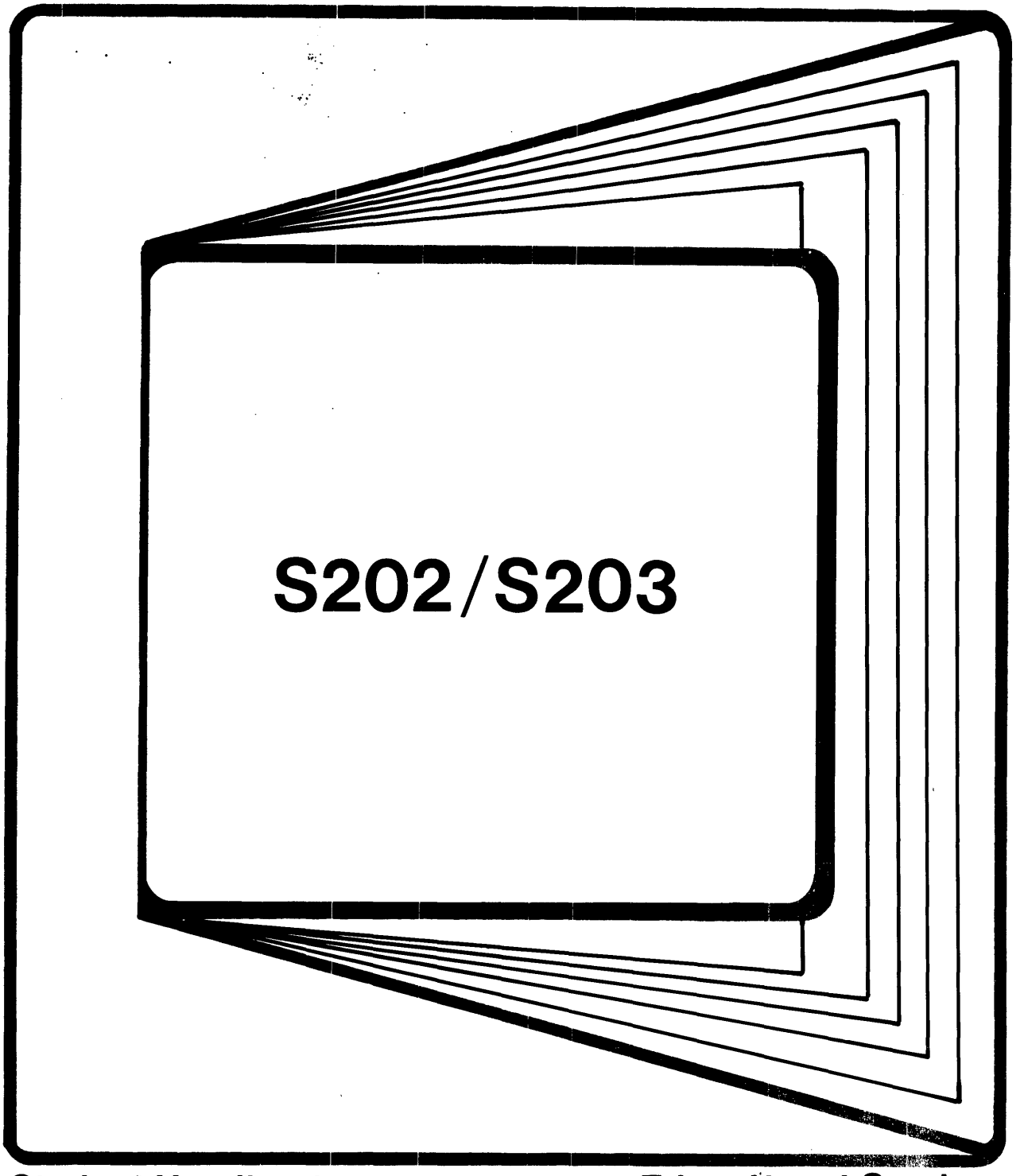
DataGeneral



Student Handbook

Educational Services

019-000031-01

NOTICE

Data General Corporation (DGC) has prepared this manual for use by DGC personnel and/or customers as a guide to the proper installation, operation, and maintenance of DGC equipment and software. The drawings and specifications contained herein are the property of DGC and shall neither be reproduced in whole or in part without DGC prior written approval nor be implied to grant any license to make, use, or sell equipment manufactured in accordance herewith.

DGC reserves the right to make changes without notice in the specifications and materials contained herein and shall not be responsible for any damages (including consequential) caused by reliance on the materials presented including but not limited to typographical or arithmetic errors, company policy and pricing information. The information contained herein on DGC software is summary in nature. More detailed information on DGC software is available in current released publications.

The following are trademarks of Data General Corporation, Westboro, Massachusetts:

U.S. Registered Trademarks	Trademarks
ECLIPSE	INFOS
NOVA	DASHER
	microNOVA

Table of Contents

LAB EXERCISE

MULTITASKING IMPLEMENTATION

The instructor will provide the .SR or .RB files necessary to accomplish the tasks below.

The student will be responsible for the code necessary to:

A. Identify these as individual tasks to the user task scheduler.
B. Implement the inter-task communications (transmits and receives)
C. Open channels for communications
D. Terminate the default task by killing itself

Monitor Task

Monitor the front panel switches every 10 seconds. If the switch setting equals minus one (177777), return to CLI. If bit Ø = 1, transmit the switch setting to the teletype task. If bit Ø = Ø, treat switches 1 - 15 as an arbitrary address of two ASCII characters. Transmit these two characters to the teletype task.

Teletype Task

Upon receipt of a message (indicating two characters ready for printing), output the two characters, high byte first, to the $TTO, followed by a carriage return and line feed. Then wait for receipt of next message.

Input Task

Accept a string of characters from $TTI without an echo. Pack the characters in a table high byte first. Upon receipt of the "ESC" character, transmit the number-of-characters-entered to the $LPT task. Upon receipt of the "=" character, transmit the number-of-characters-entered to the output task. Upon receipt of the "SHIFT L" character, return to CLI. Otherwise, return to accept more characters.

$LPT Task

Upon receipt of a message (indicating a table of characters to be printed) print the high byte then low byte of each address. This task knows where the table is; what it needs is the number of characters to be printed. At completion, return to await receipt of next message.

Output Task

Upon receipt of a message (indicating #-of-characters) convert the number to decimal and type this decimal quantity on $TTO followed by a carriage return and line feed. At completion, return to await receipt of next message.

S202

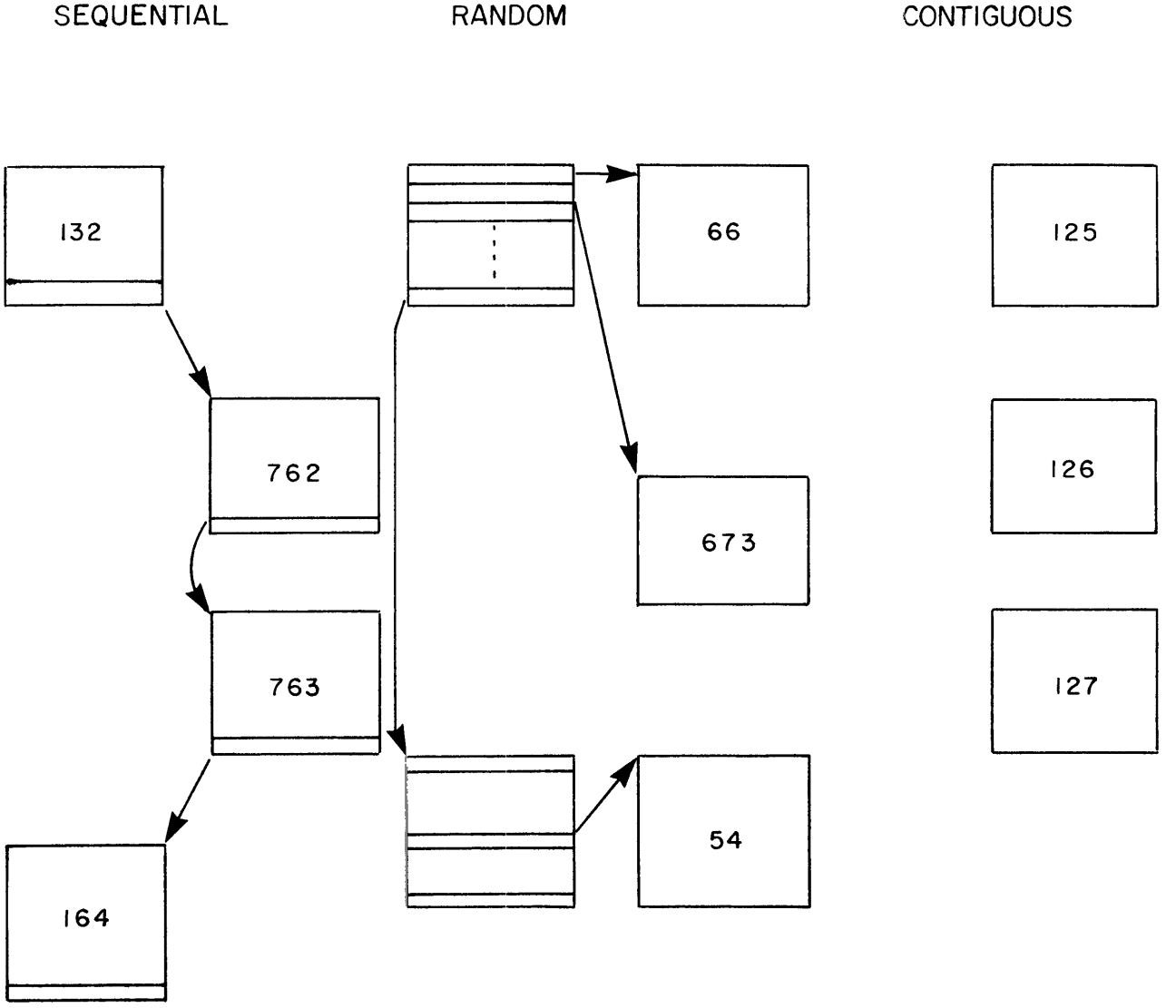
TOPIC OUTLINE

TOPIC OUTLINE (cont.)

Section I

DISC FILE STRUCTURES

DISC FILE STRUCTURES



CHARACTERISTICS OF FILE STRUCTURES

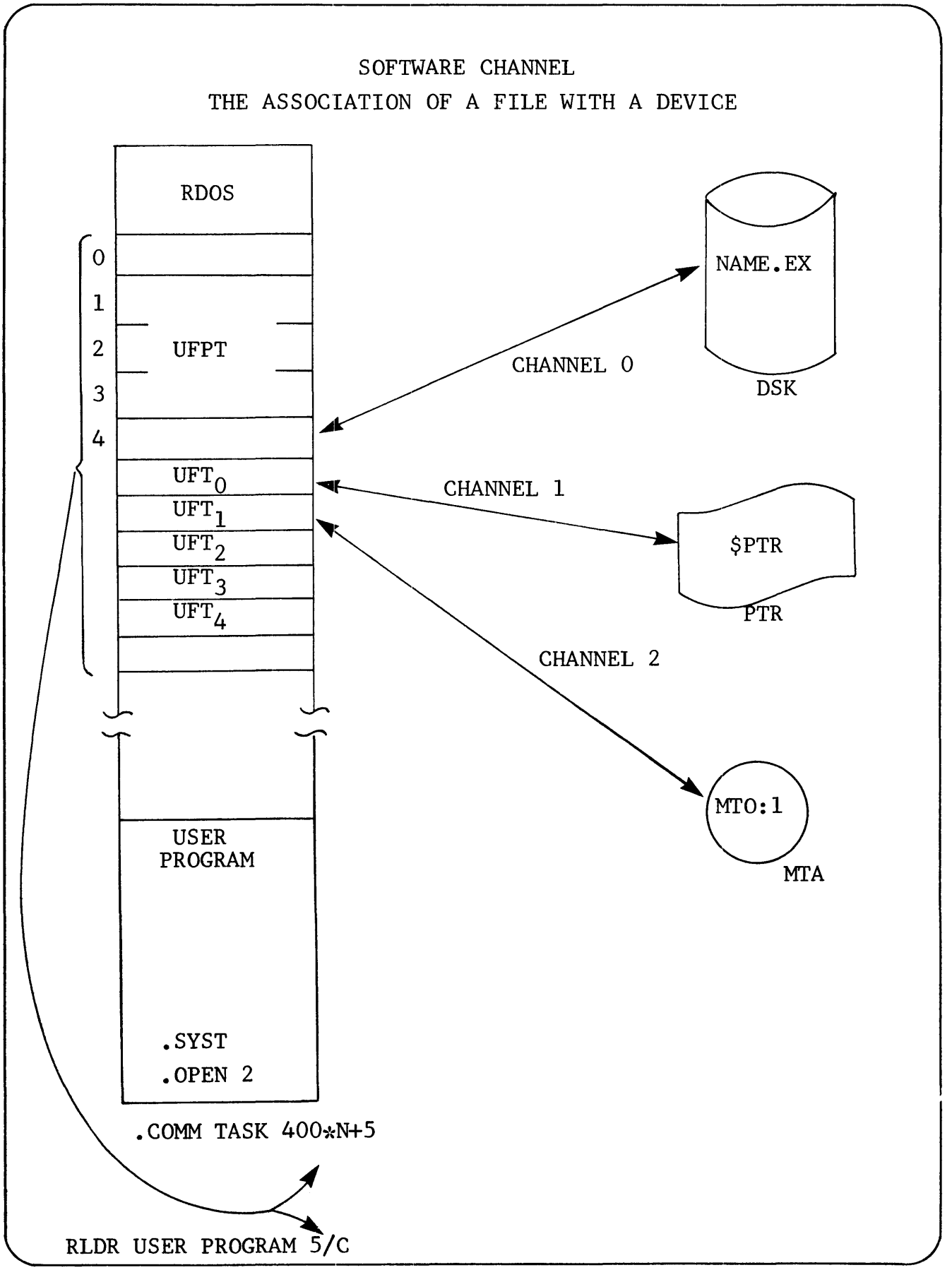
	ACCESS	OVERHEAD	GROWTH
SEQUENTIAL	LOSE	WIN	WIN
RANDOM	WIN	LOSE	WIN
CONTIGUOUS	WIN	WIN	LOSE

RESOLUTION UFD

0 1 2 3 4	FILENAME	UFTFN
5	EXTENSION	UFTEX
6	FILE ATTRIBUTES	UFTAT
7	LINK ATTRIBUTES	UFTLK
10	LOGICAL # OF LAST BLOCK	UFTBK
11	# BYTES IN LAST BLOCK	UFTBC
12	LOGICAL ADDRESS FIRST BLOCK	UFTAD
13	DATE LAST ACCESSED	UFTAC
14	DATE CREATED	UFTYD
15	TIME CREATED	UFTHM
16	UFD TEMPORARY	UFTP1
17	UFD TEMPORARY	UFTP2
20	FILE USE COUNT	UFTUC
21	DCT LINK	UFTDL

Section II

FILE and DEVICE ACCESS



RDOS FILE/DEVICE ACCESS

SOFTWARE CHANNELS

GENERAL CHARACTERISTICS

To access a file (or device) it must first be associated with a software channel.

64 possible channels

The quantity a program may need is specified by the /C local switch in the RLDR command

The actual channel numbers available to an assembly program are Ø through N-1 where N is the argument of the /C local switch

User File Tables are built by RDOS to hold the UFD's for associated channels

RDOS COMMAND SUMMARY

SYSTEM OPEN CALLS

FUNCTION	CLI	ASM	FORT
SHARED READ/WRITE MULTIUSER ACCESS	N/A	ACØ - BYTEPOINTER TO FILE NAME AC1 - CHAR, MASK . SYS . SYSTEM . OPEN n	CALL OPEN (CH#, FILE, MODE/ARRAY, ERROR, SIZE) MODE = Ø
SINGLE USER WRITE ACCESS	N/A	ACØ - BYTEPOINTER TO FILENAME AC1 - CHAR. MASK . SYSTEM . EOPEN n	CALL OPEN (SEE ABOVE) MODE = 3
MULTIUSER READ ACCESS	N/A	ACØ - BYTEPOINTER TO FILE NAME AC1 - CHAR. MASK . SYSTEM . ROPEN n	CALL OPEN (SEE ABOVE) MODE = 1
APPEND OUTPUT TO END OF FILE	N/A	ACØ - BYTEPOINTER TO FILENAME AC1 - CHAR. MASK . SYSTEM . APPEND n	CALL APPEND (SEE ABOVE)

RDOS COMMAND SUMMARY

SYSTEM OPEN CALLS

FUNCTION	CLI	ASM	FORT
INDIVIDUALLY CLOSE A SINGLE FILE	N/A	.SYSTEM .CLOSE n	CALL CLOSE (CH.#, ERROR)
GLOBALLY CLOSE ALL OPEN FILES	N/A	.SYSTEM .RESET	CALL RESET

RDOS FILE/DEVICE ACCESS

GENERAL CHARACTERISTICS

Buffered read/write access

Buffered by RDOS in system buffers

Four basic types

- character
- line
- sequential byte
- record

Unbuffered read/write access

Direct transfer from device to user program

Two basic types

- direct block disc I/O
- free format magnetic/cassette tape I/O

RDOS COMMAND SUMMARY

BUFFERED READ/WRITE ACCESS

FUNCTION		CLI	ASM	FORT
CHARACTER I/O	READ	N/A	.SYSTEM .GCHAR ACØ: BITS 9-15=CHAR BITS Ø-8=CLEARED	ACCEPT
	WRITE	N/A	ACØ: BITS 9-15=CHAR BITS Ø-8=IGNORED .SYSTEM .PCHAR	TYPE
LINE I/O	READ	N/A	ACØ-BYTEPOINTER TO USER CORE AREA .SYSTEMS .RDL n AC1-READ BYTE COUNT	READ
	WRITE	N/A	ACØ-BYTEPOINTER TO USER CORE AREA .SYSTEM .WRL n AC1-WRITE BYTE COUNT	WRITE

RDOS COMMAND SUMMARY

BUFFERED READ/WRITE ACCESS

FUNCTION		CLI	ASM	FORT
SEQUENTIAL BYTE I/O	READ	N/A	ACØ - BYTEPOINTER TO USER CORE AREA AC1 - READ BYTE COUNT .SYSTEM .RDS n	READ BINARY
	WRITE	N/A	ACØ - BYTEPOINTER TO USER CORE AREA AC1 - WRITE BYTE COUNT .SYSTEM .WRS n	WRITE BINARY
RECORD I/O	READ	N/A	ACØ - BYTEPOINTER TO USER CORE AREA AC1 - RECORD NUMBER .SYSTEM .RDR n	CALL READR (CH#, SREC, ARRAY, NREC, ERROR) CALL RDRW (SAME AS ABOVE)
	WRITE	N/A	ACØ - BYTEPOINTER TO USER CORE AREA AC1 - RECORD NUMBER .SYSTEM .WRR n	CALL WRITR (SAME AS ABOVE) CALL WRTRW (SAME AS ABOVE)

RDOS COMMAND SUMMARY

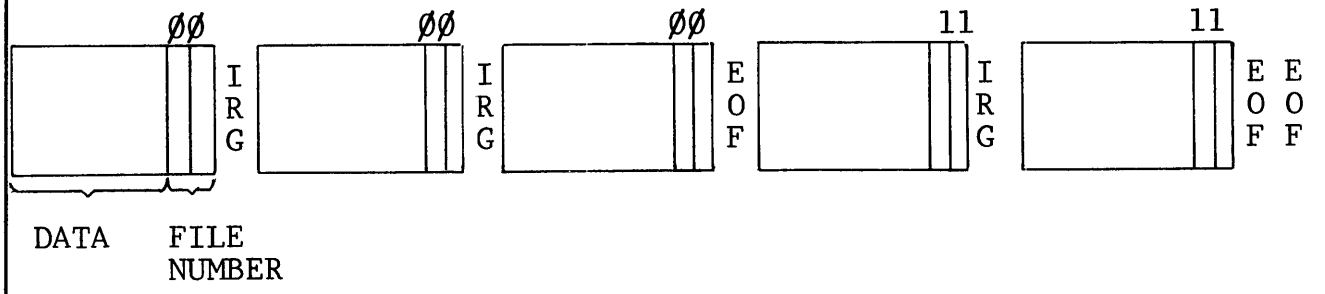
UN-BUFFERED READ/WRITE ACCESS

	FUNCTION	CLI	ASM	FORT
DIRECT DISC BLOCK ACCESS	READ	N/A	ACØ - CORE ADDRESS OF USER AREA AC1 - STARTING BLOCK NUMBER AC2 - BITS Ø-7=# OF BLKS .SYSTEM .RDB n	CALL RDBLK (CH#, SBLK, ARRAY, n BLOCK, ERROR)
	WRITE	N/A	ACØ - CORE ADDRESS OF USER AREA AC1 - STARTING BLOCK NUMBER AC2 - BITS Ø-7=# OF BLKS .SYSTEM .WRB n	CALL WRBLK (SEE ABOVE)

TAPE FILE STRUCTURES

FIXED FORMAT TAPE FILES

PHYSICAL ORGANIZATION



- Tape reel may have Ø to 1ØØ files
- Files are unlimited in size
- Physical data records have 255 words of data and 2 ID words, each containing tape file number
- Mag tape and cassette tape use same format
- Inter Record Gaps between physical records
- End-Of-File after last record in file
- Two End-Of-File after last file
- Last record packed with nulls as required
- Files referenced by numerical order on tape reel

  i.e., first file is Ø
  second file is 1 etc.

