Title:

RC 3600 PAGINGS SYSTEM
SYSTEM PROGRAMMERS GUIDE

A/S REGNECENTRALEN

RC SYSTEM LIBRARY: FALKONERALLE 1 DK-2000 COPENHAGEN F

RCSL No: 43-RI0142 (PG1)
Edition: October 1975
Author: Philippe Gaugin

Keywords:

Mus, Paging System, Virtual Memory, Address Mapping

Abstract:

This manual describes how to use the RC 3600 paging system from assembly programs under the MUS-system. 19 pages.

CONTENTS PAGE

1. INTRODUCTION

The RC3600 paging system makes it possible to write large programs and to run them in a small amount of core storage at the cost of execution time. The programs are, by the programmer, broken into minor pieces, called pages, which are placed on a disk. When running the program the system takes care of bringing the pages into core.

As the hardware on the RC3600 computer does not support virtual memory systems, such programs are bound to be coded according to some rules, which makes it possible to detect and check by software every reference to virtual objects that may cause pages to be read into core and perhaps pages to be written back to disk from core.

Under the RC3600 paging system the programs are allowed to reference local objects by means of relative addressing, and core resident objects by means of deferred absolute addressing. The programs may reference objects on other pages by calling some procedures to obtain a first reference and by indexing to get or modify the objects on that page.

The system makes it possible to collect some statistics about the perfermance of the system.

2. PAGING SYSTEM ADDRESSING TECHNIQUE

The paging system extends the address space of a program with almost 32K of virtual memory. A program is divided into a core resident part and a number of pages of equal size. The page size should be 256, 512, 1024 or 2048 words, corresponding to the storage capacity of 1, 2, 4 or 8 disk sectors.

During load of paged programs, normal relocatable code is assigned to absolute addresses, while absolute code is assigned to virtual memory addresses except for absolute code in page zero locations. Thus the following type of assembly code is legal:

Type of assembly code	Range	Is loaded into
absolute	$(0, 377_8)$	page zero of core
absolute	$(PS, 77777_8)$	virtual memory
normal relocatable	$(0, 77777_8)$	core
byte relocatable	$(0, 177776_8)$	core
PS = page size	Note: absolute addresses $(400_8, PS-1)$ are illegal	

During run of paged programs the paging system maintains a partial map of virtual memory addresses into computer word addresses. Any access (read/write/execute) to a virtual memory location cannot be made before the virtual memory page has been brought into core, defining the map of that specific page. This is done by means of some procedures, which operates on addresses and program points.

A program point is a 16 bit quantity representing some place in a program. If bit 0 of a program point is zero, the point represents the address of a computer word. If bit 0 of a program point is set, the remaining part of the point (bits 1-15) represents a virtual address in the virtual address space of that program.

Points in interval	Represents
$(0, 377_8)$	page zero computer word addresses
$(400_8, 77777_8)$	other computer word addresses
$(100000_8 + PS, 177777_8)$	virtual memory word addresses
PS= page size	Note: points $(100000_8, 77777_8 + PS)$ represents nothing

3. PAGING SYSTEM PROCEDURES

3.1. Procedure Call (point)

	call	continuation	link
ac0		unchanged	+ 0: point
ac1		unchanged	+ 1: possible return
ac2		unchanged	
ac3	link	link + 1	

Executes a subroutine jump to the point given as parameter in the word following the call. Continues execution with ac3 pointing to a possible return address.

3.2. Procedure Goto (point)

	call	continuation	link
ac0		unchanged	+ 0: point
ac1		unchanged	
ac2		unchanged	
ac3	link	destroyed	

Executes a jump to the point given as parameter in the word following the call.

3.3. Procedure Getadr (point, address)

	call	return	link
ac0	point	unchanged	+ 0: return
ac1		unchanged	
ac2		unchanged	
ac3	link	address	

Computes the address of the point given as parameter in ac0. If the point is less than 100000_8, the address returned is equal to the point.

The above 3 procedures may change the page map. However, in a non-coroutine environment, the calling page would not be involved in the change, i.e. the calling page is untouched, when using the procedures Call and Goto.

