfr. type 34)
8: devicehost unknown

Device has already a temp (remote) link, which is:

* 20: reserved by BOSS
* 21: reserved by some other process

BOSS received a dummy answer to the host message (MFR = 40 + answer result):

* 42: rejected
* 43: unintelligible
* 44: malfunction
* 45: does not exist

Reservation error as the subprocess could not be reserved (MFR = 50 + reserve result)

(* 51: reserved by other – see * 21.)
* 52: BOSS not user
* 53: process does not exist

* MEANS: this value comes from BOSS.

6.1 Codepage Identifications

Codepage identification is found in the EXIT-record:

central
9: pager

tjobstart
11: codepage 1 (job birth, online convert)
12: codepage 2 (load commands, set claim)
13: codepage 3 (command reading)
14: codepage 4 (create job, request alarm)
15: codepage 5 (load, rb-comm)

tterm 1
20: command input
21: commandio bulk file
22: command print
23: snapshot, autoline

tprinter
70: paper description
71: central page
72: triangle page
73: main loop

tjob
40: psjob I/O
41: psjob aux
42: psjob break
43: clean catalog
44: psjob finis

tprocs
80: account
81: bsadjust
82: init from usercat
83: unknown sender
84: various procedures for remote devices

tmount
50: psmount
52: remoter
54: rewinder
56: watchdog
58: comandio, name, label

tbanker
90: main banker
91: core allocation
92: resource allocation

tread
60: tape reader
61: card reader
62: start card
63: pstape
64: pscard
65: psload

tterm 2
100: kill
101: go, run, job, newjob
102: rename, clear, scope
103: login, get, save
104: transmit
105: display
106: kit changer
107: term out
108: request display

6.2 Coroutine Identifications

The coroutine identification is found in 3rd word in most test-output records.

ident	name	(init-code in ...)
0	output during start-up	(central)
1	timer	(banker)
2	banker	(banker)
3	unknown sender	(procs)
5	weteravaid	(Banker)
7	watchdog	(mount)
8	request display	(term2)
9	pager	(central)
10	operator display	(term2)
20*	kit changers	(term2)
40*	printers	(printer)
60*	rewinders	(mount)
80*	remoters	(mount)
100*	commandios	(term1)
(198	downer)	(banker)
199	convert	(banker)
200	account job	(jobstart)
200*	psjobs	(jobstart)
300*	termouts	(term2)
400*	card readers	(reader)
450*	readers	(reader)

Relation between coroutine numbers of psjobs and commandios:

```
                  PSJOBS:              COMMANDIOS:
               ┌ 200 account
               │ 201 ┐
20 psjobs      │  :  ├for newjobs etc.
= option      ─┤ 209 ┘
i45            │ 210 <- - --maybe- - -> 100 main console
               │ 211 <-------------> 101 ┐ 9 terminals
               │  :                   :  ├ = option
               └ 219 <-------------> 109 ┘ i4
                         │
                         └ MATCH in run/go-jobs i.e.:'run'
                           from term-coroutine 108
                           uses psjob-coroutine 218
```

RETURN LETTER

Title: BOSS, Basic Internal Formats Maintenance Manual RCSL No.: 31-D673

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

REGNECENTRALEN
af 1979

Information Department
Lautrupbjerg 1
DK-2750 Ballerup
Denmark