TAPE FILE STRUCTURES

FREE FORMAT TAPE FILES

PHYSICAL ORGANIZATION

* USER DEFINED *

GENERAL CHARACTERISTICS

Essentially machine level access to magnetic or cassette tape

2 to 4Ø96 words per data record

1 to 4Ø95 records per file

Unlimited number of files on a tape reel

Data encoding depends on particular tape unit

Additional capabilities

- Space tape backward or forward within file
- Space to start of new file
- Rewind reel
- Write end-of-file
- Read transport status

RDOS COMMAND SUMMARY

UN-BUFFERED READ/WRITE ACCESS

FUNCTION		CLI	ASM	FORT
MAG/CAS TAPE ACCESS FREE FORMAT	OPEN CHANNEL FOR FREE FORMAT ACCESS	N/A	ACØ: BYTEPOINTER TO TAPE UNIT NAME AC1: CHAR, MASK .SYSTEM .MTOPD n	CALL MTOPD (CH#, NAME, CHAR MASK, ERROR)
	PERFORM FREE FORMAT TAPE I/O	N/A	SEE P. 4-43 IN RDOS USER's MANUAL .SYSTEM .MTDIO n	CALL MTDIO (CH#, . . .) SEE P. 3-17 IN, FORTRAN IV USER's MANUAL

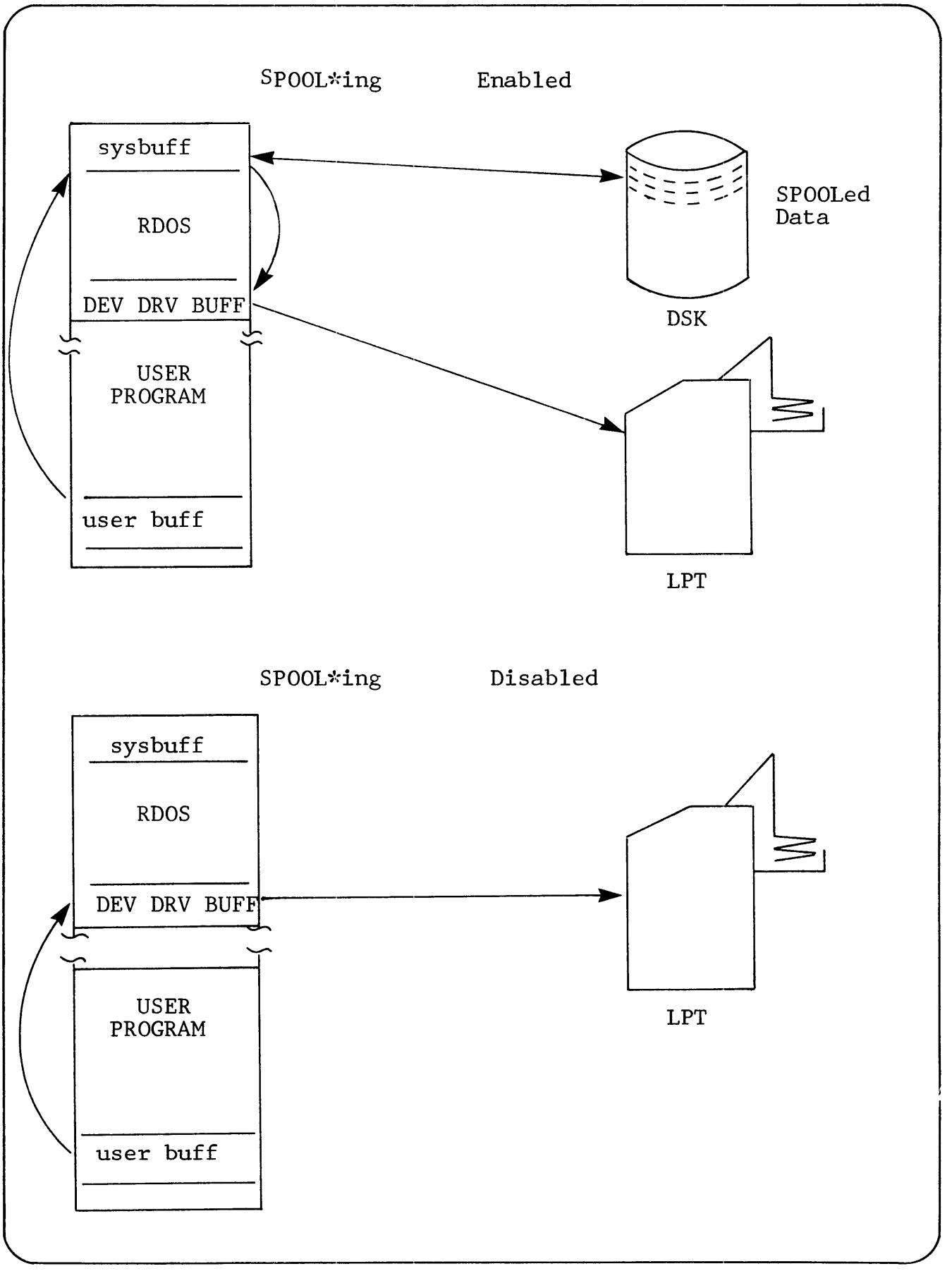
RDOS COMMAND SUMMARY

DEVICE EQUIVALENCE

FUNCTION	CLI	ASM	FORT
EQUIVALENCE NEW NAME FOR OLD NAME OF DEVICE	EQUIV NEW NAME OLD NAME	ACØ:BYTEPOINTER TO OLDNAME AC1:BYTEPOINTER TO NEWNAME . SYSTEM . EQUIV	CALL EQUIV (OLDNAME, NEWNAME, ERROR)

Section III

SPOOLING



SPOOLING

GENERAL CHARACTERISTICS

Permits queuing of output data on disc for one or more spoolable devices simultaneously

Allows CPU to become available once queue is established

Spoolable devices: (also second devices)

$DPO

$LPT

$PLT

$PTP

$TTO

$TTP

RDOS COMMAND SUMMARY

SPOOLING COMMANDS

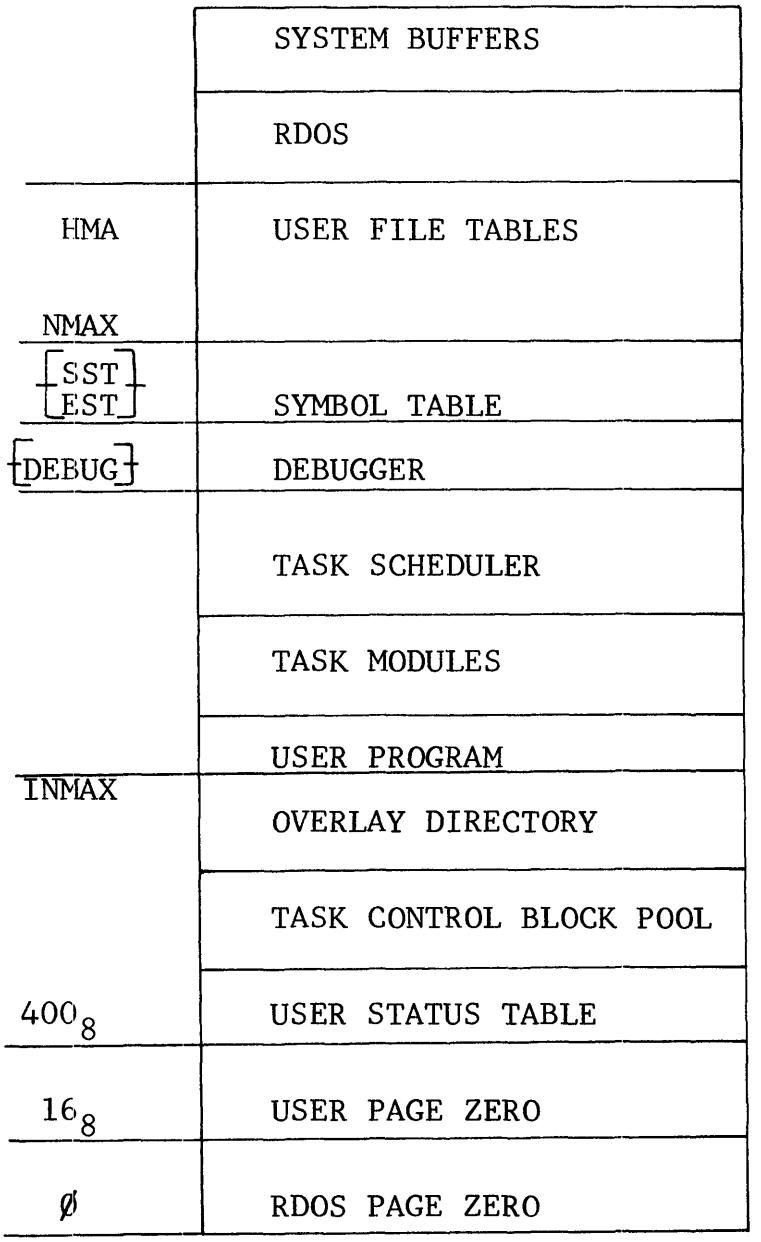
FUNCTION	CLI	ASM	FORT
Terminate spooling and current output	SPKILL DEVICENAME ↵	AC0: bytepointer to devicename .SYSTEM .SPKL	CALL SPKILL (DEVICENAME, ERROR)
Disable Future Spooling	SPDIS DEVICENAME ↵	AC0: BYTEPOINTER TO DEVICENAME .SYSTEM .SPDA	CALL SPDIS (DEVICENAME, ERROR)
Enable Future Spooling	SPEBL DEVICENAME ↵	AC0: BYTEPOINTER TO DEVICENAME .SYSTEM .SPEA	CALL SPEBL (DEVICENAME, ERROR)

Section IV

CORE MANAGEMENT TECHNIQUES

CORE CONFIGURATION

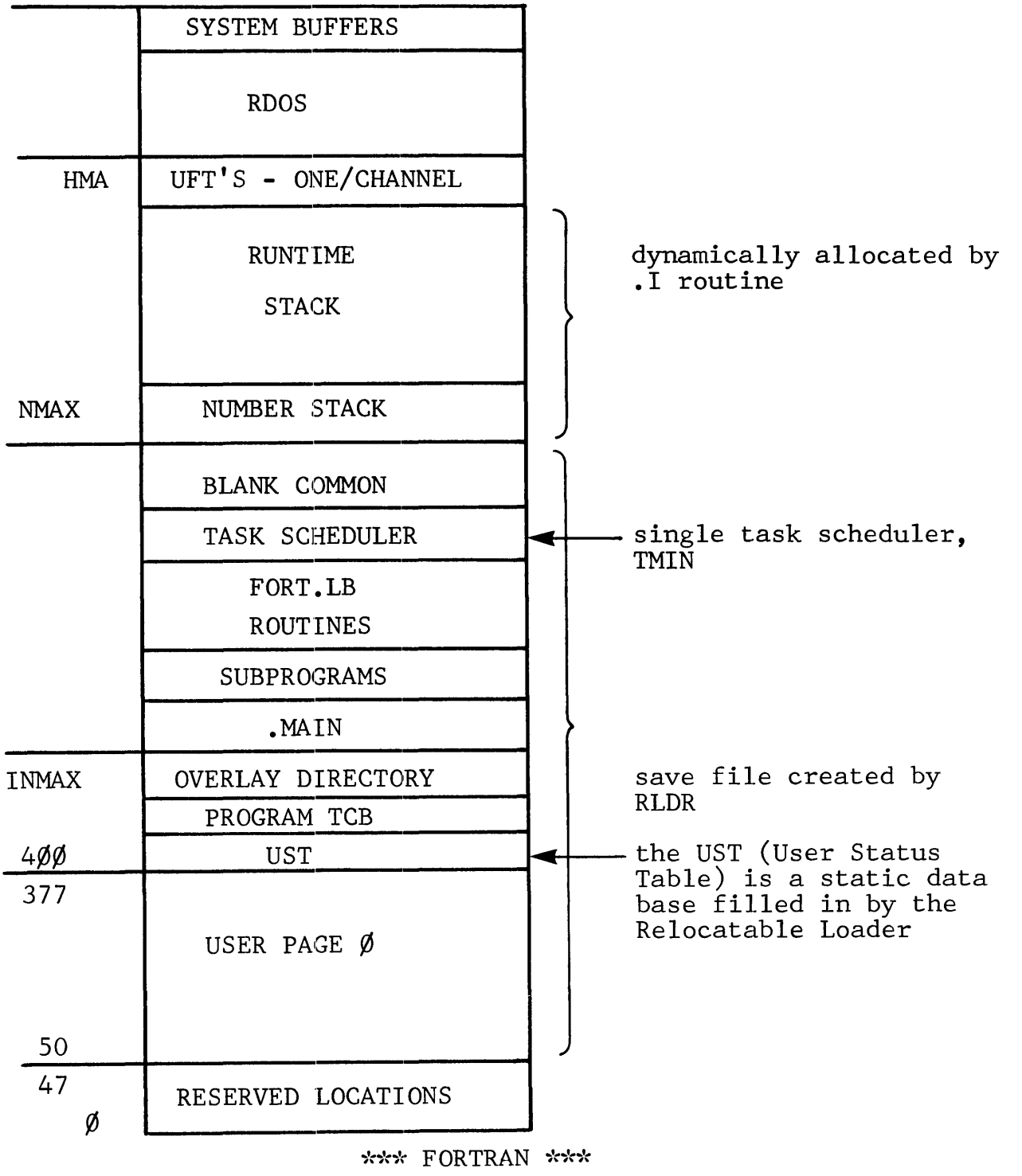
UNMAPPED - SINGLE PROGRAM ENVIRONMENT



*** ASSEMBLER ***

CORE CONFIGURATION

UNMAPPED SINGLE PROGRAM ENVIRONMENT



*** FORTRAN ***

CORE MANAGEMENT TECHNIQUES

PROGRAM CHAINING

GENERAL CHARACTERISTICS

A program can terminate its own execution and invoke another program to execute in core - this is chaining

The currently running program is totally destroyed when the new program overwrites it; there is no normal return from a chain call; no image of the current program is saved anywhere

No limit on number of programs chained

Chained programs can communicate using common disc files

RDOS COMMAND SUMMARY

PROGRAM CHAIN COMMANDS

FUNCTION	CLI	ASM	FORT
EXECUTE PROGRAM CHAIN PROGRAM LEVEL N ⟶ N	CHAIN NAME↵	AC1: 1ØØØØØ ACØ: BYTE POINTER TO .SV FILENAME .SYSTEM .EXECUTE	CALL FCHAN (NAME) CALL CHAIN (NAME, ERROR)

CORE MANAGEMENT TECHNIQUES

PROGRAM SWAPPING

GENERAL CHARACTERISTICS

All programs execute at an execution level

- level determined by how program execution was initiated
- CLI starts at highest level: Ø
- Four additional levels: 1,2,3 and 4

A program can suspend its own execution, save a current core image on disc, and invoke another program - this is swapping

New core resident program runs at lower program execution level

Core image picture saved as random file using disc block 7 as file index block

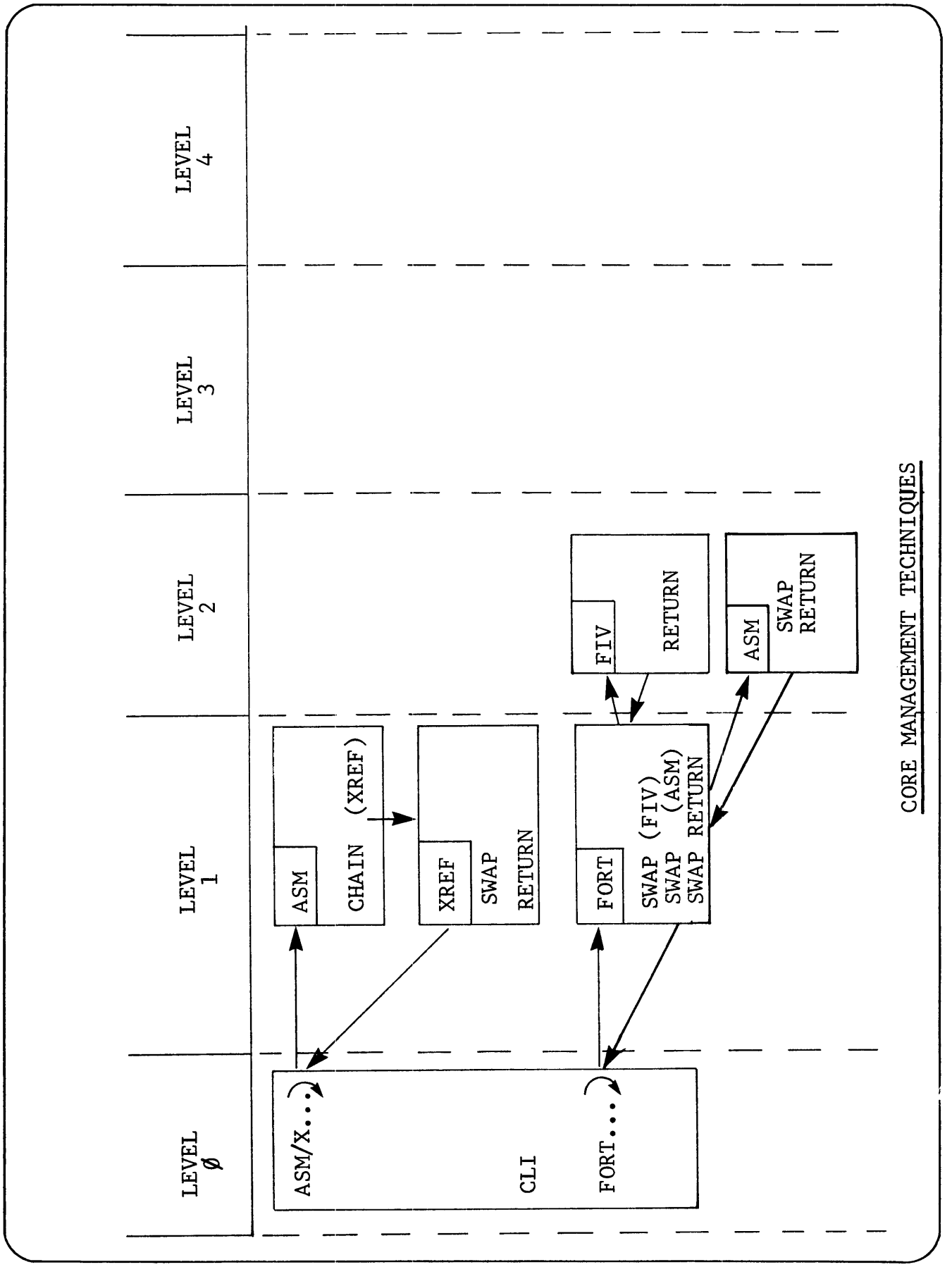
Core image is from 16_8 thru ZMAX and 4ØØ thru NMAX and channel information

Swapped programs can communicate using common disc files

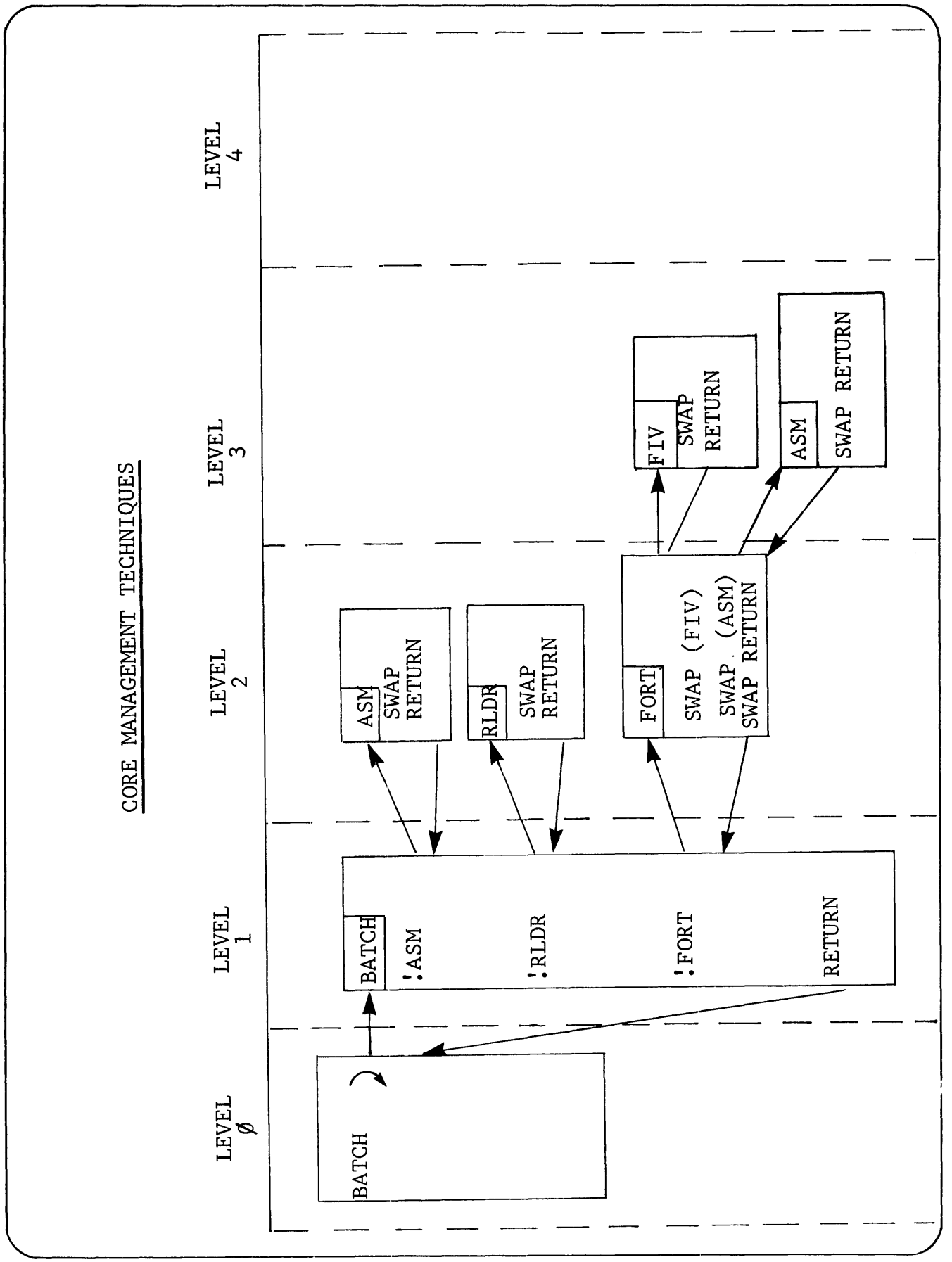
RDOS COMMAND SUMMARY

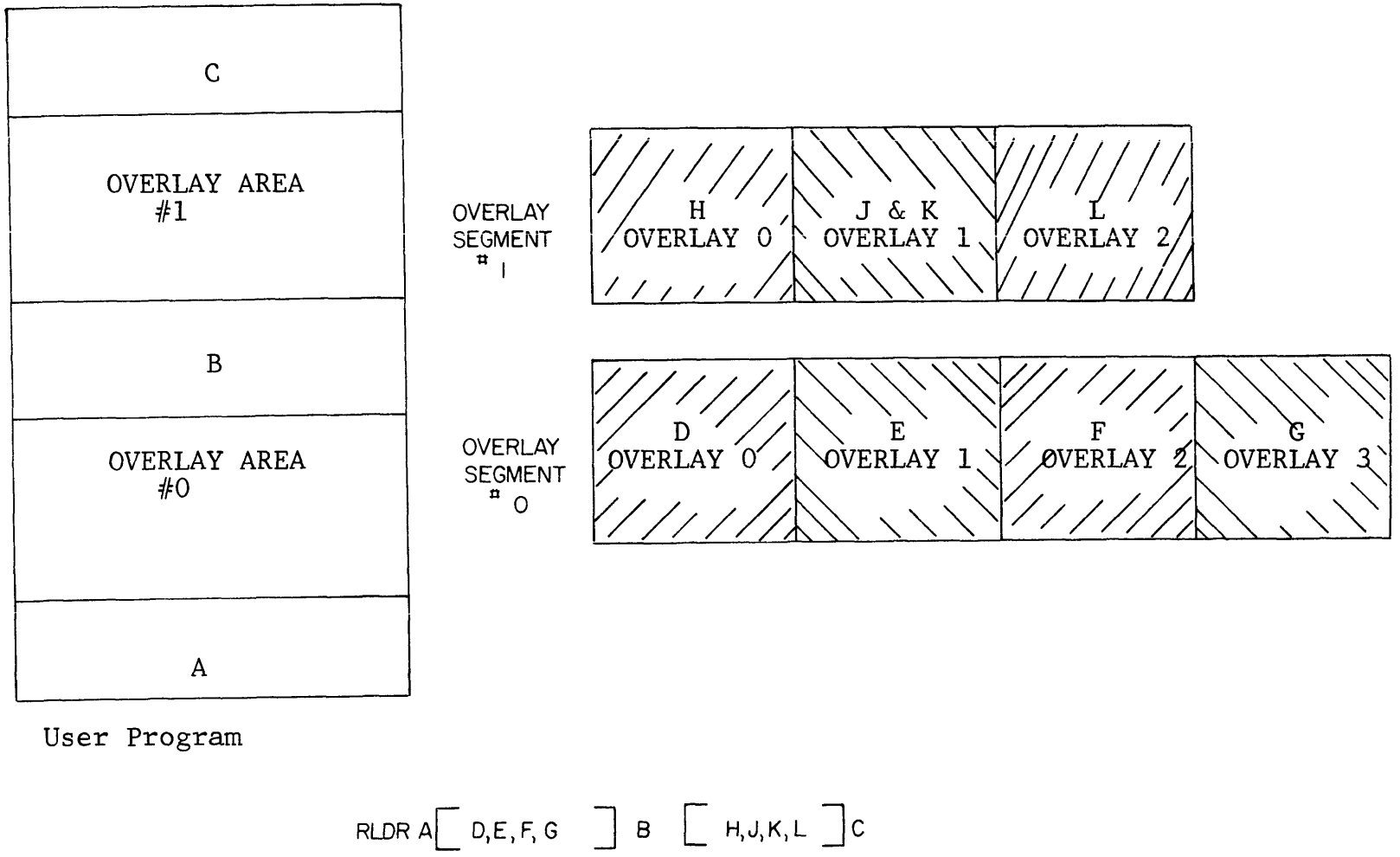
PROGRAM SWAP COMMANDS

FUNCTION	CLI	ASM	FORT
EXECUTE PROGRAM SWAP PROGRAM LEVEL N → N + 1	NAME	AC0 = BYTE POINTER TO .SV FILENAME AC1 = 0 .SYSTEM .EXECUTE	CALL FSWAP (NAME) CALL SWAP (NAME, ERROR)
NORMAL RESTORATION OF HIGHER PROGRAM LEVEL N → N - 1	N/A	.SYSTEM .RTN	CALL FBACK CALL BACK
ERROR RESTORATION OF HIGHER PROGRAM LEVEL N → N - 1	N/A	.SYSTEM .ERTN	CALL EBACK (ERROR)
NORMAL RESTORATION OF CLI FROM ANY LEVEL N → CLI	N/A	N/A	CALL EXIT STOP {MESSAGE}