3.4 Procedure Getpoint (address, point)

	call	return	link
ac0	address	unchanged	+ 0: return
ac1		unchanged	
ac2		unchanged	
ac3	link	point	

Computes the point corresponding to the address given as parameter in ac0. If the address points to a word inside a frame (a set of locations used to swop a page) the point corresponding to the virtual address of that word, is returned. Otherwise the address is returned.

This procedure does not change the page map.

3.5 Example (subroutine, linkage)

Subroutine:

```
SUBR:   STA 3   RETUR, 2
; the routine does not
; change the page map
        .
        .
        .
        JMP @   RETUR, 2
```

Subroutine:

```
SUBR:   MOV 3,0
        GETPOINT
        STA 3   RETUR, 2
; the routine does
; change the page map
        .
        .
        .
        LDA 0   RETUR, 2
        GETADR
        JMP 0,3
```

Calls:

```
1)   SUBR not resident
     CALL
     @ SUBR

2)   SUBR resident
     CALL
     SUBR

3)   SUBR resident
     JSR @        XX
     .
     .
     .
XX:  SUBR
```

4. PAGING SYSTEM IN A COROUTINE ENVIRONMENT.

If the paging system is used together with the coroutine monitor, the procedures call, goto, getadr may cause other coroutines to become active, if the referenced page is not in core. The condition that the calling page would not be involved in the change of the page map, fails when using coroutines, but the following weaker condition holds for coroutines: the calling page will be present in core when the referenced page has been brought into core, although its position in core may have changed. If so, the register ac3 is changed according to the new position of the page. Returns from subroutines can be made exactly as shown in the former examples.

In order to ease the handling of coroutine calls from pages, the following procedure is supplied:

4.1 Procedure Comon (coroutine monitor call)

	call	return	link
ac0	*)	*)	+ 0: coroutine monitor call
ac1	*)	*)	+ 1: return
ac2	*)	*)	
ac3	link	corout	*) = as for the coroutine monitor call

Executes the coroutine monitor call and arranges a proper return. At return the page map may have changed.

4.2 Example (call of coroutine monitor)

```
COMON
WAITSEM
.
.
.
SIGNAL
```

Note that SIGNAL is called normally, since a call of signal will not cause any immediate activation of other coroutines.

5. PAGING SYSTEM SETUP.

5.1 Programs

The paging system requires some variables to be set up in the beginning of the program.

pspec	:	Add 1b6 to the program descriptor word.
page size	:	number of words per page, i.e. 256, 512, 1024 or 2048.
page mask	:	minus number of words per page.
blocking factor	:	number of sectors occupied by 1 page, i.e. 1, 2, 4 or 8.
adr pagetable	:	the address of a table describing where to find the pages on the disk. The table should contain: pagetable : number of pages in the program: m; pagetable + 1 ... pagetable + m : irrelevant, these locations are set up by the loader.
adr pagemap	:	the address of a table describing the map of virtual addresses into core addresses. The table should contain: pagemap : 3 (semaphore used by paging routines) pagemap + 1 ... pagemap + m : irrelevant, these locations are set up by the loader.
adr statproc	:	the address of a procedure used to collect statistics. This procedure is called at every pagefault. The procedure is described later. If no procedure is present, this variable should be set to zero.
first of frames	:	the address of the first word in the core storage area used to load the pages from disk.
top of frames	:	the address of the first word after the core storage area used to load the pages from disk. This area should contain at least two frames, i.e. room for two pages.
victim	:	the address of the next frame to be used for transfer of pages. This should be set equal to first of frames.
pages read	:	counts the number of pages read into core. This should be set to zero.

pages written	:	counts the number of pages written back to disk. This should be set to zero.
page in	:	contains at the call of the statproc the pagenumber of the page which is going to be read into core. Initial contents irrelevant.
page out	:	contains at the call of the statproc the pagenumber of the page which is going to be written back to disk. Initial contents irrelevant.
adr input message	:	the address of the following word.
input message	:	contains the message used by the paging system to load pages into core. It should be initialized to: mess 0 : 9 (operation). mess 1 : number of bytes per page. mess 2 : byte address of victim. mess 3 : irrelevant.
adr output message	:	the address of the following word.
output message	:	contains the message used by the paging system to write pages back to disc. It should be initialized to: mess 0 : 11 (operation). mess 1 : number of bytes per page. mess 2 : byte address of victim. mess 3 : irrelevant.
pager flag	:	0.
working locations	:	.BLK PWSIZE.