CORE MANAGEMENT TECHNIQUES





CORE MANAGEMENT TECHNIQUES

SINGLE TASK ENVIRONMENT

OVERLAYS

GENERAL CHARACTERISTICS

Overlay area in save file corresponds to overlay segment in overlay file

Overlay area is sized to fit largest overlay in corresponding overlay segment, then rounded to M*4ØØ$_8$

Overlay File is contiguous

Each overlay in an overlay segment alloted same number of disc blocks to facilitate system transfers

Overlays in FORTRAN IV must have overlay statement in at least one relocating binary within each overlay

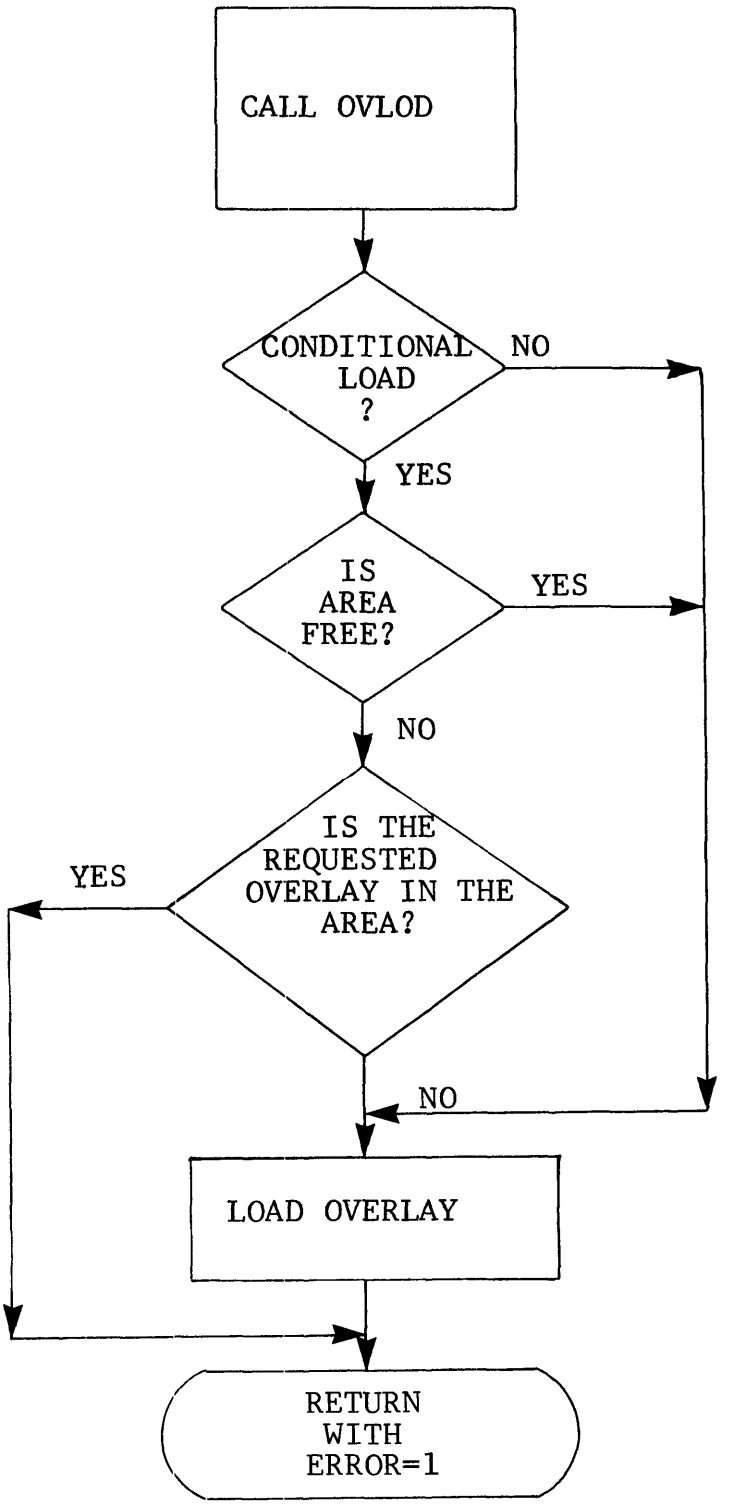
Overlays in Assembly Language should have a .ENTO in at least one relocatable binary within each overlay

Within an overlay file there can be:

- up to 128 decimal overlay segments
- up to 256 decimal overlays in each segment

To access overlays, the user must

- associate software channel with overlay file by using special overlay open call
- load the overlay desired
    * load may be unconditional in which case the overlay will be put in core irregardless of whether it is in core already
    * load may be conditional in which case the overlay will be put in core only if it is not already in core

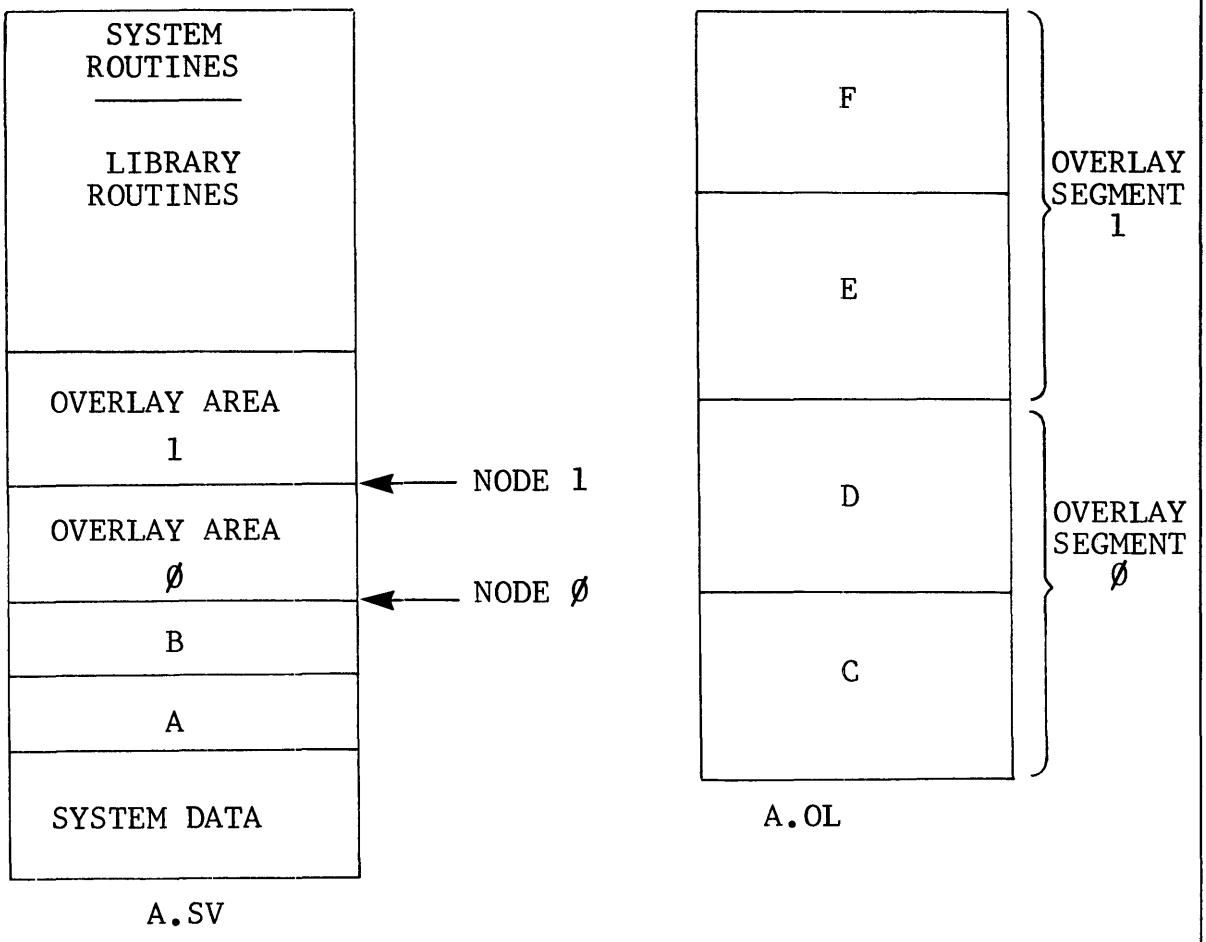


CORE MANAGEMENT TECHNIQUES

OVERLAYS

RLDR A B [C, D] [E, F] LIBRARIES

A.RB AND B.RB ALWAYS CORE RESIDENT
C.RB OR D.RB ALTERNATIVELY CORE RESIDENT
E.RB OR F.RB ALTERNATIVELY CORE RESIDENT

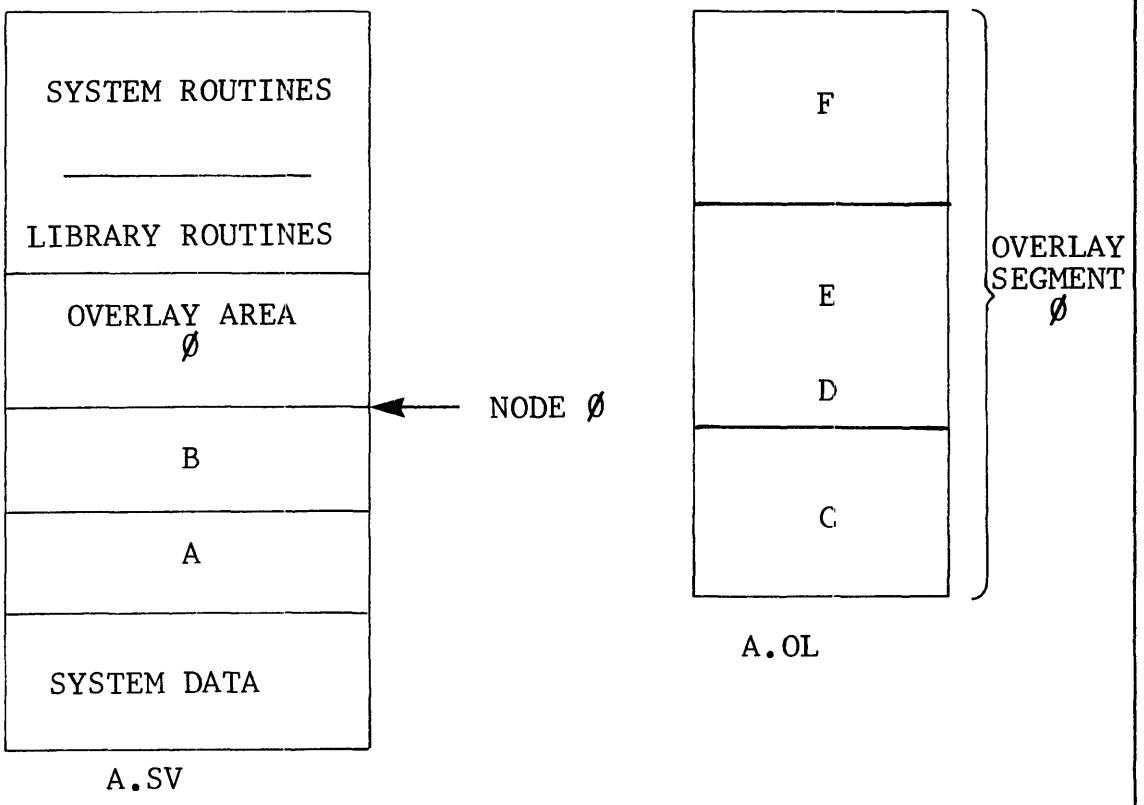


CORE MANAGEMENT TECHNIQUES

OVERLAYS

RLDR A B [C, D E, F] LIBRARIES

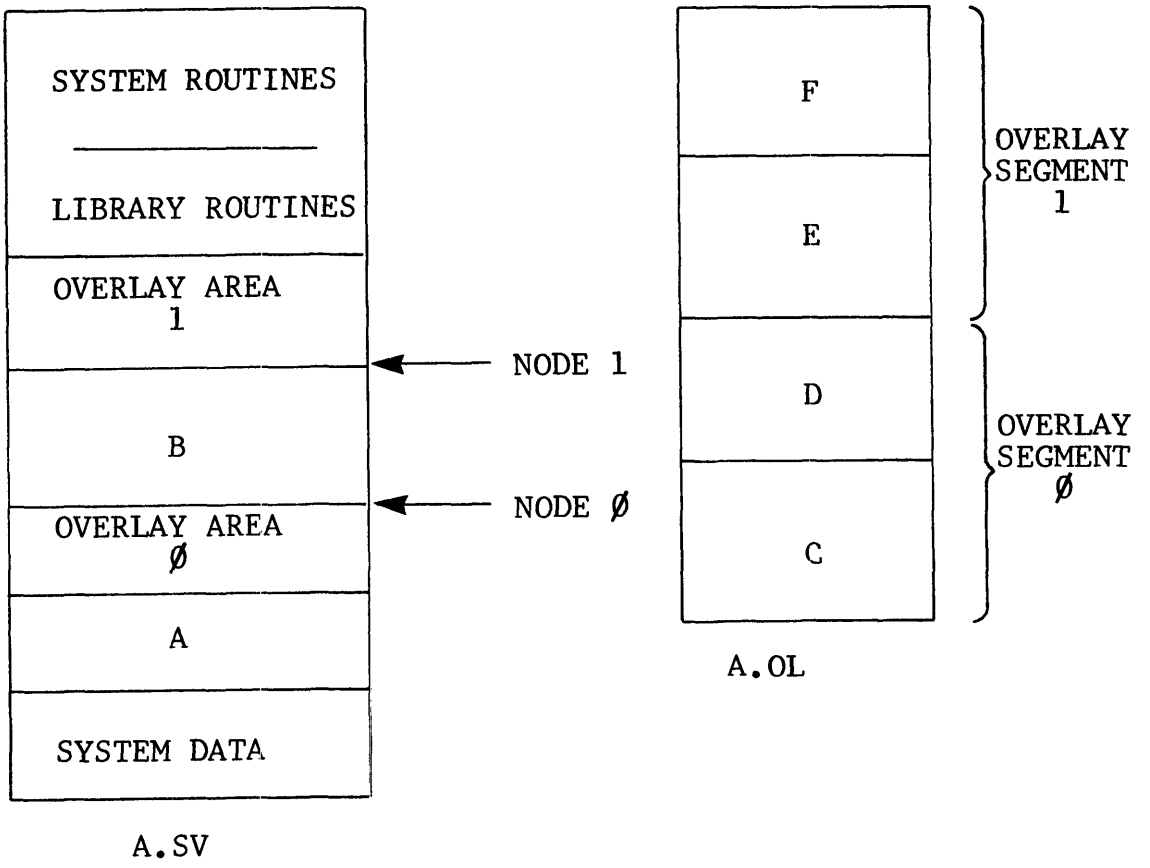
A.RB AND B.RB ALWAYS CORE RESIDENT
C.RB OR, D.RB WITH E.RB OR: F.RB ALTERNATIVELY CORE RESIDENT



CORE MANAGEMENT TECHNIQUES

OVERLAYS

```
RLDR  A  [C,  D]  B  [E,  F]     LIBRARIES
      A.RB AND B.RB  ALWAYS CORE RESIDENT
      C.RB OR D.RB ALTERNATIVELY CORE RESIDENT
      E.RB OR F.RB ALTERNATIVELY CORE RESIDENT
```

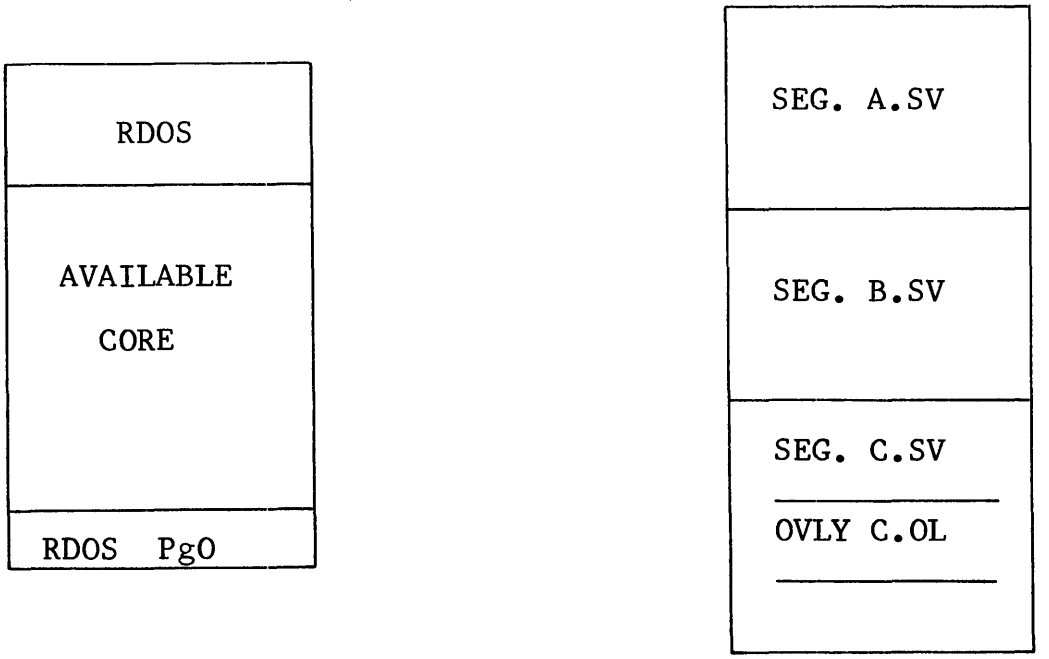


RDOS COMMAND SUMMARY

SINGLE TASK CORE MANAGEMENT TECHNIQUES - OVERLAYS

FUNCTION	CLI	ASM	FORT
ASSOCIATE A SOFTWARE CHANNEL WITH AN OVERLAY FILE	N/A	AC0: BYTEPOINTER TO OVERLAY FILE .SYSTEM .OVOPN n	CALL OVOPN (CH#, FILENAME, ERROR)
LOAD AN OVERLAY IN A SINGLE TASK ENVIRONMENT	N/A	AC0: AREA#/OVERLAY# AC1: -1:UNCONDITIONAL 0:CONDITIONAL .SYSTEM .OVLOD n	CALL OVLOD (CH#, OVERLAYMENT, FLAG, ERROR) FLAG = 0: UNCONDITIONAL LOAD 1:CONDITIONAL LOAD

PROGRAM SEGMENTATION



Available Core < User Program

SWAP from segments that have to be resumed.

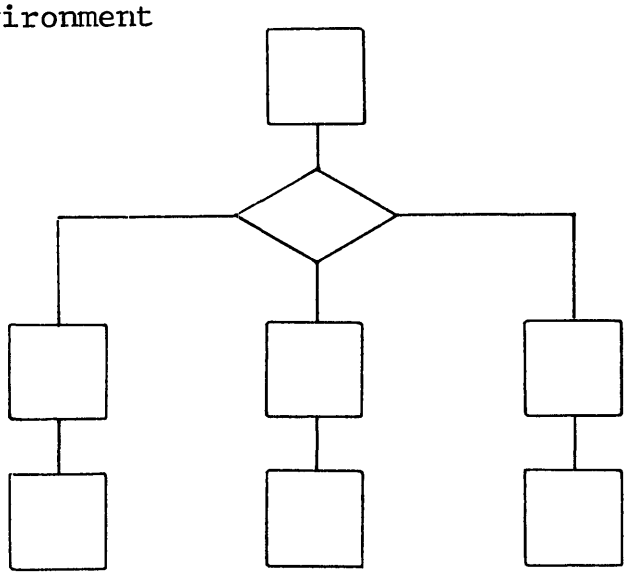
CHAIN from segments that run one time only.

OVERLAY infrequently used subroutines.

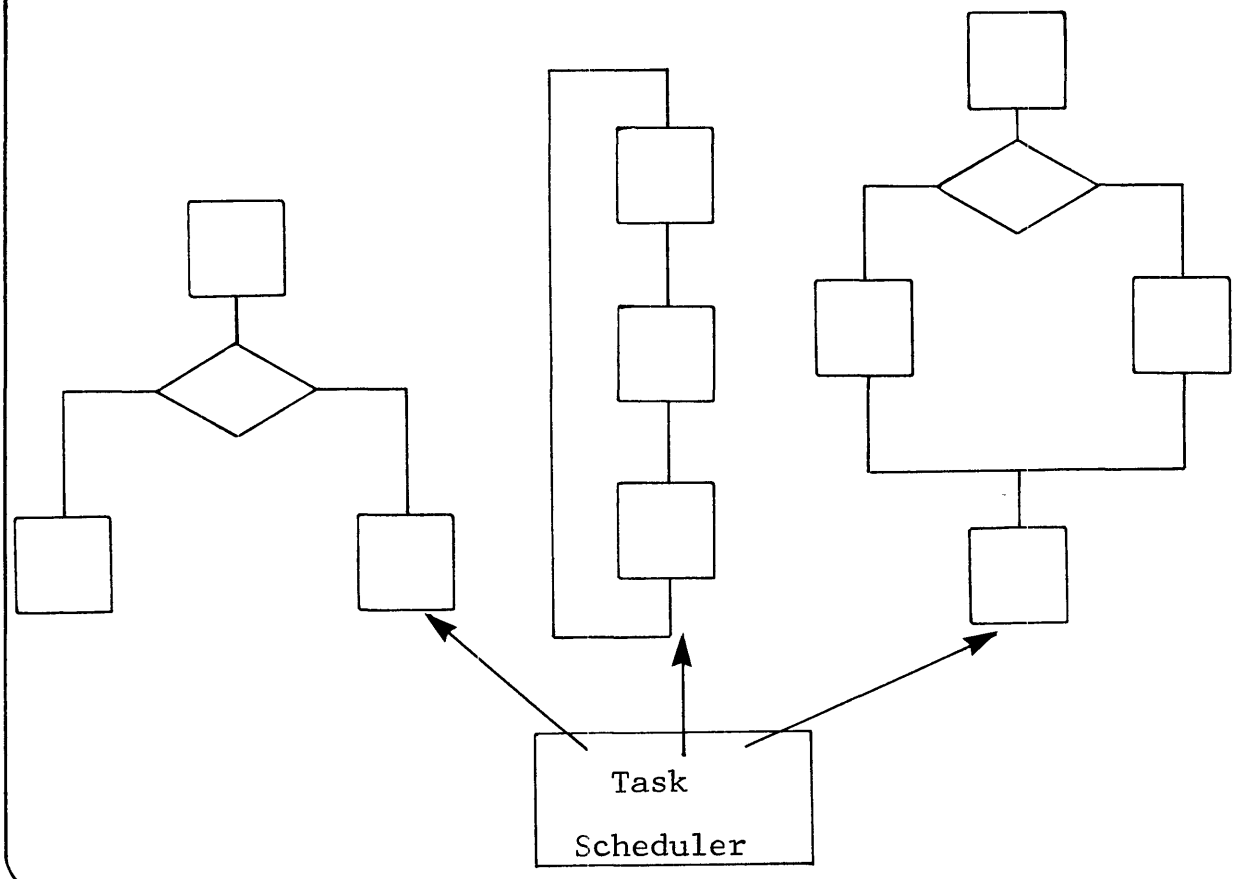
Section V

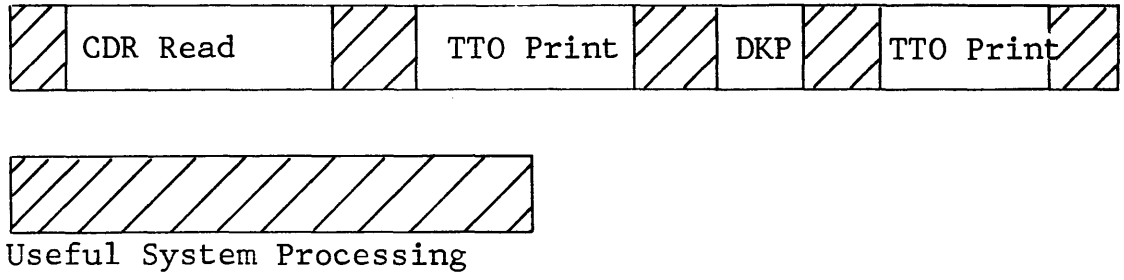
MULTI-TASKING

Single Task Environment

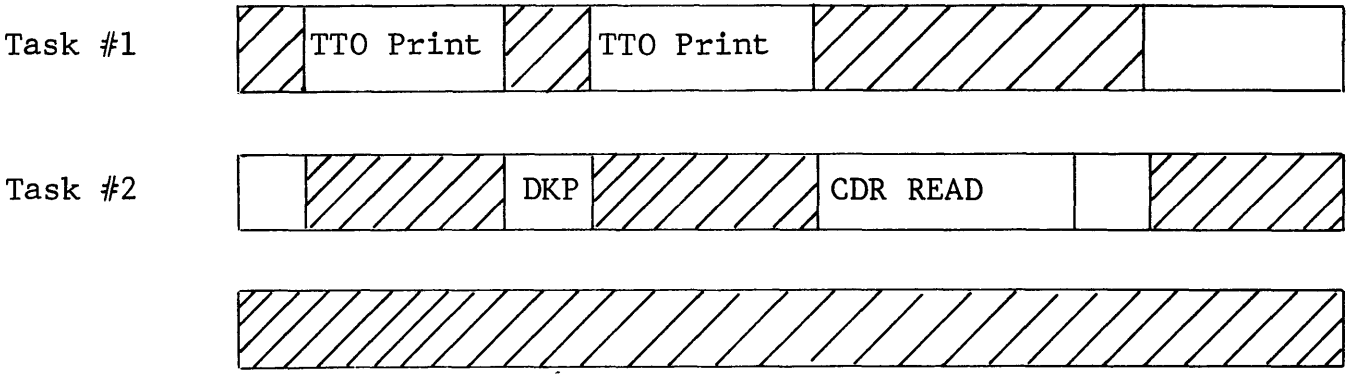


Multi-task Environment





Single Task Operating System



Multiple Task Operating System

I/O Processing or Task Suspension

Useful System Processing

TASK STATE	TASK STATE DESCRIPTION
EXECUTING	A TASK HAS CONTROL OF THE CENTRAL PROCESSING UNIT.
READY	A TASK IS AVAILABLE FOR EXECUTION, BUT IT IS WAITING FOR ALL HIGHER PRIORITY READY TASKS TO BECOME LOWERED IN PRIORITY AND FOR THE CURRENTLY EXECUTING TASK TO BE REDUCED TO THE READY, SUSPENDED, OR DORMANT STATE.
SUSPENDED	A TASK IS AWAITING THE OCCURRENCE OR COMPLETION OF SOME SYSTEM OR TASK CALL, OR IT IS AWAITING A SPECIFIC REAL-TIME EVENT.

TASK STATES

TASK SCHEDULING

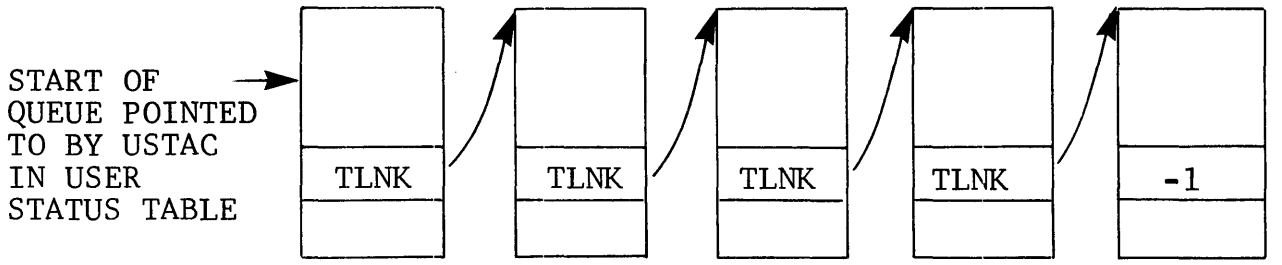
What causes task scheduling?

1. System Calls - When the executing task issues an I/O system call, the user scheduler reduces the executing task to the suspended state and updates its TCB. At this time control is passed to the highest priority ready task. At the completion of system call processing the issuing task is then raised to the ready state to compete with other tasks in the ready state for execution.

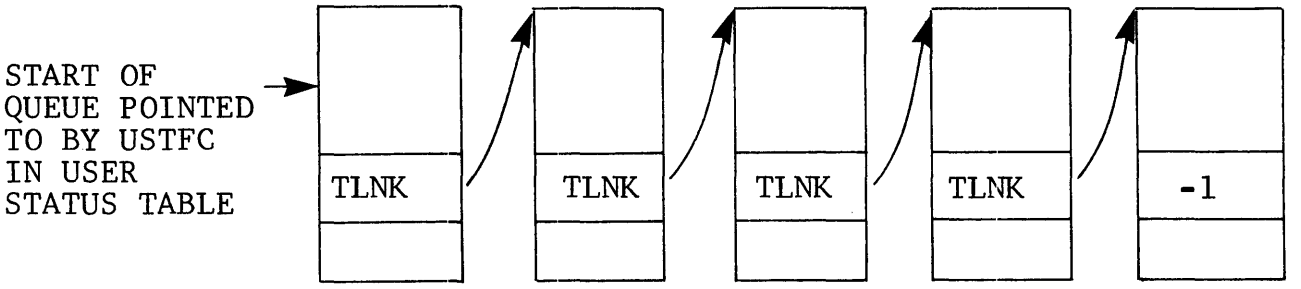
2. Task Calls - When the executing task issues a task call control is passed to a task module. The task module does the actual task call processing and then passes control to the user scheduler. The user scheduler updates the TCB and passes control to the highest priority ready task.

3. User defined Interrupt - When the executing task is interrupted by a user defined interrupt control is passed directly to RDOS's interrupt dispatch routine (JMP @1). Upon exit (.UIEX), control will go to the user scheduler if AC1 $\neq$ 0. If AC1 = 0 control returns to the interrupted executing task.

ACTIVE TCB QUEUE



INACTIVE (FREE) TCB QUEUE



TASK CONTROL BLOCK QUEUES

RDOS COMMAND SUMMARY
TASK ACTIVATION CALLS

FUNCTION	CLI	ASM	FORT
ACTIVATE A TASK	N/A	AC0: ID & PRIORITY AC1: Task ENTRY ADDRESS AC2: MESSAGE TO TASK .TASK	CALL FTASK (TASKNAME, $ERROR, PRI, [IASM] CALL ITASK (TASKNAME, ID, PRI, ERROR [IASM] CALL ASSOC (TASKNAME, ID PRI,ERROR, [IASM])

TASK TERMINATION CALLS

FUNCTION	CLI	ASM	FORT
KILL THE EXECUTING TASK	N/A	.KILL	CALL KILL
KILL ALL TASKS OF A GIVEN PRIORITY LEVEL	N/A	AC0: PRIORITY .AKILL	CALL AKILL (PRIORITY)
KILL A TASK WITH SPECIFIED ID	N/A	AC1: TASK ID .TIDK	N/A
KILL A TASK WITH SPECIFIED ID IMMEDIATELY	N/A	AC1: TASK ID .ABORT	CALL ABORT (ID,IER)

RDOS COMMAND SUMMARY

TASK SUSPENSION CALLS

FUNCTION	CLI	ASM	FORT
SUSPEND THE EXECUTING TASK	N/A	.SUSP	CALL SUSP
SUSPEND ALL TASKS OF A GIVEN PRIORITY LEVEL	N/A	ACØ: PRIORITY .ASUSP	CALL ASUSP (PRIORITY)
SUSPEND A TASK WITH SPECIFIED ID	N/A	AC1: TASK ID .TIDS	CALL HOLD (ID,ERROR)

RDOS COMMAND SUMMARY

INTERTASK COMMUNICATION

FUNCTION	CLI	ASM	FORT
SUSPEND THE EXECUTION TASK UNTIL A MESSAGE IS RECEIVED	N/A	ACØ: MESSAGE ADDRESS AC1: MESSAGE .REC	CALL REC (MESSAGE-KEY, MESSAGE—DESTINATION)
SEND A MESSAGE TO A TASK, IF RECEIVING TASK IS SUSPENDED AWAITING THIS MESSAGE, THEN READY THE RECEIVING TASK	N/A	ACØ: MESSAGE ADDRESS AC1: MESSAGE .XMT	CALL XMT (MESSAGE KEY, MESSAGE SOURCE, $ERROR RETURN)
SEND A MESSAGE TO A TASK, REMAIN SUSPENDED UNTIL MESSAGE IS RECEIVED THEN READY THE TRANSMITTING TASK	N/A	ACØ: MESSAGE ADDRESS AC1: MESSAGE .XMTW	CALL XMTW (MESSAGE KEY, MESSAGE SOURCE, $ERROR RETURN)

CORE MANAGEMENT TECHNIQUES (OVERLAYS)

MULTI-TASK ENVIRONMENT

GENERAL CHARACTERISTICS

.Overlay construction and structure same as in single task program environment

.If only one task in the multi-task program issues overlay system and task calls, use same procedures as in a single task environment

.When more than one task must issue overlay system and task calls, special considerations must be introduced:

- Overlays and overlay areas must be carefully managed to prevent loading of new overlays when another task is currently using the present overlay
- User task scheduler retains a record of overlay usage of each overlay area, i.e., overlay use count = OUC

1. Overlay may be loaded in overlay area only if OUC = Ø for that area

2. OUC is incremented whenever overlay is loaded unconditionally

3. OUC is incremented if overlay is loaded conditionally and it is present

4. OUC is unchanged if overlay is loaded conditionally and it is not present and area is not free

5. OUC is set to 1 if overlay is loaded conditionally and it is not present and area is free

6. OUC is decremented whenever a task releases its usage of the overlay.

RDOS COMMAND SUMMARY
MULTITASK CORE MANAGEMENT TECHNIQUES - OVERLAYS

FUNCTION			ASM	FORT
LOAD AN OVERLAY IN A MULTITASK ENVIRONMENT			ACØ:AREA#/OVERLAY# AC1:CONDITIONAL FLAG AC2:CHANNEL NUMBER .TOVLD	CALL FOVLD (CHANNEL, OVERLAY, COND-FLAG, ERROR)
RELEASE OVERLAY AREA	RELEASE COMMAND ISSUED FROM OUTSIDE THE OVERLAY AREA		ACØ:AREA#/OVERLAY# .OVREL	CALL FOVRL (OVERLAYNAME, ERROR)
	RELEASE COMMAND ISSUED FROM WITHIN THE OVERLAY AREA	SPECIFY RETURN LOCATION OUTSIDE OVERLAY AREA	AC2:RETURN LOCATION ACØ:AREA#/OVERLAY# .OVEXIT	CALL OVEXT (OVERLAYNAME, RETURN LOCATION) ISSUE FROM INSIDE PROGRAM UNIT WITH OVERLAY STATEMENT
				CALL OVEXX (OVERLAYNAME, RETURN LOCATION)= ISSUE FROM OUTSIDE PROGRAM UNIT WITH OVERLAY STATEMENT
		KILL THE EXECUTING TASK	ACØ:AREA#/OVERLAY# .OVKILL SEE RDOS USER MANUAL PG 5-19.	CALL OVKILL (OVERLAYNAME) ISSUE FROM INSIDE PROGRAM UNIT WITH OVERLAY STATEMENT
				CALL OVKIX (OVERLAYNAME) ISSUE FROM OUTSIDE PROGRAM UNIT WITH OVERLAY STATEMENT

ORGANIZATION AND INITIALIZATION OF MULTITASK ENVIRONMENT

PROBLEM: To initialize program environment without forcing a core area to be "locked-up" once initialization is completed.

SOLUTION: Use overlay area for initialization procedure:

1. Default task initially does the following:

   -open channel to overlay file
   -load initializing overlay
   -call initializing subroutine

2. Initializing subroutine does the following:

   -open all channels program will need
   -activate necessary program tasks
      .pass channel numbers to tasks
      .pass user stack pointers to tasks

   -identify user defined devices
   -identify user defined power-up routine
   -identify user clock
   -return to default task

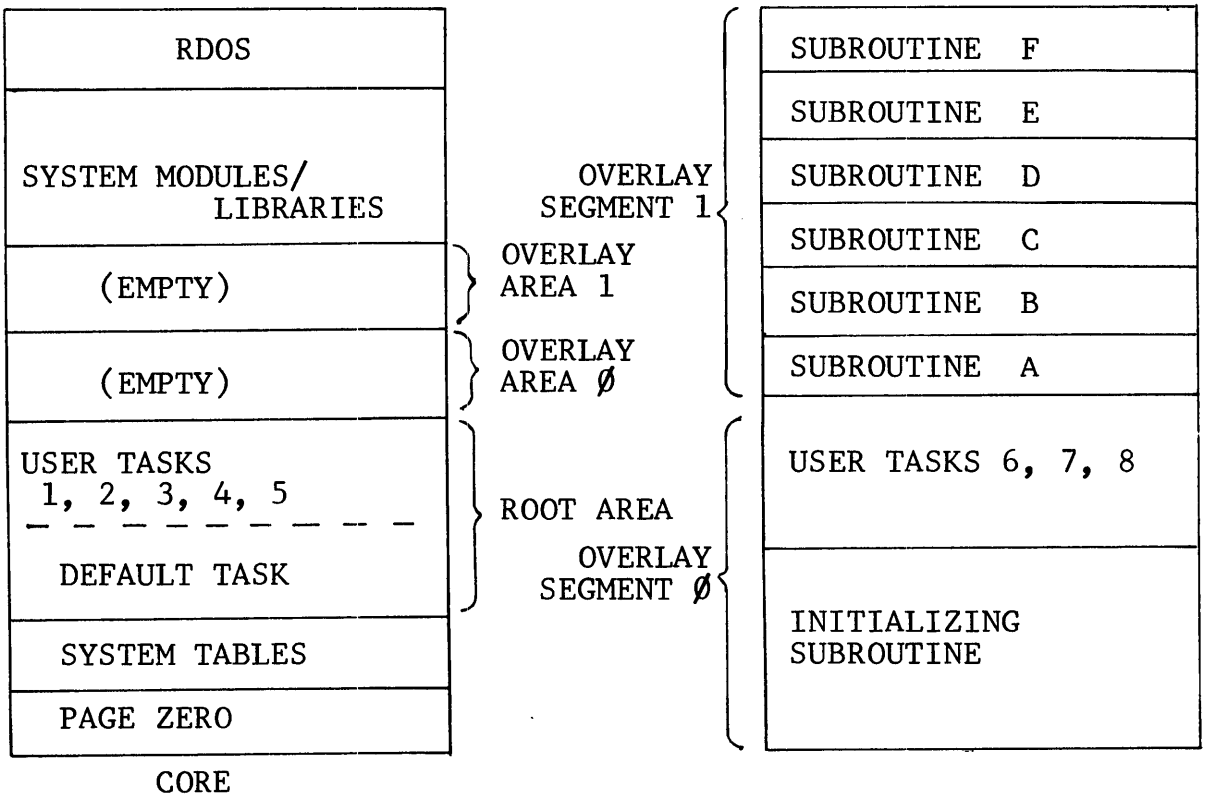
3. Default task then does the following:

   -release the overlay
   -load new overlay
   -self destructs (kill)