5.2 Processes

The paging system also requires a variable to be set up in the process:

ccorout	:	should be set to zero if the process does not use the coroutine monitor. Otherwise it should be set to point to the first coroutine.

Also add one extra message buffer to the process, to be used by the paging system.

5.3 Coroutines

If the process uses the coroutine monitor, some working locations should be set up in every coroutine, just after the variable caclsave:

working locations	:	.BLK PCWSIZE. These locations may be used by the coroutine but are destroyed at every pagefault and at every call of COMON.

5.4 Pages

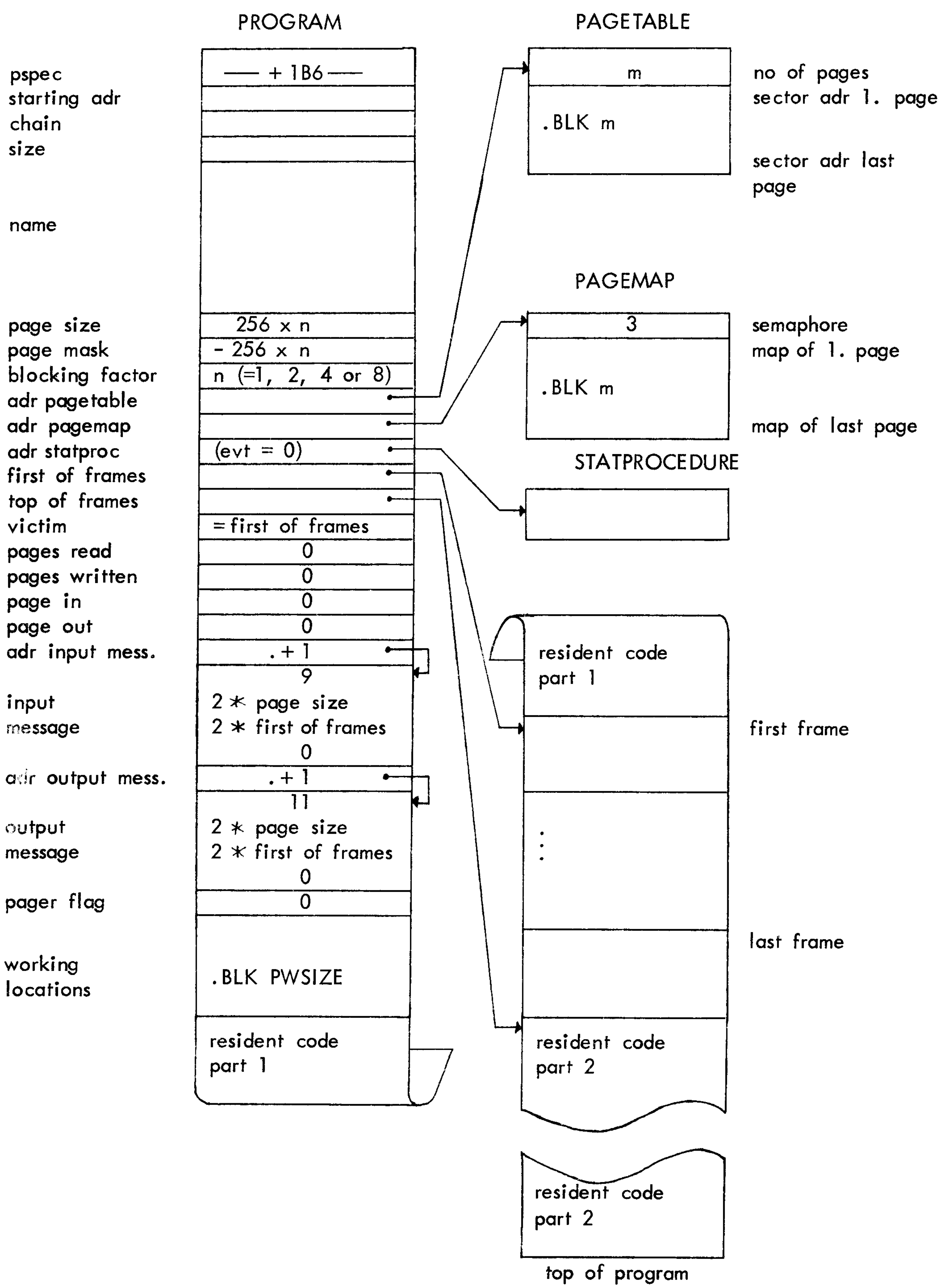
The first word on every page should contain the following:

virtual address of this page + page descriptor.