4. Highest priority ready task now gets CPU

PROGRAM INITIALIZATION

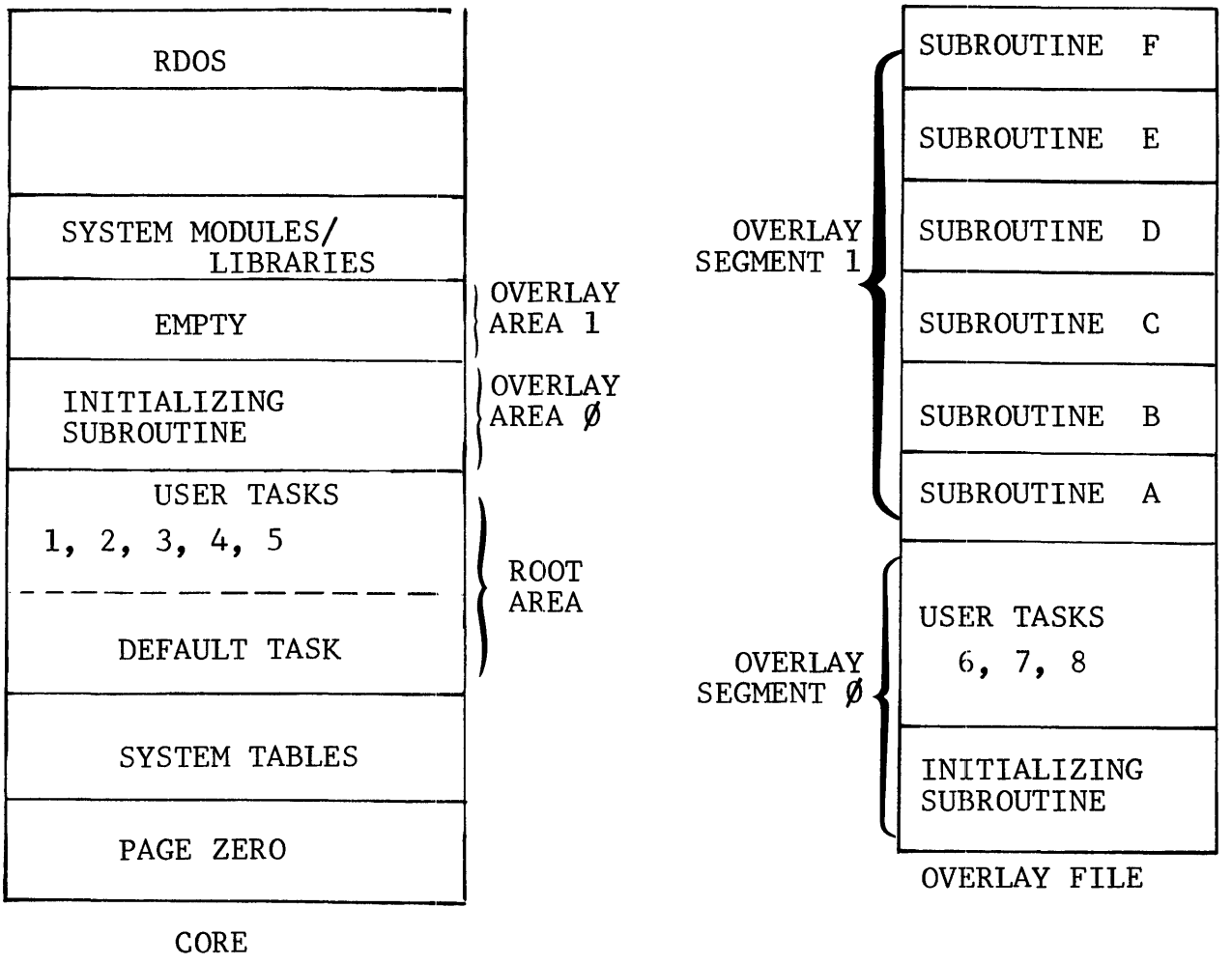
INITIAL CONFIGURATION



RLDR #/C #/K DEF 1 2 3 4 5[INIT,6 7 8] [A, B, C, D, E, F] LIBRARIES

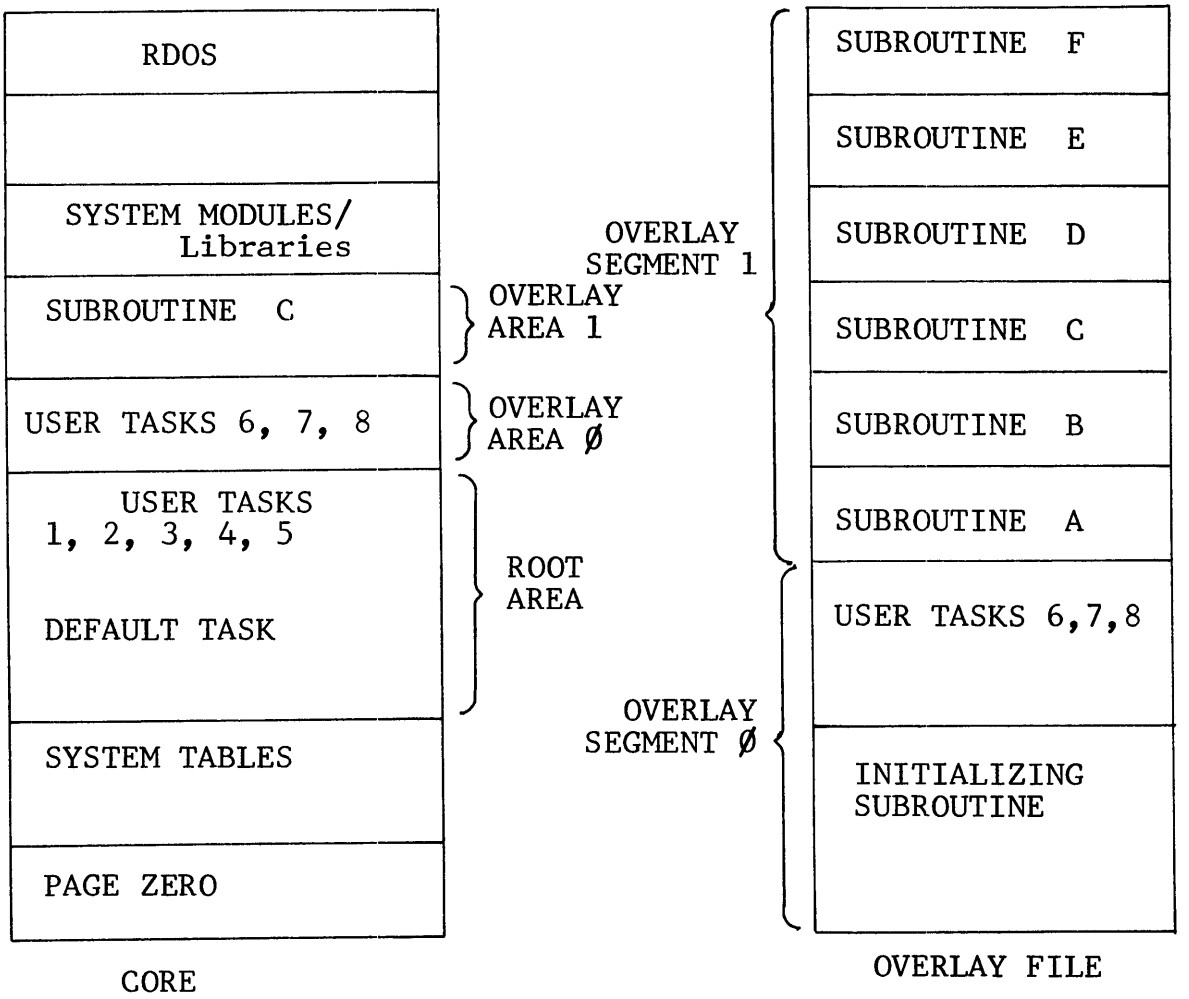
PROGRAM INITIALIZATION

CONFIGURATION DURING INITIALIZATION



PROGRAM INITIALIZATION

CONFIGURATION AFTER INITIALIZATION



RDOS COMMAND SUMMARY

MULTITASK TASK SYNCHRONIZATION

FUNCTION	CLI	ASM	FORT
READY A TASK AT EXPIRATION OF TIME DELAY	N/A	AC1: NUMBER OF RTC TICKS .SYSTEM .DELAY	CALL FDELY (NUMBER OF RTC TICKS) CALL WAIT (TIME, UNITS, ERROR) CALL START (ID, TIME, UNITS, ERROR)
QUEUE A TASK FOR ACTIVATION	N/A	AC2: ADDRESS OF USER TASK QUEUE TABLE .QTSK	CALL FQTASK (OVERLAYNAME, TASK, ARRAY, ERROR, TYPE

MULTI TASK SYNCHRONIZATION

QUEUE TASK TABLE

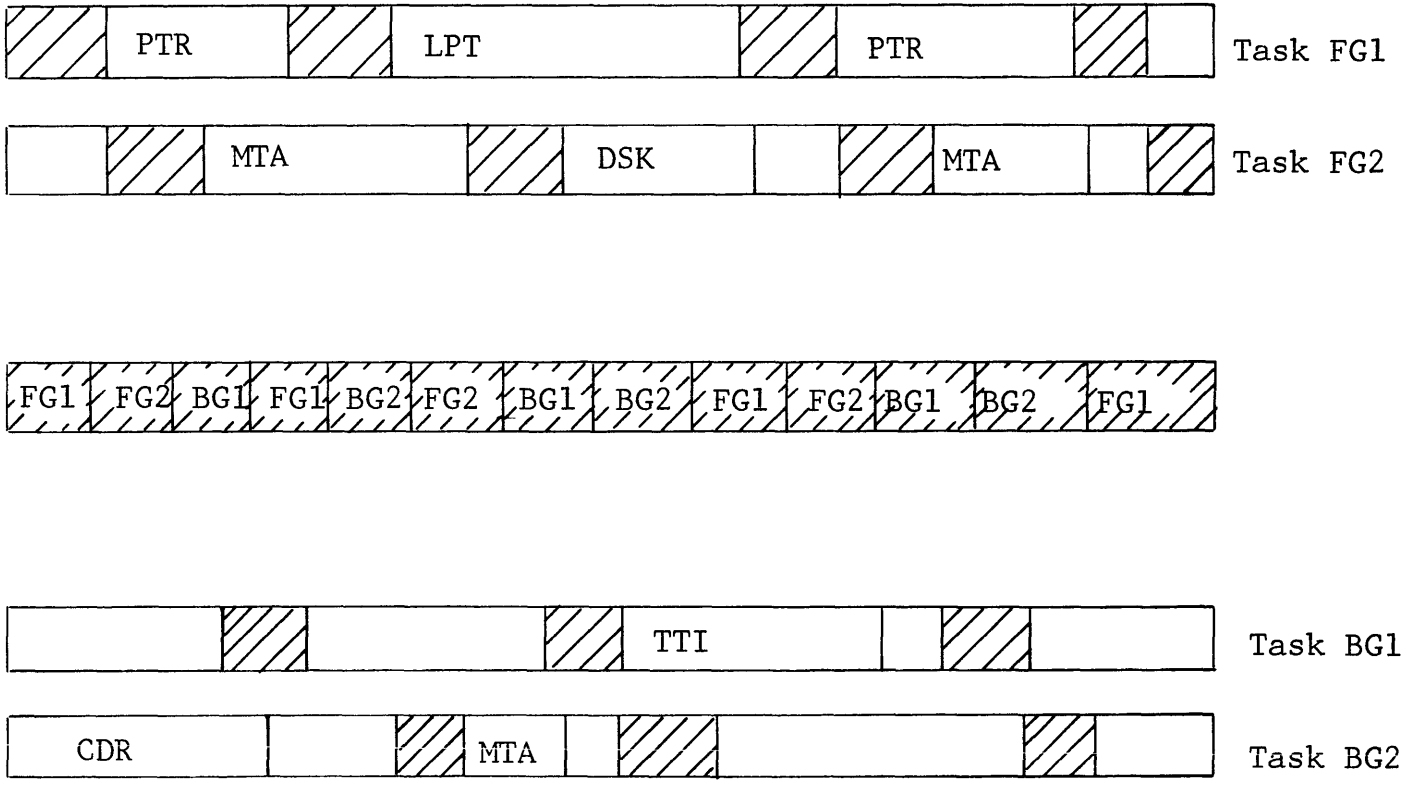
DISPLACEMENT	MEANING	MNEMONIC
0	Starting address of task	QPC
1	Number of times to execute task	QNUM
2	Node number/overlay number (-1 for core-resident tasks)	QTOV
3	Starting hour	QSH
4	Starting second in hour	QSMS
5	Task I.D./task priority	QPRI
6	Rerun time increments in seconds	QRR
7	System word	QTLNK
10	Overlay channel (unused by core resident tasks)	QOCH
11	Conditional/unconditional load flag (unused by core-resident tasks)	QCOND

Section VI

DUAL PROGRAMMING

DUAL PROGRAMMING

EFFICIENT USE OF CPU TIME



DUAL PROGRAMMING

GENERAL CHARACTERISTICS

Both programs are independent, separate entities

Programs' relative system priority can be set when FG program is executed

- FG greater priority than BG so FG will get CPU whenever FG needs it
- FG same priority as BG so FG & BG will share CPU

BG may inquire to see if FG program exists

FG may be interrupted by "Control F" at $TTI resulting in "FG INT" being typed at $TTO

FG may terminate itself by level zero FG program issueing program swap return resulting in "FG Term" being typed at $TTO

BG & FG may communicate

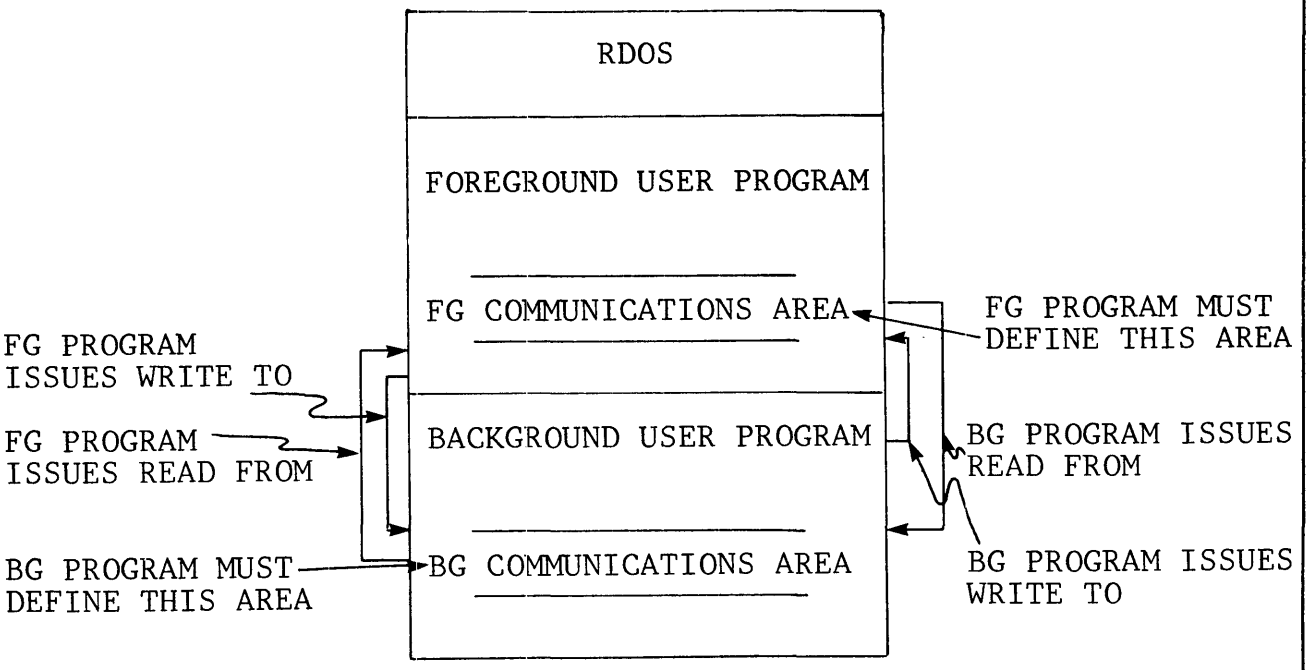
- Use common disc files
- Define program communications areas
- Use MCA for core to core data channel transfers

BG & FG cannot output to same device simultaneously

BG & FG should not input from same device simultaneously

DUAL PROGRAMMING

PROGRAM COMMUNICATIONS AREA



RDOS COMMAND SUMMARY

DUAL PROGRAMMING - PROGRAM COMMUNICATIONS

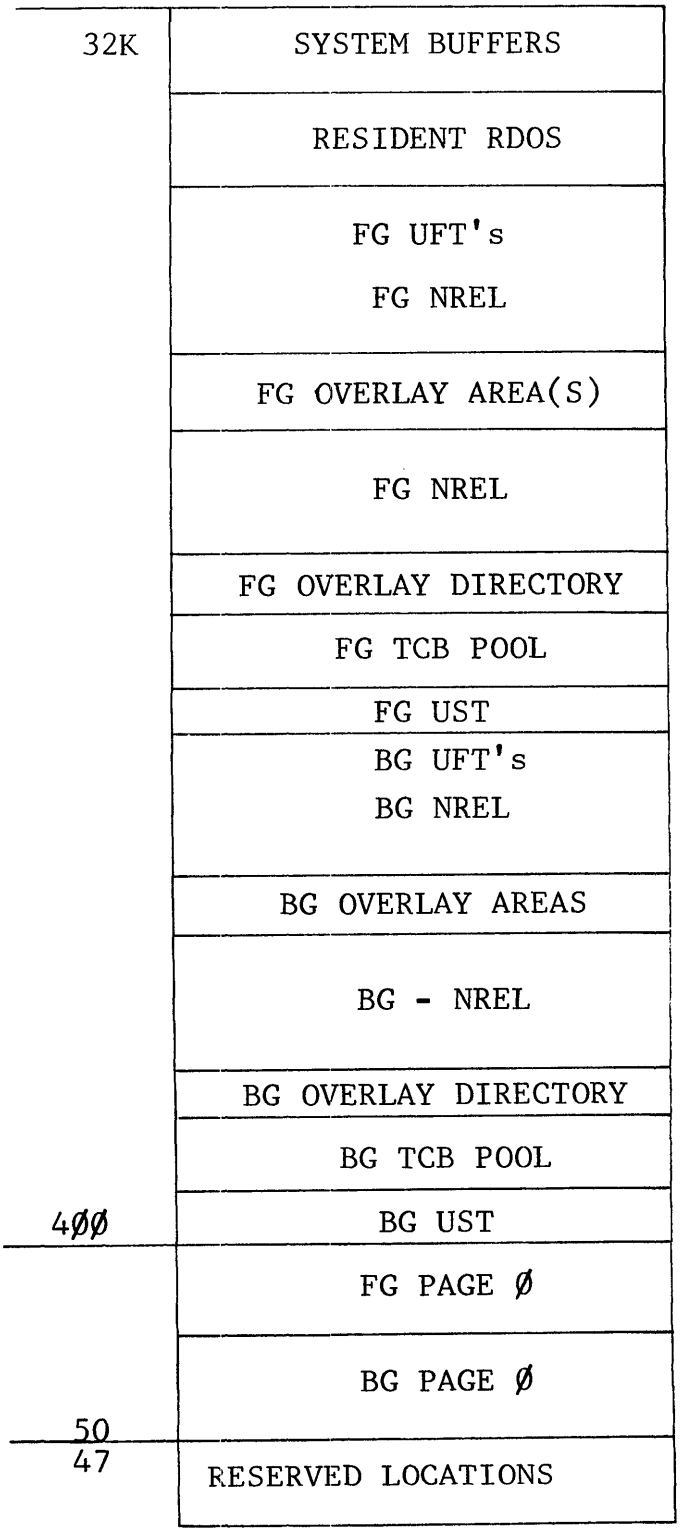
FUNCTION	CLI	ASM	FORT
DEFINE PROGRAM COMMUNICATIONS AREA	N/A	ACØ: START ADDRESS OF COMMUNICATIONS AREA AC1: WORD SIZE OF AREA .SYSTEM .ICMN	CALL ICMN (ARRAY, LENGTH, ERROR)
READ FROM OTHER PROGRAM'S COMMUNICATION AREA INTO ARBITRARY AREA IN THIS PROGRAM	N/A	ACØ: START ADDRESS OF THIS PROGRAM RECEIVING AREA AC1: OFFSET IN OTHER PROGRAM'S COMMUNICATION AREA AC2: WORD COUNT TO BE READ .SYSTEM .RDCMN	CALL RDCMN (ARRAY, START, NUMBER, ERROR)
WRITE INTO OTHER PROGRAM'S COMMUNICATION AREA FROM ARBITRARY AREA IN THIS PROGRAM	N/A	ACØ: START ADDRESS OF THIS PROGRAM'S SENDING AREA AC1: OFFSET IN OTHER PROGRAM'S COMMUNICATION AREA AC2: WORD COUNT TO BE WRITTEN .SYSTEM .WRCMN	CALL WRCMN (ARRAY, START NUMBER, ERROR)

DUAL PROGRAMMING

SPECIAL CHARACTERISTICS

UNMAPPED SYSTEMS
Software core program partitions
Single page zero shared by FG and BG
RDOS, FG & BG share up to 32K core
Only one core resident CLI possible
Swap/Chain in BG only
Save files must be specially loaded and can only run in ground loaded for
No checkpoint BG allowed

U N M A P P E D F G / B G



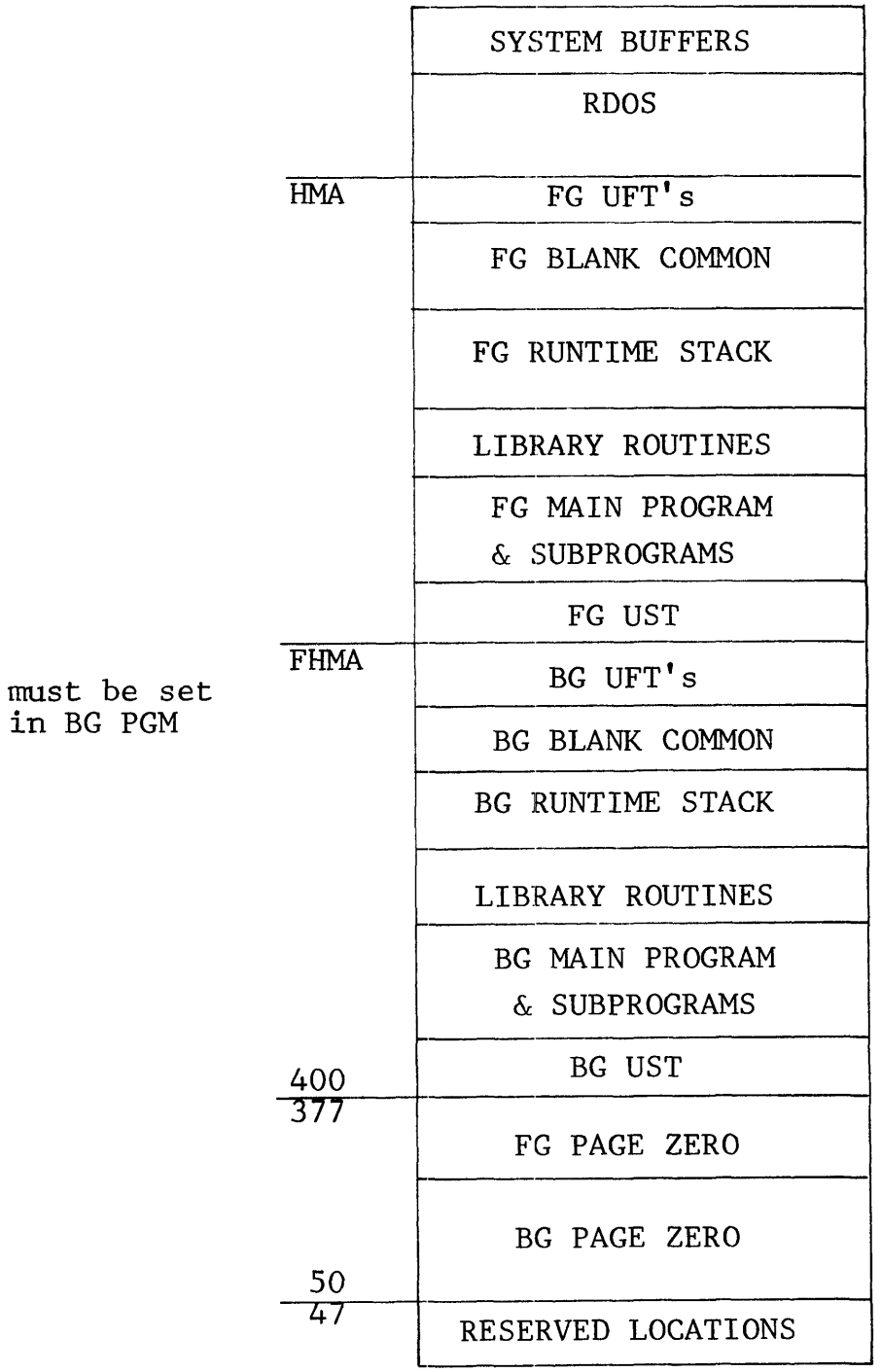
size of each ground is determined by the requirements of each program

*** A S S E M B L E R ***

All addresses shown are actual physical addresses

U N M A P P E D S Y S T E M

D U A L P R O G R A M E N V I R O N M E N T



*** F O R T R A N ***

DUAL PROGRAMMING

LOAD AND EXECUTE FOREGROUND PROGRAM

UNMAPPED SYSTEMS

Loading:

RLDR Octal #/F Octal #/Z Filenames, etc.

Octal #/F specifies FG starting address in NREL core

- Actual load address is M*$4ØØ_8$ + 16_8

Octal #/Z specifies FG starting address is ZREL core

- Actual load address is the octal #.