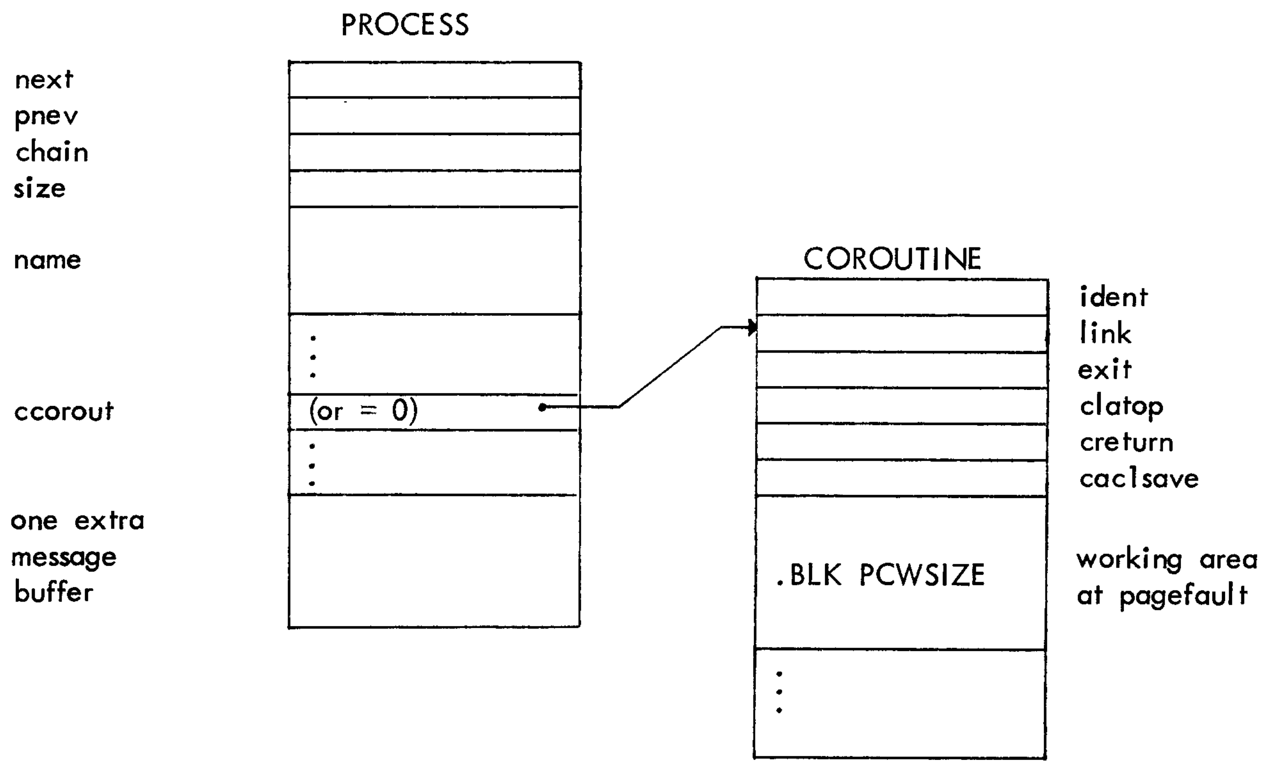
pagedescriptor	:	bit 15	:	0 read only page. 1 read/write page.
		bit 14	:	0 non locked in core. 1 locked in core for the moment.

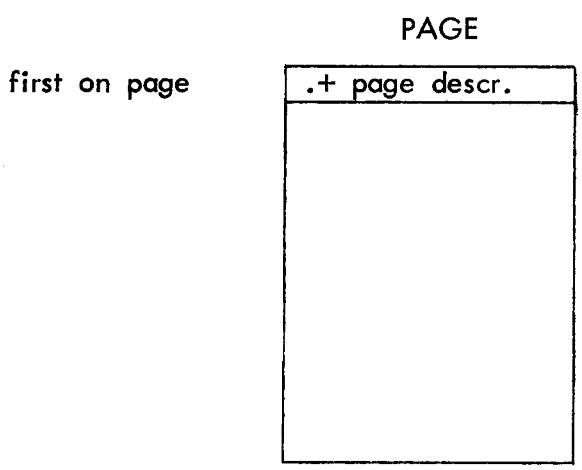
It is the users responsibility to use the lock bit properly

6. PAGING SYSTEM SETUP SUMMARY



PAGING SYSTEM SETUP SUMMARY (continued)





7. HOW TO USE THE STATPROC.

The statproc should fulfil the conventions:

Procedure Statproc

	call	return	link
ac0		destroyed	+ 0: return
ac1		destroyed	+ 1: special return
ac2	program	unchanged	
ac3	link	destroyed	

At the entry the following variables in the program are set to relevant values:

victim	:	frame to be used for transfer.
pages read	:	number of pages read before this pagefault.
pages written	:	number of pages written before this pagefault.
page in	:	page to be read.
page out	:	page to be written.

It is possible but not recommendable to change victim in the statproc. If this is done, the return should be made to link + 1 where victim will be checked and the statproc will be reentered with the new values of victim and page out.

8. ERRORS.

If an error occurs, the process will be breaked with errornumber = 7, and ac1 containing an errorcause:

ac1 = 0 : addressing error.
ac1 = 1 : too many frames locked.
ac1 ≠ 0 and 1 : disk error, ac1 = status.

9. EXECUTION TIMES.

Execution times for procedures when no pagefaults:

COMON	(- coroutine monitor call)	140 μ
	executed from resident part	68 μ
CALL	point (0 : 0) set	74 μ
	otherwise	24 μ
GOTO	point (0 : 0) set	70 μ
	otherwise	20 μ
GETADR	point (0 : 0) set	69 μ
	otherwise	19 μ
GETPOINT	result point (0 : 0) set	75 μ
	otherwise	43 μ

If a pagefault occurs, add the time for a pagefault to the above execution times.

Pagefault	administration	300 μ
	input transfer	1000 μ
	output transfer	1000 μ
	coroutine adm.	200 μ