Execution:

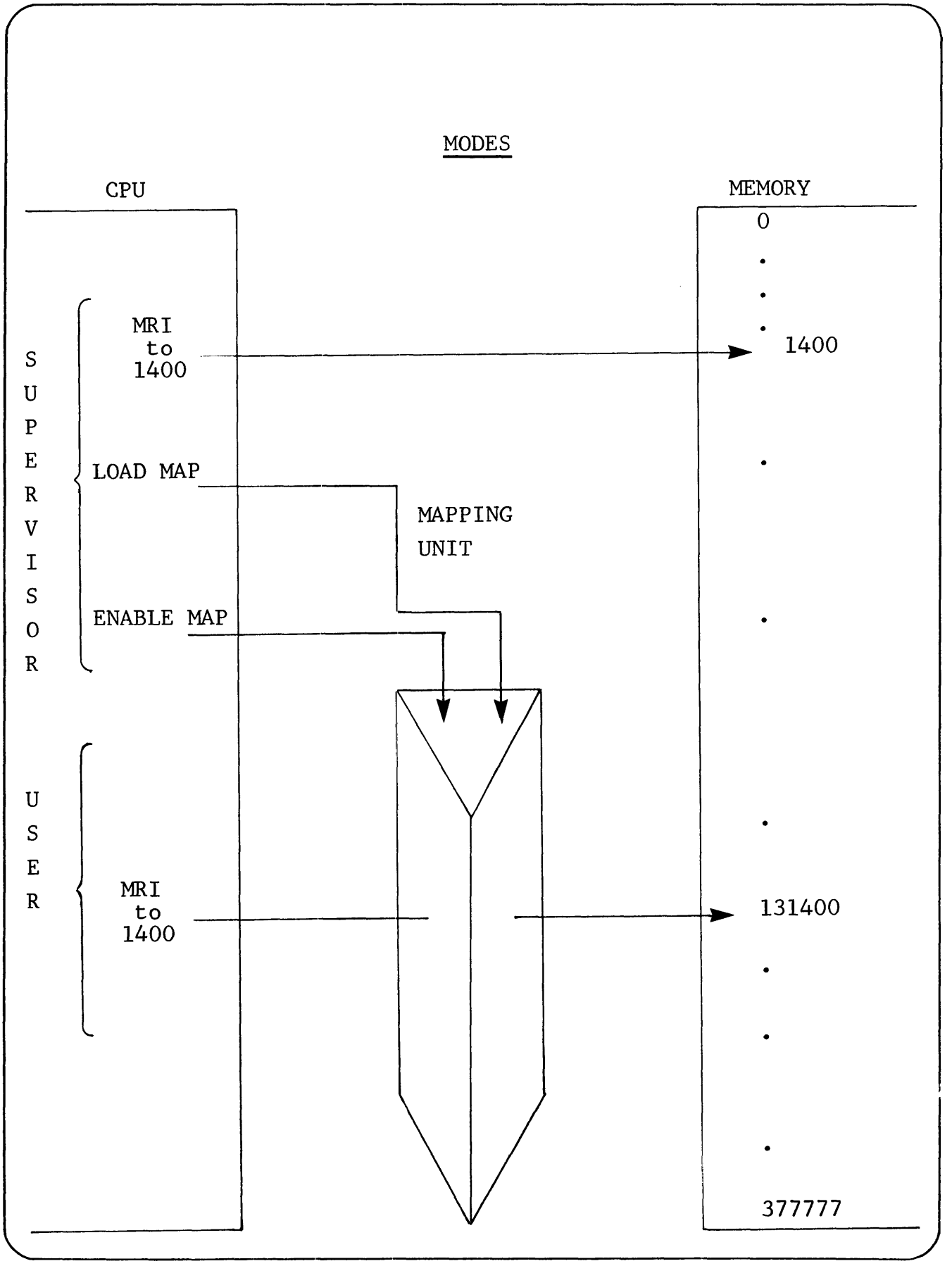
EXFG Filename FG higher priority than BG

EXFG/E Filename FG same priority as BG

RDOS COMMAND SUMMARY

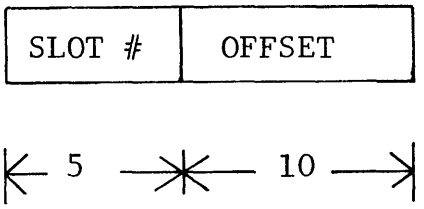
EXECUTION OF FOREGROUND PROGRAMS

FUNCTION	CLI	ASM	FORT
EXECUTE A FOREGROUND PROGRAM	EXFG NAME	ACØ = BYTE POINTER TO .SV FILENAME AC1 = Ø; FG BG AC1 = 4ØØØØ;FG = BG .SYSTEM .EXFG	CALL EXFG (NAME, PRI, ERROR) PRI = Ø; F BG PRI = 1; FG = BG
DETERMINE IF FOREGROUND PROGRAM EXISTS	N/A	.SYSTEM .FGND ACØ = Ø; NO FG ACØ = 1; YES FG	CALL FGND (IVAR) IVAR = Ø; NO FG IVAR = 1; YES FG

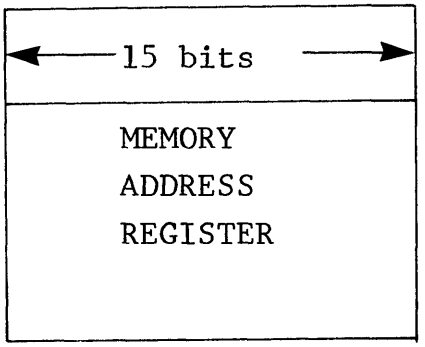


M A P P I N G

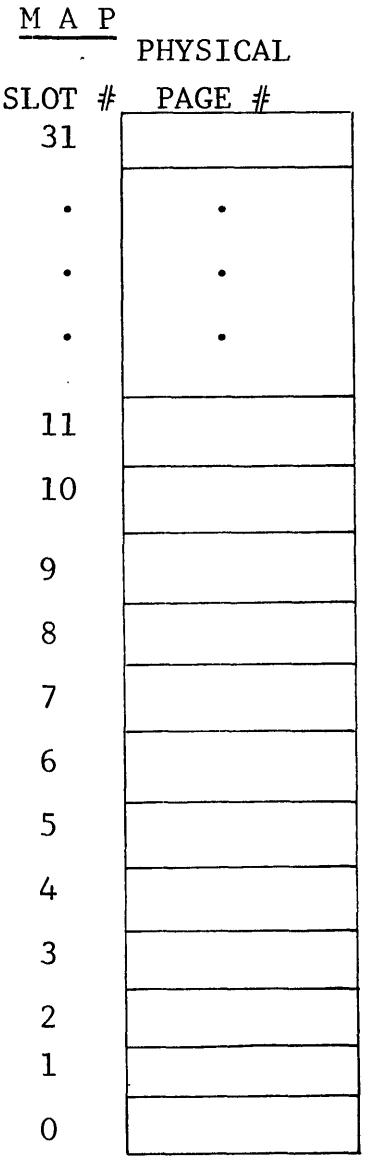
MAP CODE

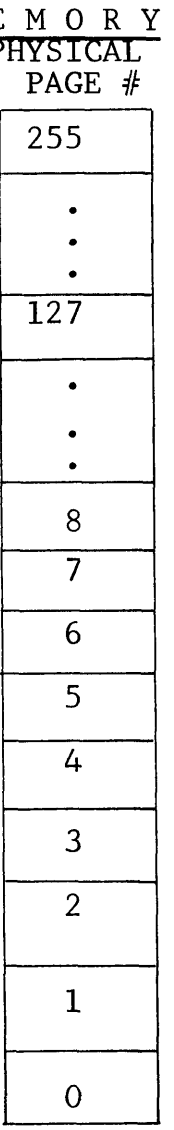


C P U



BITS	RANGE
5	32
7	128
8	256
10	1 K
15	32 K





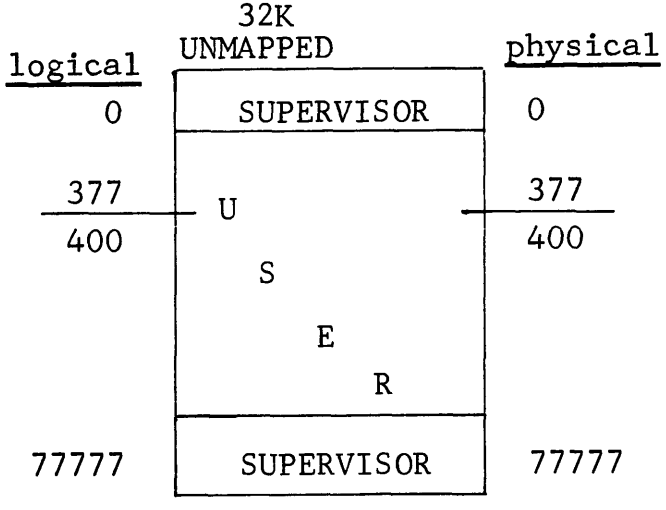
1. Replace leading 5 bits with contents of that map slot
2. Add offset

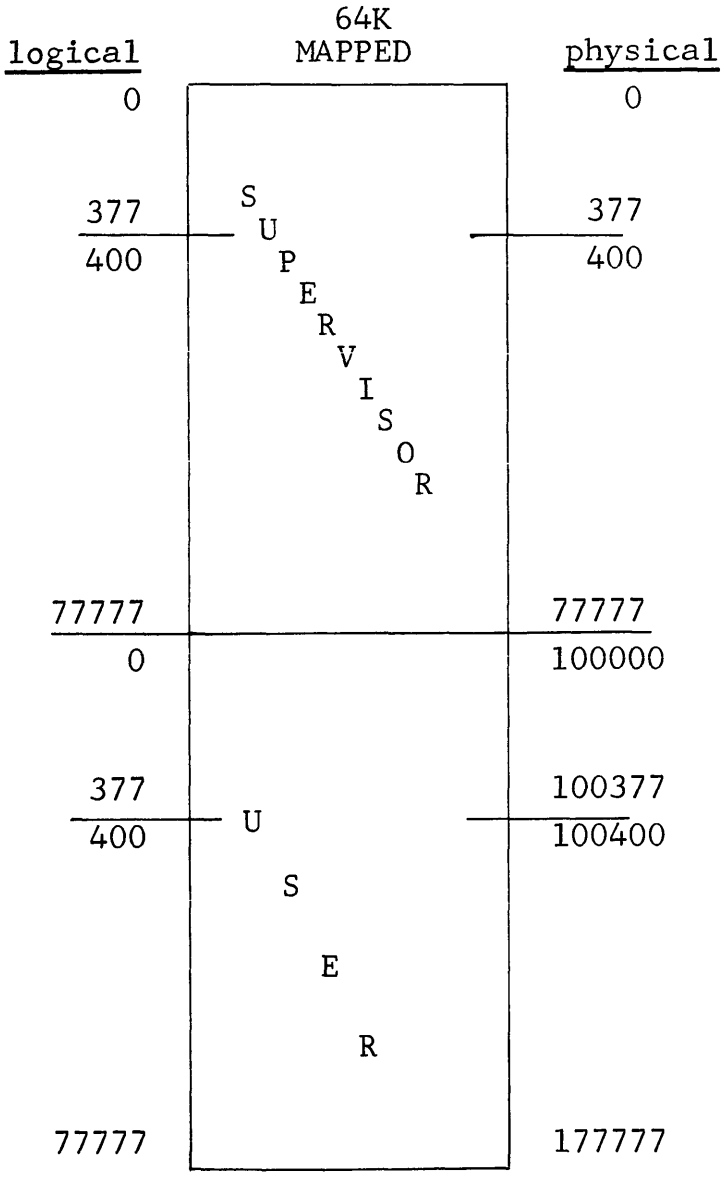
   if map slot 3 contained 70_8

   and the CPU accessed address 06022_8=000,11 0,000,010,010_2

   the physical location is 160022_8=1,110,00 0,000,010,010_2

CORE USAGE





CORE CONFIGURATION

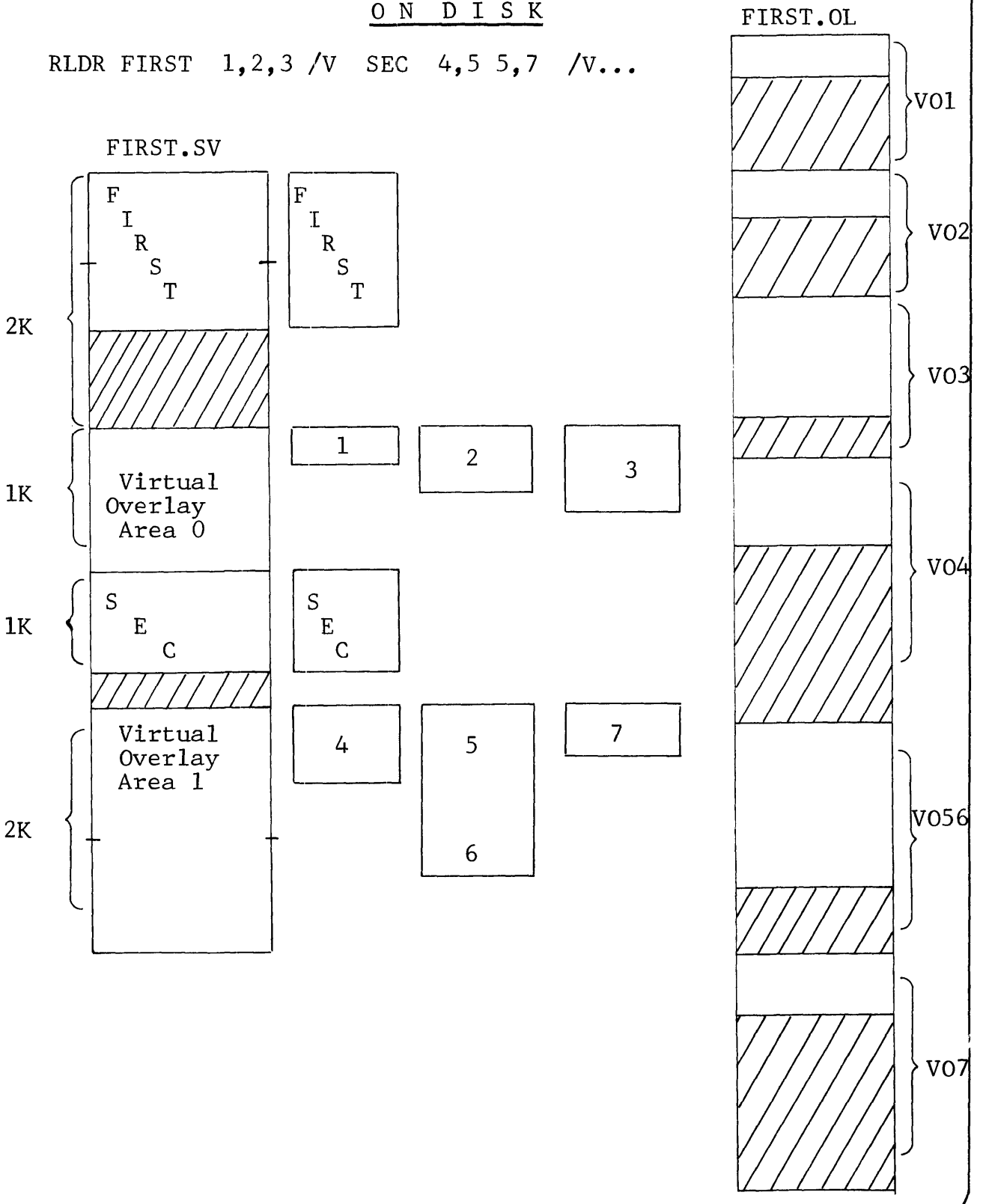
MAPPED - SINGLE PROGRAM ENVIRONMENT

HMA NMAX	
SST EST	SYMBOL TABLE
DEBUG	DEBUGGER
	TASK SCHEDULER
	TASK MODULES
INMAX	USER PROGRAM
	OVERLAY DIRECTORY
	TASK CONTROL BLOCK POOL
400_8	USER STATUS TABLE
0	USER PAGE ZERO
	USER FILE TABLES
	SYSTEM BUFFERS
	MRDOS
	MRDOS PAGE ZERO

VIRTUAL OVERLAYS

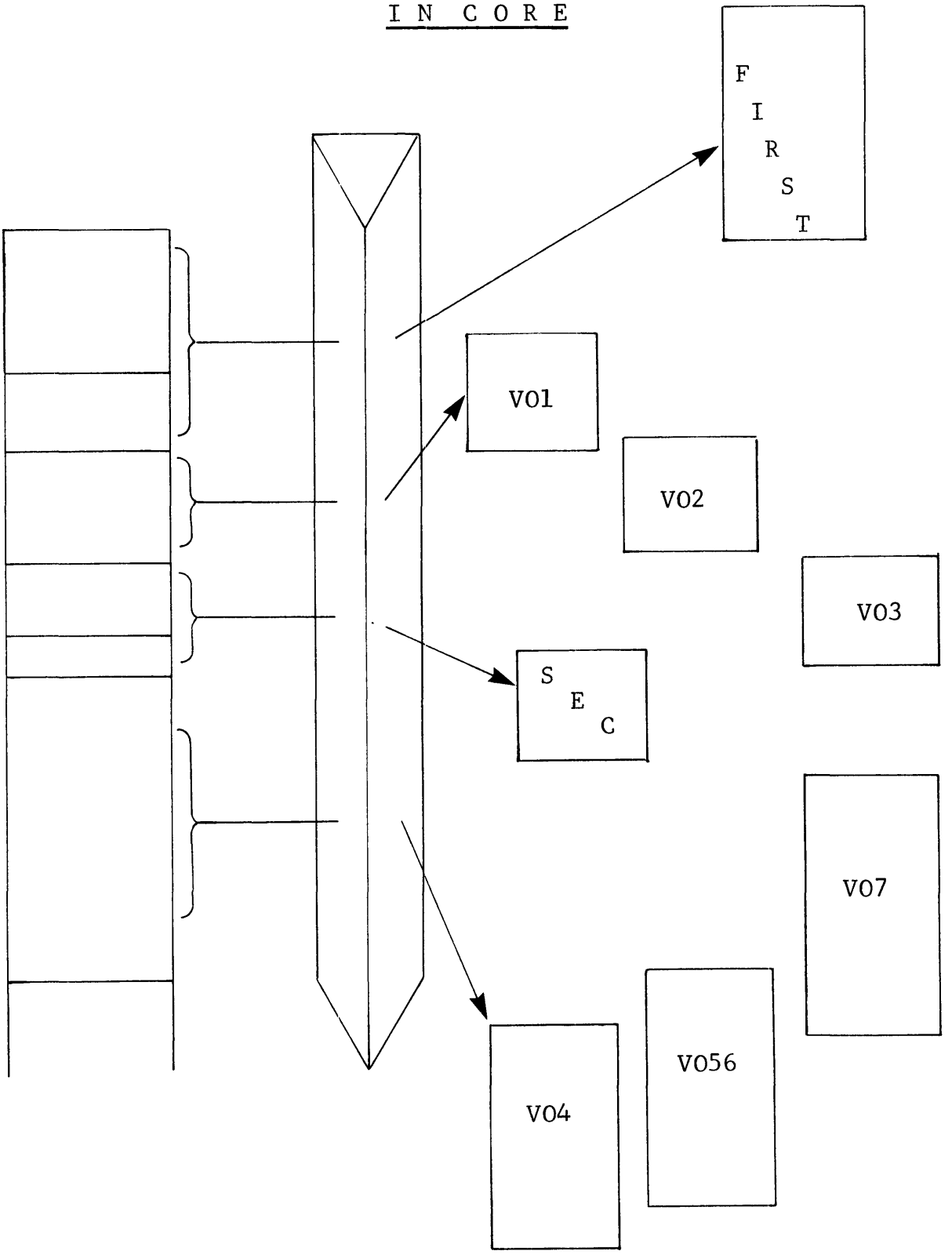
ON DISK

RLDR FIRST 1,2,3 /V SEC 4,5 5,7 /V...