PAGEFAULT 1300 - 2500 μ

Transfer time of 1 page to/from disk	4 - 200 ms
average	70 ms

10. EXAMPLE.

```
                        ; PROGRAMMING EXAMPLE: PAGED PROGRAM
                        .TITLE EXAMP
                        .NREL
              000012    .RDX 10
              000001    .TXTM 1

00000'101000 PG0:  1B0+1B6              ; PSPEC
00001'002077'      PG1                  ; STARTING ADDRESS
00002'000000       0                    ; CHAIN
00003'002104       PG10-PG0             ; SIZE
             .TXT  .EXAMP.              ; NAME
00004'042530
00005'040515
00006'050000
       001000 PVSIZE=512                ; SIZE OF PAGES IN VIRTUAL MEMORY
       000003 PVNO=3                    ; NO OF PAGES IN VIRTUAL MEMORY

00007'001000       PVSIZE               ; PAGE SIZE
00010'177000       -PVSIZE              ; PAGE MASK
00011'000002       PVSIZE/256           ; BLOCKING FACTOR
00012'000053'      PGTAB                ; PAGE TABLE
00013'000057'      PGMAP                ; PAGE MAP
00014'000063'      PGSTAT               ; PAGE STATISTICS PROCEDURE
00015'000077'      PGFOF                ; FIRST OF FRAMES
00016'002077'      PGTOF                ; TOP OF FRAMES
00017'000077'      PGFOF                ; VICTIM
00020'000000       0                    ; PAGES READ
00021'000000       0                    ; PAGES WRITTEN
       000022 PAGEIN=.-PG0
00022'000000       0                    ; PAGE IN
00023'000000       0                    ; PAGE OUT
00024'000025'      .+1                  ; ADDRESS INPUT MESSAGE
00025'000011       9                    ; INPUT MESSAGE: OPERATION
00026'002000       PVSIZE*2             ;                PAGE LENGTH
00027'000176"      PGFOF*2              ;                FIRST ADR
00030'000000       0                    ;
00031'000032'      .+1                  ; ADDRESS OUTPUT MESSAGE
00032'000013       11                   ; OUTPUT MESSAGE: OPERATION
00033'002000       PVSIZE*2             ;                 PAGE LENGTH
00034'000176"      PGFOF*2              ;                 FIRST ADR
00035'000000       0                    ;
00036'000000       0                    ; PAGER FLAG
       000014 .BLK  PWSIZE              ; WORKING LOCATIONS

00053'000003 PGTAB:PVNO                 ; PAGE TABLE
       000003 .BLK  PVNO                ;

00057'000003 PGMAP:3                    ; PAGE MAP
       000003 .BLK  PVNO                ;
```

```
                         PGSTAT:                           ; STATISTIC PROCEDURE
   00063'054413                 STA     3    PGST1         ; COMPUTE FREQUENCY COUNT PER PAGE
   00064'035022                 LDA     3    PAGEIN,2      ; AC3:= PAGE TO LOAD
   00065'024405                 LDA     1    PGSTO         ; AC1:= ADR FREQUENCY COUNT TABLE
   00066'137000                 ADD     1,3                ; FREQUENCY COUNT(PAGEIN):=
   00067'011400                 ISZ          0,3           ; FREQUENCY COUNT(PAGEIN) + 1;
   00070'002406                 JMP@          PGST1        ; RETURN
   00071'000777                 JMP           .-1
   00072'000071'PGSTO:.+0                                  ; FREQUENCY COUNT TABLE
   00073'000000                 0                          ; PAGE 1
   00074'000000                 0                          ; PAGE 2
   00075'000000                 0                          ; PAGE 3
   00076'000000 PGST1:0

         001000 PGFOF: .LOC          PVSIZE            ; FIRST OF FRAMES:
                ; *************** PAGE 1 **************
   01000 001000 PV1:    .+0                            ; READ ONLY PAGE
   01001 020406 PV10:  LDA      0    PV11              ; AC0:= POINT CASE TABLE
   01002 006367         GETADR                         ; GET ADDRESS OF CASE TABLE
   01003 137000         ADD      1,3                   ;
   01004 021400         LDA      0    0,3              ; AC0:= POINT(I)
   01005 006367         GETADR                         ; GET ADDRESS OF POINT(I)
   01006 001400         JMP      0,3                   ; GOTO POINT(I)
   01007 103003 PV11:  @PV31                           ; POINT CASE TABLE
         002000 .LOC    ./PVSIZE+1*PVSIZE              ; FILL UP SPACE

                ; *************** PAGE 2 **************
   02000 002000 PV2:    .+0                            ; READ ONLY PAGE
   02001 006366 PV20:  CALL
   02002 103001         @PV30
   02003 006365         GOTO
   02004 101001         @PV10
         003000 .LOC    ./PVSIZE+1*PVSIZE              ; FILL UP SPACE

                ; *************** PAGE 3 **************
   03000 003000 PV3:    .+0                            ; READ ONLY PAGE
   03001 125400 PV30:  INC      1,1                    ; SUBROUTINE: I:= I+1;
   03002 001400         JMP      0,3                   ; RETURN
                ; CASE TABLE
   03003 102001 PV31:  @PV20
   03004 102001         @PV20
   03005 002102'        PG2
         004000 .LOC    ./PVSIZE+1*PVSIZE              ; FILL UP SPACE
                ; **************************************
         002077'.LOC    PVSIZE*2+PGFOF                 ; TWO FRAMES
                PGTOF:                                 ; TOP OF FRAMES
   02077'024055 PG1:   LDA      1    .0                ; START: I:= 0;
   02100'006365         GOTO                           ;
   02101'101001         @PV10
   02102'030040 PG2:   LDA      2    CUR               ;
   02103'006013         STOPPROCESS                    ;
                ; ********** PROCESS DESCRIPTOR *********
                PG10:                                  ;  ...

         002145'.LOC PG10+CCOROUT
   02145'000000         0
         002104'.END PG10
```

11. REFERENCES.

1. MUS - SYSTEM INTRODUCTION
AND
MUS PROGRAMMERS GUIDE

RCSL: 44 - RT 759

2. DGC - EXTENDED ASSEMBLER MANUAL

DGC: 93 - 000040