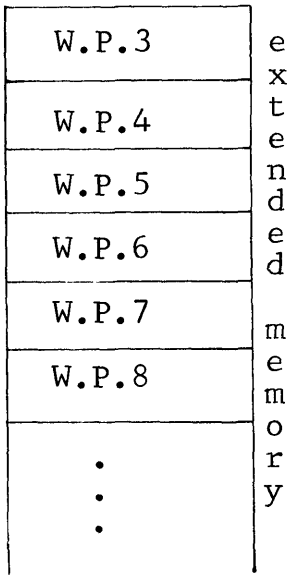


VIRTUAL OVERLAYS

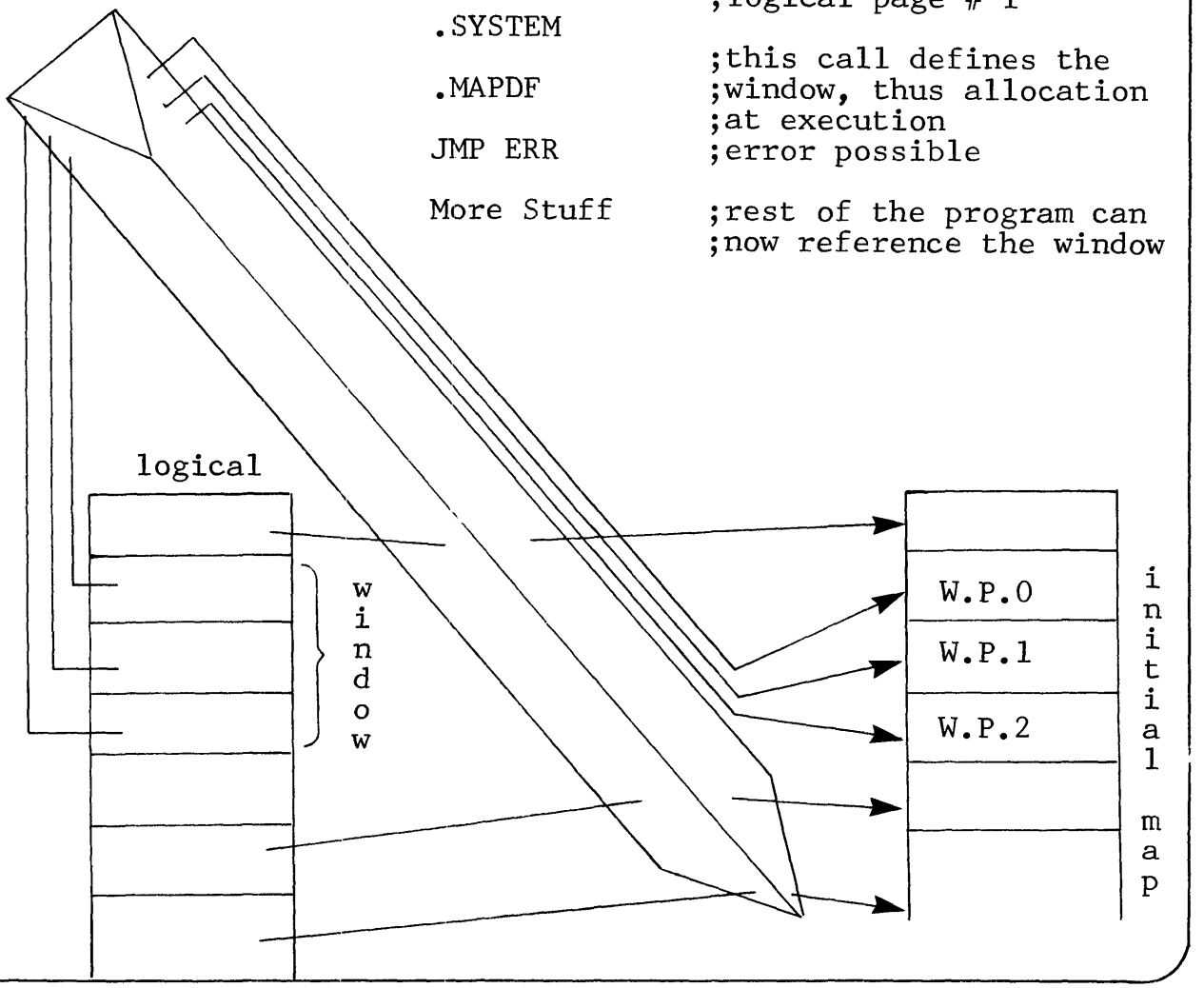
IN CORE



W I N D O W M A P P I N G

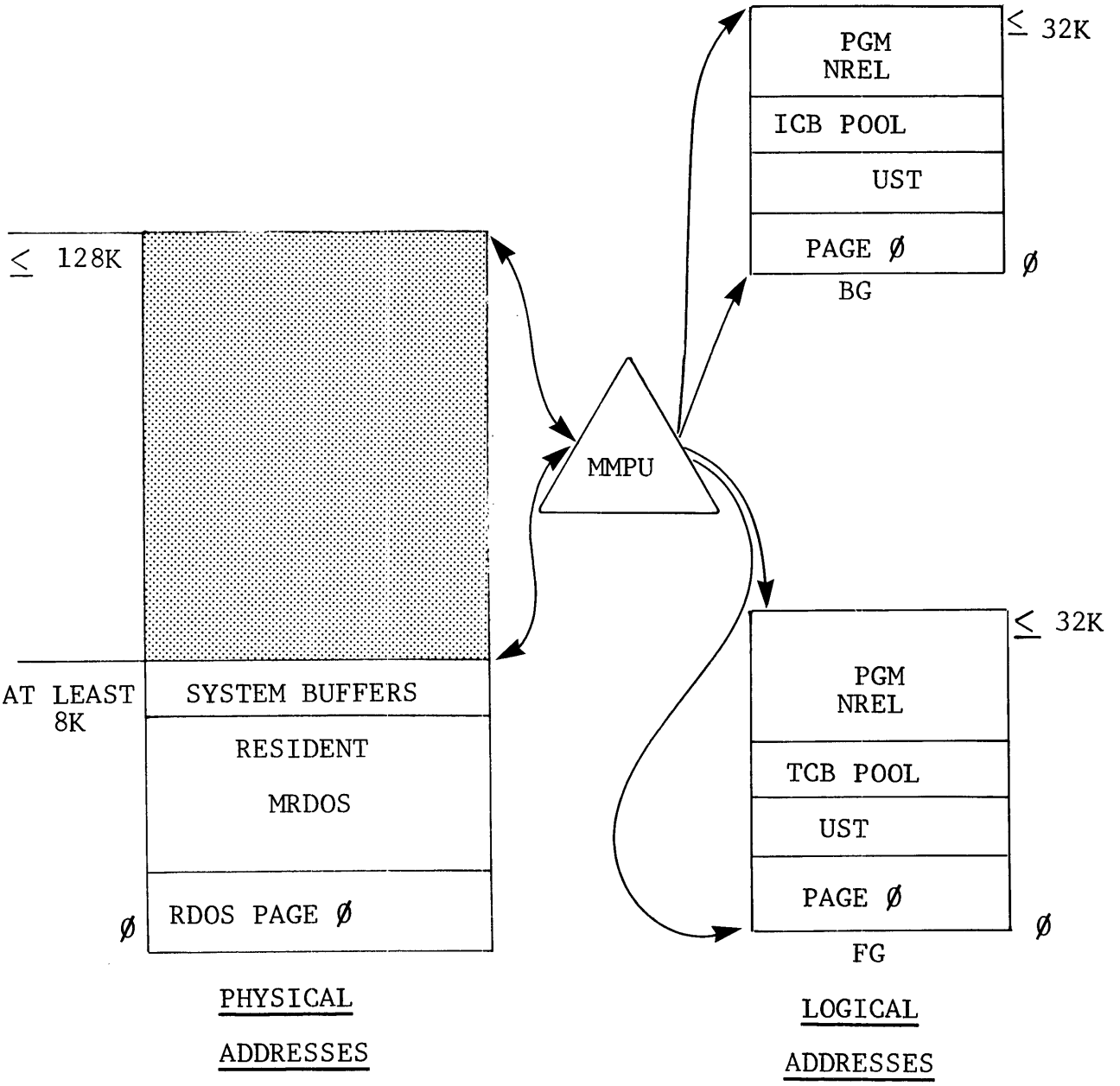



```
.SYSTEM
.VMEM
0                ;no error return
STA 0,FREE       ;# of extended memory
                 ;pages available
LDA 2,THREE      ;user sets window width
ADD 2,0          ;total # of pages affected
LDA 1,ONE        ;window will start at
                 ;logical page # 1
.SYSTEM
                 ;this call defines the
.MAPDF           ;window, thus allocation
                 ;at execution
JMP ERR          ;error possible
More Stuff       ;rest of the program can
                 ;now reference the window
```



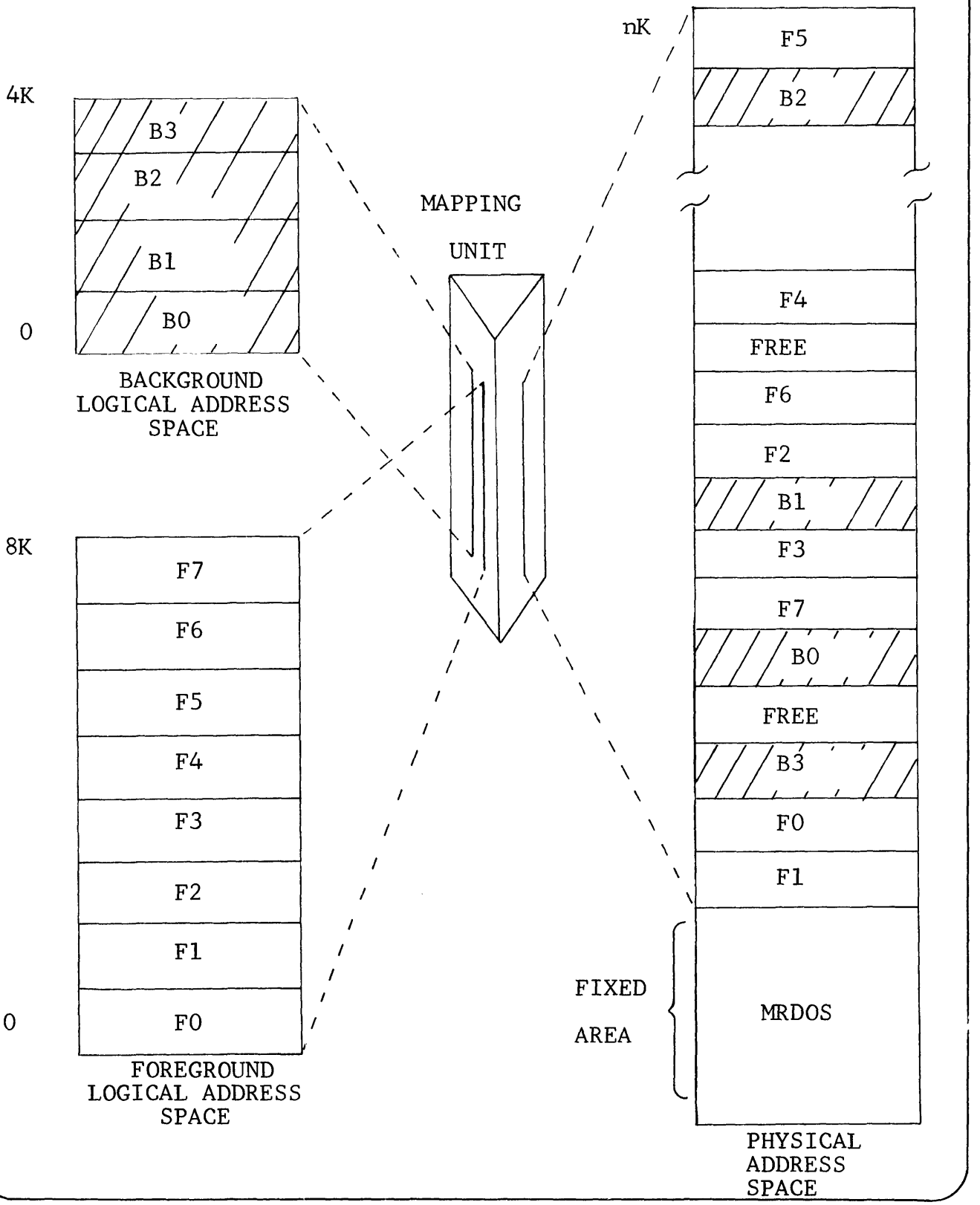
M A P P E D F G / B G

CHANGING FROM BG TO FG = CHANGING THE CONTENTS OF THE MAP SLOTS



LOGICAL TO PHYSICAL ADDRESS MAPPING

ADJACENT MAP SLOTS NEED NOT POINT TO ADJACENT PHYSICAL PAGES



DUAL PROGRAMMING

SPECIAL CHARACTERISTICS

MAPPED SYSTEMS

HARDWARE CORE PROGRAM PARTITIONS

SEPARATE PAGE ZERO'S FOR
EACH GROUND

MRDOS MAY HAVE UP TO 32k CORE
FG & BG EACH MAY HAVE UP TO 32k CORE

SEPARATE CLI FOR EACH BG & FG
POSSIBLE

BOTH BG & FG CAN SWAP/CHAIN

ALL SAVE FILES CAN RUN IN
BG & FG WITH NO MODIFICATION

SUPPORTS CHECKPOINT BG CAPABILITY

DUAL PROGRAMMING

LOAD AND EXECUTE FOREGROUND PROGRAM

MAPPED SYSTEMS

LOADING:

SAME AS BG PROGRAM IN MAPPED SYSTEM

EXECUTION:

EXFG CLI Begin execution of FG CLI (only if TTY1 exists)

EXFG Utility command string execute single utility in FG

EXFG Filename begin execution of FG program

RDOS COMMAND SUMMARY

PROGRAM CORE PARTITION COMMANDS

FUNCTION	CLI	ASM	FORT
DETERMINE CURRENT NMAX & HMA	MAPPED SYSTEMS ONLY GMEM	.SYSTEM .MEM ACØ - HMA AC1 - NMAX	N/A
CHANGE CURRENT NMAX	MAPPED SYSTEMS ONLY SMEM BG FG	ACØ - NMAX .SYSTEM .MEMI AC1 - NEW NMAX	N/A

CHECKPOINTING

GENERAL CHARACTERISTICS

Mapped systems only

Only one checkpoint BG program at a time

Definition: Suspend current background program (BG_{old}) at foreground's request, run new background program (BG_{new}) until completion, then restore BG_{old}

When checkpoint occurs .CP ENT typed at $TTO

When BG_{new} is interrupted by CTRLA or CTRLC from $TTI, .CP INT^{new} typed at $TTO and BG_{old} restored

FG may pass single word message to BG_{new}

For BG_{old} to be checkpointed it must not have any of the following calls outstanding at time checkpointing is initiated:

- QTY I/O Requests
- Time Delays
- Read Operator Messages
- User Defined Interrupt Servicing Routines
- User Defined Clock

This capability allows the separation of time critical real time program from the analysis of real time data while still allowing program development to proceed at low priority background activity.

RDOS COMMAND SUMMARY

DUAL PROGRAMMING - CHECKPOINTING

FUNCTION	CLI	ASM	FORT
EXECUTE A CHECKPOINT BG PROGRAM	N/A	ACØ = BYTE POINTER TO BG_{NEW} .SV FILENAME AC1 = Ø; BG_{NEW} = BG_{OLD} AC1 = 4ØØØØ; BG_{NEW} = FG .SYSTEM .EXBG	CALL EXBG (NAME, PRI,IER) PRI = Ø; BG_{NEW} = BG_{OLD} PRI = 1; BG_{NEW} = FG

DUAL PROGRAMMING SUMMARY

GROUND	SWAP	CHAIN	CHECKPOINT
BG	YES	YES	NO
FG	YES / NO	YES	YES / NO

MAPPED / UNMAPPED

Section VII

USER DEVICE INTERRUPT PROCESSING

INTERRUPT PROCESSING

TWO METHODS TO SERVICE USER DEVICES

- ADD DEVICE DRIVER DIRECTLY TO RDOS

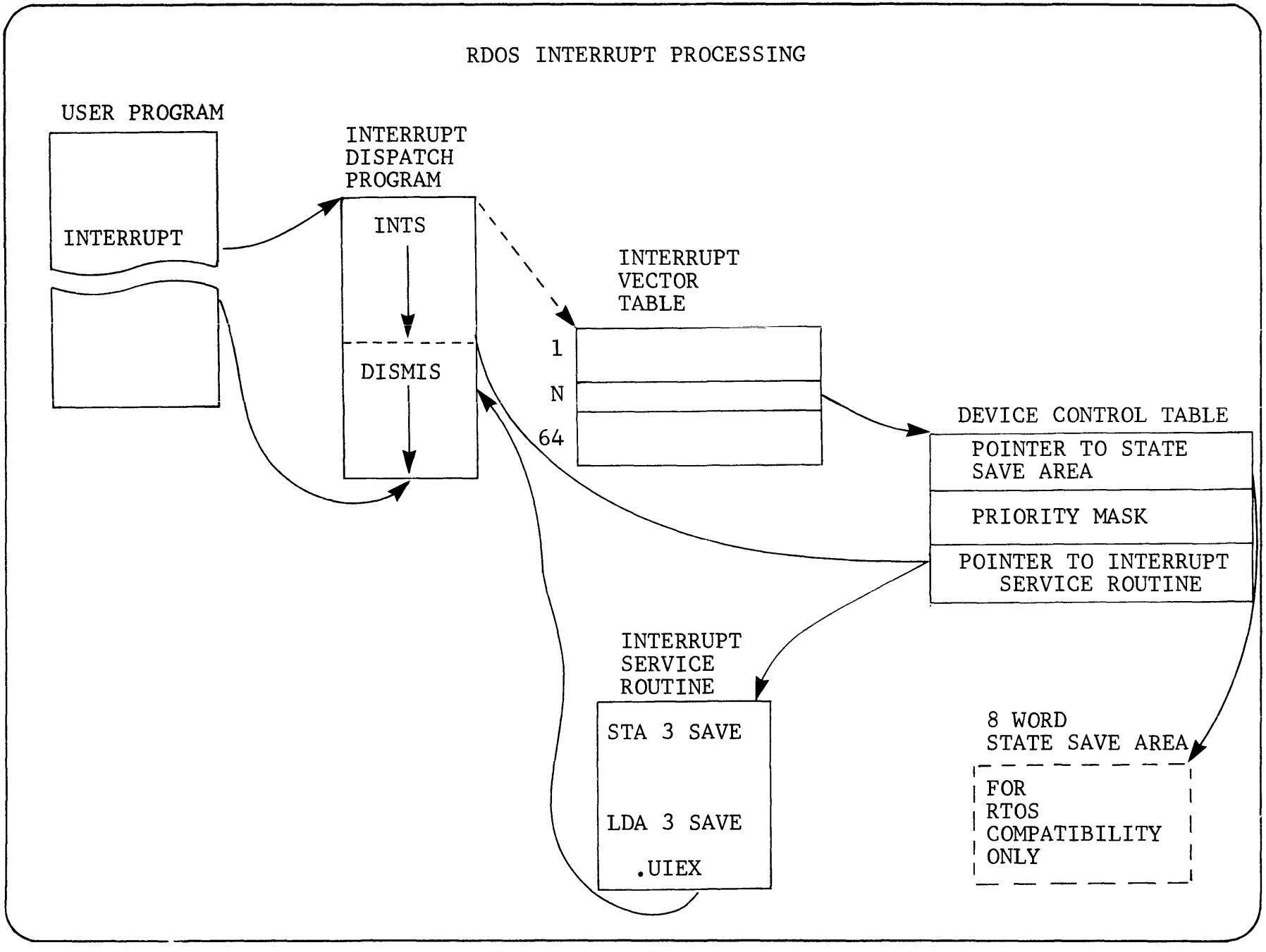
  ADVANTAGES:
  - DRIVER BECOMES PART OF RDOS
  - DRIVER CAN USE GENERAL RDOS SUBROUTINES
  - ACCESS TO DEVICE SAME AS OTHER RDOS DEVICES
  - DRIVER CAN USE GENERALIZED I/O ROUTINES

  DISADVANTAGES:
  - USER MUST HAVE RDOS SOURCE FILES
  - TO CHANGE DRIVER USER MUST REASSEMBLE/LOAD/SYSGEN

- DEFINE DEVICE DRIVER IN USER ADDRESS SPACE AT RUN TIME

  ADVANTAGES:
  - EASY TO IMPLEMENT!
  - SERVICED BY RDOS AS THOUGH A PART OF RDOS
  - EASY TO CHANGE DRIVERS
  - NO MODIFICATIONS NEEDED TO RDOS

  DISADVANTAGES:
  - USER MUST BE CAREFUL WITH PROGRAM SWAPS/CHAINS
  - NO GENERAL RDOS SUBROUTINES OR I/O ROUTINES AVAILABLE



USER INTERRUPTS AND MULTITASKING

WHEN A USER INTERRUPT IS DETECTED BY THE SYSTEM FROM EITHER A USER DEFINED CLOCK OR A USER DEVICE WHICH HAVE BEEN DEFINED, THE FOLLOWING ACTIVITIES TAKE PLACE:

1. THE MULTITASKING PROGRAM IS FROZEN, AND CONTROL ENTERS THE RDOS INTERRUPT DISPATCH ROUTINE. THIS ROUTINE WILL DETERMINE THE DEVICE CODE OF THE INTERRUPTING DEVICE AND USE THIS CODE AS A DISPLACEMENT IN THE INTERRUPT VECTOR TABLE TO OBTAIN THE ADDRESS OF THE DEVICE'S DCT (DEVICE CONTROL TABLE).
2. THE BIT PRIORITY MASK FOR THIS PARTICULAR DEVICE WILL THEN BE EXTRACTED FROM THE DCT, AND OR'ED WITH THE CURRENT SYSTEM PRIORITY MASK. THIS WILL PREVENT UNAUTHORIZED DEVICES FROM ISSUING INTERRUPTS WHILE THIS DEVICE IS BEING SERVICED.
3. THEN RDOS OBTAINS THE ADDRESS OF THE INTERRUPT SERVICE ROUTINE FOR THIS DEVICE FROM ITS DCT, AND TRANSFERS CONTROL TO THAT LOCATION. IF THIS INTERRUPT SERVICE ROUTINE IS FOR A USER CLOCK, WE CAN HAVE THE ROUTINE ITSELF PERFORM WHATEVER TIME-DEPENDENT FUNCTION IS REQUIRED AND THEN RETURN CONTROL TO THE TASKING WORLD (I.E., RESTORE THE MULTITASKING PROGRAM TO ITS PRE-INTERRUPT STATE), OR, WE CAN HAVE THIS USER CLOCK INTERRUPT SERVICE ROUTINE TRANSMIT A MESSAGE TO A WAITING TASK THROUGH THE .IXMT TASK CALL AND THEN RETURN CONTROL TO THE TASK SCHEDULER, WITH A REQUEST FOR RESCHEDULING. HOWEVER, IF THIS INTERRUPT SERVICE ROUTINE THAT HAS BEEN ENTERED IS AN INTERRUPT SERVICE ROUTINE FOR, SAY, A

USER-DEFINED INPUT DEVICE, WE CAN HAVE THE ROUTINE BRING IN THE DESIRED AMOUNT OF DATA (I.E., "SERVICE" THE DEVICE) AND THEN RETURN CONTROL EITHER TO THE TASKING PROGRAM OR TO THE TASK SCHEDULER WITH A REQUEST FOR RESCHEDULING (I.E., THE ISR WILL SERVICE THE DEVICE AND THEN TRANSMIT A MESSAGE TO A WAITING TASK).

4. WHEN THE INTERRUPT SERVICE ROUTINE IS COMPLETE, THE APPROPRIATE EXIT IS TAKEN (.UCEX OR .UIEX). CONTROL PASSES TO THE RDOS DISMISSAL ROUTINE, WHICH CHECKS THE VALUE IN AC1. IF THE VALUE IS ZERO, THEN CONTROL RETURNS TO THE TASKING WORLD (I.E., THE MULTITASKING PROGRAM IS RESTORED TO ITS PRE-INTERRUPT STATE). IF THE VALUE IN AC1 IS ANY NONZERO VALUE, THEN RESCHEDULING HAS BEEN REQUESTED AND CONTROL WILL BE PASSED BACK TO THE TASK SCHEDULER IN USER SPACE.

5. THE TASK SCHEDULER WILL REDUCE THE STATE OF THE PREVIOUSLY EXECUTING TASK TO READY (I.E., THE TASK THAT WAS EXECUTING WHEN THE INTERRUPT CAME IN) AND UPDATE ITS TCB. THEN THE SCHEDULER WILL READY THE TASK WHICH WAS WAITING FOR THE MESSAGE TRANSMITTED FROM THE INTERRUPT SERVICE ROUTINE AND THEN, FINALLY, CHOOSE THE HIGHEST PRIORITY READY TASK FOR THE EXECUTING TASK.

IN GENERAL, WHEN AN INTERRUPT SERVICE ROUTINE IS TO TRANSMIT A MESSAGE TO A WAITING TASK, THE BEST WAY TO DO SO IS BY SETTING UP THE MESSAGE LOCATION IN A LABELLED COMMON BLOCK. THIS CAN BE DONE IN THE INTERRUPT SERVICE ROUTINE THROUGH THE USE OF THE .COMM AND THE .GADD PSEUDO-OPS, (SEE EXAMPLE WITH THE USER CLOCK ROUTINE "JCLOCK"). THE TRANSMITTING OF THE MESSAGE IS ACCOMPLISHED BY THE .IXMT TASK CALL. IF .IXMT IS ISSUED, IT IS REQUIRED THAT RESCHEDULING BE REQUESTED WHEN EXITING FROM THE ROUTINE. IT IS ALSO IMPORTANT TO REMEMBER THAT THE .IXMT CALL DESTROYS ALL ACCUMULATORS. IT IS THEREFORE IMPORTANT TO REMEMBER TO PRESERVE THE RETURN ADDRESS TO THE TASKING WORLD (WHICH IS PASSED TO THE INTERRUPT SERVICE ROUTINE IN AC3) UPON ENTERING THE ROUTINE AND TO RESTORE THAT VALUE TO THE ACCUMULATOR BEFORE EXITING THE ROUTINE.

RDOS COMMAND SUMMARY

USER DEFINED INTERRUPTS COMMANDS

FUNCTION	CLI	ASM	FORT
IDENTIFY USER DEVICE	N/A	AC0: DEVICE CODE AC1: DCT ADDRESS .SYSTEM .IDEFINE	CALL FINTD (DEV-CODE, DET)
REMOVE USER DEVICE	N/A	AC0: DEV CODE .SYSTEM .IRMV	CALL FINRV (DEV-CODE)
EXIT FROM A USER DEVICE DRIVER	N/A	AC1: SCHED FLAG AC3: RETURN ADDRESS .UIEXIT	N/A
SEND A MESSAGE FROM A USER DEVICE DRIVER	N/A	AC0: MESSAGE ADDRESS AC1: MESSAGE .IXMT	N/A

Section VIII

POWER FAIL/ AUTO RESTART

USER DEFINED AUTO RESTART PROCESSING

GENERAL CHARACTERISTICS

- IF HARDWARE AVAILABLE - RDOS WILL ORDERLY SHUT DOWN WHEN POWER FAILS
- WHEN POWER IS RESTORED POWER-UP PROCEDURES ARE EXECUTED BY RDOS IF:
  - PANEL KEY IS IN LOCK POSITION
  - PANEL KEY IS NOT IN LOCK POSITION THEN PUT CPU SWITCHES ALL DOWN, DEPRESS START
- POWER-UP SERVICE PROVIDED FOR THE FOLLOWING DEVICES

  $PTP/$PTR
  $TTO/$TTI/$TTP/$TTR
  $CDR
  $LPT
  QTY
  DP n
  DK n

- TO PROVIDE POWER-UP SERVICE FOR USER DEVICES AND/OR SYSTEM DEVICES NOT IN THE LIST - IDENTIFY USER POWER-UP ROUTINE TO RDOS

RDOS COMMAND SUMMARY

USER DEFINED POWER-UP COMMANDS

FUNCTION	CLI	ASM	FORT
IDENTIFY USER DEFINED POWER-UP ROUTINE	N/A	ACØ: 77_8 AC1: START ADDRESS OF ROUTINE .SYSTEM .IDEFINE	CALL FINTD (63, NAME)
REMOVE USER DEFINED POWER-UP ROUTINE	N/A	ACØ: 77_8 .SYSTEM .IRMV	CALL FINRV (63)
EXIT FROM A USER DEFINED POWER-UP ROUTINE	N/A	AC3: RETURN ADDRESS .UPEXIT	N/A
SEND A MESSAGE FROM A USER DEFINED POWER-UP ROUTINE	N/A	ACØ: MESSAGE ADDRESS AC1: MESSAGE .IXMT	N/A

Section IX

USER CLOCK

MULTITASK TASK SYNCHRONIZATION

GENERAL CHARACTERISTICS

- ONE USER CLOCK MAY BE DEFINED BY A PROGRAM
- USER CLOCK DRIVEN BY REAL TIME CLOCK AND WILL THEREFORE BE A MULTIPLE OF THE SYSTEN'ED RTC FREQ.
- MINIMAL SYSTEM OVERHEAD IN SERVICING THE CLOCK
- USER SERVICE ROUTINE TREATED AS AN INTERRUPT SERVICE ROUTINE, I.E., TASK ENVIRONMENT FROZEN
- USER SERVICE ROUTINE MAY ISSUE INTERRUPT MESSAGE TRANSMIT COMMANDS
- USER MAY SUPPRESS OR FORCE TASK RESCHEDULING UPON EXIT FROM USER CLOCK SERVICE ROUTINE
- USER MAY INTERROGATE CURRENT RTC FREQ UNDER PROGRAM CONTROL

RDOS COMMAND SUMMARY

USER DEFINED CLOCK COMMANDS

FUNCTION	CLI	ASM	FORT
GET CURRENT RTC FREQUENCY	N/A	.SYSTEM .GHRZ ACØ: FREQ CODE	CALL GFREQ (IVAR)
DEFINE A USER CLOCK	N/A	ACØ: NUMBER OF RTC TICKS AC1: START ADDRESS OF SERVICE ROUTINE .SYSTEM .DUCLK	CALL DUCLK (RTC TICKS, NAME, ERROR)
REMOVE A USER CLOCK	N/A	.SYSTEM .RUCLK	CALL RUCLK
EXIT FROM A USER DEFINED CLOCK SERVICE ROUTINE	N/A	AC1: SCHEDULE FLAG AC3: RETURN ADDRESS .UCEXIT	N/A
SEND A MESSAGE FROM A USER DEFINED CLOCK SERVICE ROUTINE	N/A	ACØ: MESSAGE ADDRESS AC1: MESSAGE .IXMT	N/A

Section X

QTY

ASYNCHRONOUS MULTIPLEXOR: QTY

DGC PART NO. 4060

HARDWARE DESCRIPTION

- 4 LINES PER 15" PC CARD
- 1 to 16 CARDS PER CPU (4 to 64 QTY LINES)
- FULL OR HALF DUPLEX OPERATION
- TYPICALLY USED FOR 1200 or 2400 BAUD OPERATION

SOFTWARE DESCRIPTION

- RDOS QTY DRIVER IS A SYSGEN OPTION
- ACCESS TO A LINE IS BY CONVENTIONAL RDOS SOFTWARE CHANNEL ASSOCIATION WITH A LINE E.G. OPEN QTY:14 TO CHANNEL 3
- READ/WRITE ACCESS IS BY LINE MODE OR SEQUENTIAL BYTE MODE ONLY:

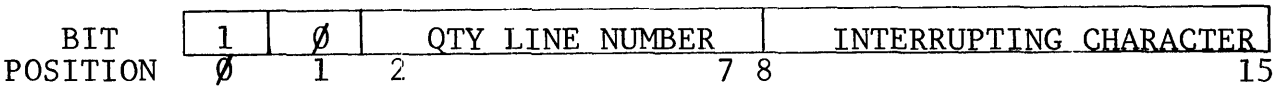
ACCESS MODE		ASM	FORTRAN
LINE	READ	.SYSTEM .RDL n	READ (CHANNEL#, FORMAT#) LIST
	WRITE	.SYSTEM .WRL n	WRITE (CHANNEL#, FORMAT#) LIST
SEQ BYTE	READ	.SYSTEM .RDS n	READ BINARY (CHANNEL#) LIST
	WRITE	.SYSTEM .WRS n	WRITE BINARY (CHANNEL#) LIST

ASYNCHRONOUS MULTIPLEXOR: QTY

LINE MONITOR CAPABILITY

. TO MONITOR ACTIVITY ON UNOPENED QTY LINES:

1. OPEN CHANNEL TO QTY: 64
2. ISSUE READ SEQUENTIAL FOR 2 BYTES
   - TASK THAT ISSUED READ WILL BE SUSPENDED
   - WHEN INTERRUPT OCCURS ON ANY UNOPENED QTY LINE TASK IS READIED -
     LINE NUMBER AND INTERRUPTING CHARACTER SENT TO TASK VIA AC2 IN ASSEMBLY LANGUAGE OR AN INTEGER VARIABLE IN FORTRAN:



3. MONITORING TASK CAN NOW ACTIVATE APPROPRIATE TASK AND ASSIGN CHANNEL NUMBERS AS DESIRED
4. REISSUE READ SEQUENTIAL AGAIN AND WAIT FOR NEXT INTERRUPT ON AN UNOPENED QTY LINE

Section XI

IPB

INTERPROCESSOR BUS: IPB

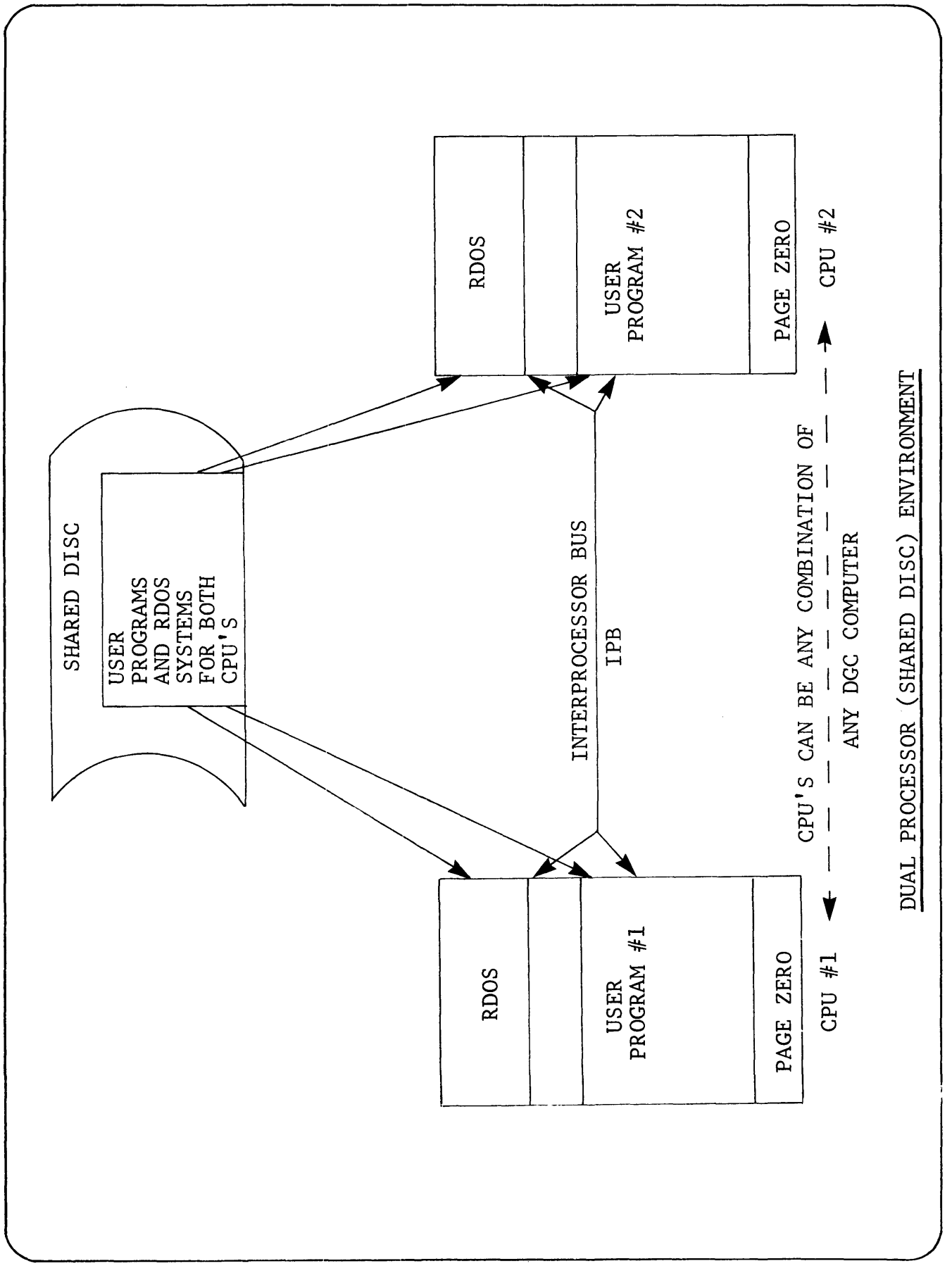
DGC PART NO.'S 4240 WITH 1065C CABLE

HARDWARE DESCRIPTION

- 2 INTERPROCESSOR BUFFERS; ONE PER CPU
- 1 CABLE; MAX 25 FOOT LENGTH
- FULL DUPLEX OPERATION
- INTERVAL TIMER

SOFTWARE DESCRIPTION

- THIS IS THE ONLY MEANS WHEREBY TWO OPERATING SYSTEMS IN SEPARATE CPU'S CAN MANAGE AND ACCESS SHARED DISC FILES WITH COMPLETE SYSTEM PROTECTION
- RDOS IPB DRIVER IS A SYSGEN OPTION
- ACCESS TO OTHER CPU IS BY CONVENTIONAL RDOS SOFTWARE CHANNEL ASSOCIATION WITH $DPO OR $DPI
- READ/WRITE ACCESS IS BY LINE MODE OR SEQUENTIAL BYTE MODE ONLY
- $DPO IS A SPOOLABLE DEVICE
- INTERVAL TIMER WILL INTERRUPT EITHER PROCESSOR IF THE OTHER PROCESSOR FAILS TO SERVICE ITS RTC WITHIN ONE SECOND



DUAL PROCESSOR (SHARED DISC) ENVIRONMENT

INTERPROCESSOR BUS: IPB

INTERVAL TIMER USAGE

- TIMER WILL GENERATE AN INTERRUPT IN ONE CPU IF THE OTHER CPU FAILS TO SERVICE ITS RTC WITHIN ONE SECOND
- USER MUST DEFINE AN INTERRUPT SERVICE ROUTINE FOR PROCESSING THE INTERVAL TIMER INTERRUPT WHOSE DEVICE CODE IS 37_8
- INTERVAL TIMER SERVICE ROUTINE CAN SEND A MESSAGE TO A WAITING RESTART TASK
  - THE RESTART TASK CAN ORDERLY SHUT DOWN THE FUNCTIONING CPU'S OPERATING SYSTEM
  - IT CAN THEN BOOTSTRAP A NEW OPERATING SYSTEM SIMILAR TO OR THE SAME AS THE SYSTEM FORMERLY OPERATING IN THE FAILED CPU
- THE NEWLY BOOTSTRAPPED OPERATING SYSTEM CAN THEN EXECUTE A RESTART PROGRAM THAT WILL BEGIN THE TERMINATED PROGRAMS OPERATION

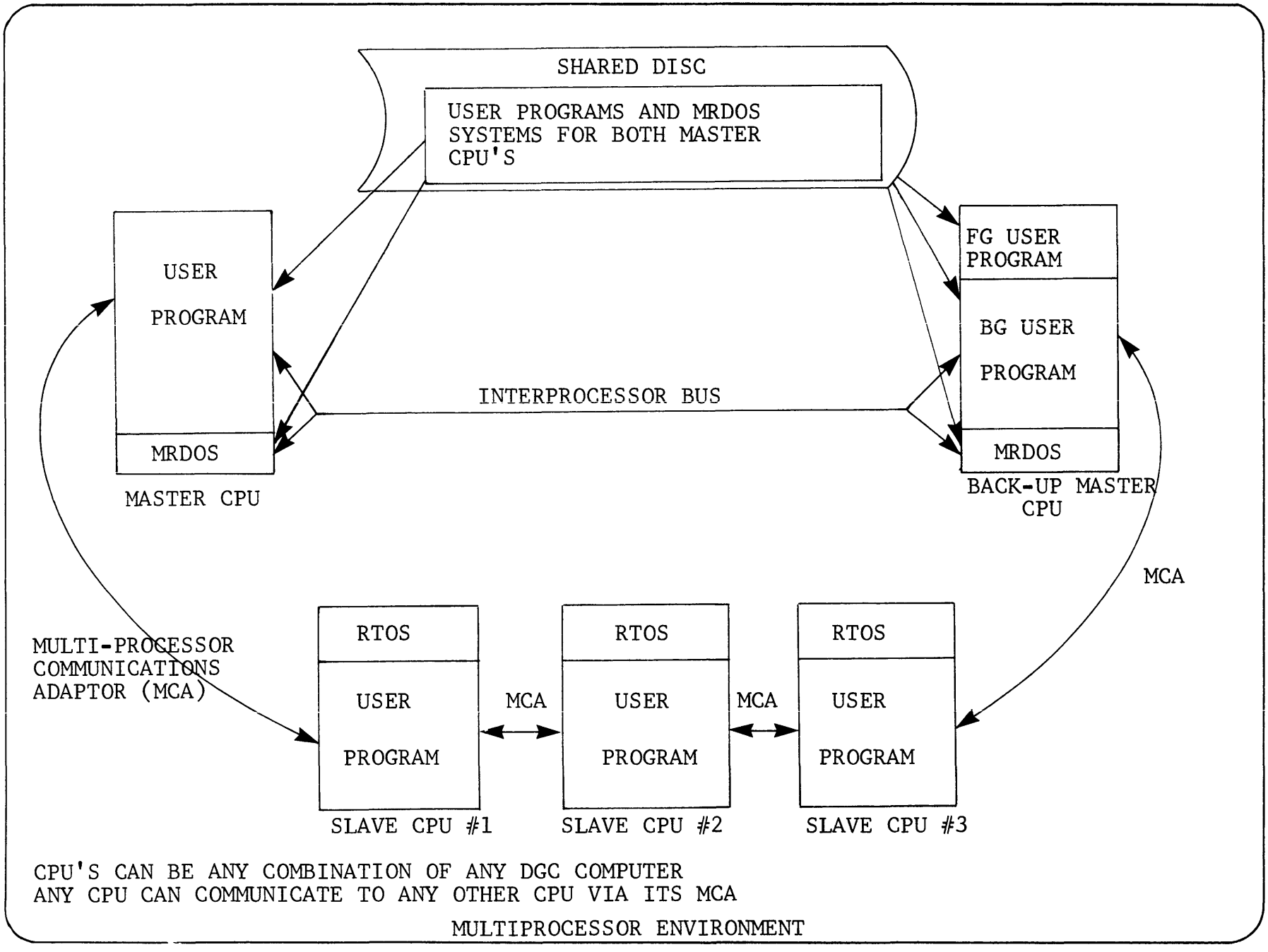
RDOS COMMAND SUMMARY

INTERPROCESSOR BUS/INTERVAL TIMER COMMANDS

FUNCTION	CLI	ASM	FORT
GET NAME OF CURRENT OPERATING SYSTEM	GSYS	ACØ: BYTEPOINTER TO RECEIVING AREA .SYSTEM .GSYS	CALL GSYS (ARRAY, ERROR)
BOOTSTRAP NEW OPERATING SYSTEM	BOOT PARTITIONNAME	ACØ: BYTEPOINTER TO PARTITION NAME .SYSTEM .BOOT	CALL BOOT (DEVICE, ERROR)

Section XII

MCA



MULTIPROCESSOR ENVIRONMENT

MULTIPROCESSOR COMMUNICATIONS ADAPTOR: MCA

DGC PART NO. 4038

HARDWARE DESCRIPTION

- MCA RECEIVER AND MCA TRANSMITTER PAIR MAKE UP UNIT
- TWO PAIRS POSSIBLE PER CPU
- UP TO 75' BETWEEN CPU'S
- FULL DUPLEX OPERATION
- DATA CHANNEL TRANSFER SPEEDS
- MCA RECEIVER CAN WAIT INDEFINITELY FOR DATA
- MCA TRANSMITTER TIMEOUT SETTABLE FROM 200 MILLISEC TO 655 SECONDS

SOFTWARE DESCRIPTION

- RDOS MCA DRIVER IS A SYSGEN OPTION
- ACCESS TO A CPU IS BY CONVENTIONAL RDOS SOFTWARE CHANNEL ASSOCIATION WITH A CPU'S UNIT NUMBER

  - 1st MCA CAN ACCESS 14 OTHER CPU'S BESIDES ITSELF

    MCAR: m m m m = 1 to 15
    MCAT: m m DEVICE CODE = 6_8

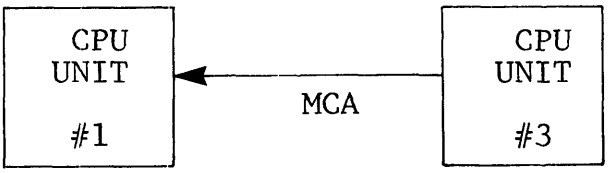
  - 2nd MCA CAN ACCESS 15 OTHER CPU'S BESIDES ITSELF

    MCAR1: n n n n = Ø to 15
    MCAT1: n n DEVICE CODE: 46_8

MULTIPROCESSOR COMMUNICATIONS ADAPTOR: MCA

SOFTWARE DESCRIPTION - READ/WRITE ACCESS

- READ/WRITE ACCESS BY SEQUENTIAL BYTE MODE ONLY
- DATA TRANSFERS UP TO 8192 BYTES PER TRANSFER
- READ ACCESS IS BY OPENING RDOS SOFTWARE CHANNEL TO MCA RECEIVER SPECIFYING THE OTHER CPU'S UNIT NUMBER, THEN ISSUING READ SEQUENTIAL BYTE SYSTEM CALL
- WRITE ACCESS IS BY OPENING RDOS, SOFTWARE CHANNEL TO MCA TRANSMITTER SPECIFYING THE OTHER CPU'S UNIT NUMBER, THEN ISSUING WRITE SEQUENTIAL BYTE SYSTEM CALL



OPEN TO MCAR:3
READ SEQUENTIAL

OPEN TO MCAT:1
WRITE SEQUENTIAL

- SINCE EACH PROGRAM IN EACH CPU IS ENTIRELY INDEPENDENT THERE IS NO IMPLIED RDOS SOFTWARE CHANNEL CORRESPONDENCE BETWEEN CPU'S
- GENERAL READ CAN BE ISSUED BY OPENING AND READING FROM MCAR:Ø WHICH INDICATES ANY CPU CAN TRANSMIT A MESSAGE TO IT

RDOS COMMAND SUMMARY

MCA COMMANDS

FUNCTION	CLI	ASM	FORT
GET CPU UNIT NUMBER FOR MCAT DEVICE	N/A	ACØ: 6 for MCAT 46_8 for MCAT1 .SYSTEM .GMCA AC1: UNIT NUMBER	N/A

A P P E N D I C E S

APPENDIX A

SAMPLE PROGRAMS

```
          .TITL S202LAB
          .ENT MON TELE INPUT LPTSK OUTSK
          .EXTD ERR ABNDC

          .NREL
MON:
          READS    2          ;
          COM#     2 2 SNR    ;SWR= -1?
          JMP      @=HOME     ;YES.
          MOVL#    2 2 SZC    ;BIT0= 0?
          MOV      2 1 SKP    ;GOT TWO CHARS.
          LDA      1 0 2      ;NO, GET TWO CHARS.
          JMP      MON        ;GO BACK TO SLEEP.
TELE:
          MOVZS    1 0        ;POSITION FOR PRINTING.
          JSR      TYPE       ;TYPE TWO BYTES.
          JSR      TYPE-1     ;OUTPUT CRLF.
          JMP      TELE       ;WAIT FOR NEXT MSG.
          LDA      0 CRLF     ;GET CARR. RET. LINE FEED.
TYPE:     STA      3 SAC3     ;SAVE RETURN.
          .SYST
          .PCHAR
          JSR      @ERR
          MOVCS    0 0 SZC    ;BOTH BYTES PRINTED?
          JMP      .-4        ;NO.
          JMP      @.+1       ;YES, RETURN.
SAC3:     0
CRLF:     12B7+15
INPUT:    LDA      1 =TABLE-1 ;GET ADDRESS OF TABLE.
          STA      1 20       ;INIT POINTER.
          SUBO     1 1        ;CLEAR AC1 & CARRY.
          .SYST
          .GCHAR              ;GET A CHAR.
          JSR      @ERR
          LDA      2 =33
          SUB#     0 2 SNR    ;CHAR = ESC?
          JMP      XXX        ;YES.
          LDA      2 ="=      ;
          SUB#     0 2 SNR    ;CHAR = EQUAL?
          JMP      YYY        ;YES.
          LDA      2 ="◊
          SUB#     0 2 SNR    ;CHAR = SHFT L?
          JMP      @=HOME     ;YES.
          ISZ      COUNT      ;MUST BE LEGAL.
          ADDCS    0 1 SZC    ;SAVE CHAR., GOT TWO?
          JMP      INPUT+3    ;NO, GET NEXT.
```

```
XXX:
YYY:    MOVCS   1 1 SZC ;REPOS. BYTES. ODD # CHARS?
        JMP     .+2     ;NO.
        JMP     .+5     ;YES.
        STA     1 @20   ;SAVE AS CHAR/CHAR.
        MOV     1 1 SZR ;CHAR/CHAR = NULLS?
        JMP     INPUT+2 ;NO, GET MOTE CHARS.
        JMP     .+3     ;YES.
        MOVS    1 1     ;REPOS ODD BYTE TO HI-BYTE.
        STA     1 @20   ;SAVE AS CHAR/NULL.
        LDA     1 COUNT ;PRESERVE COUNT FOR
        STA     1 TALLY ;OUTPUT & SLPT TASKS.
        SUB0    1 1     ;CLEAR COUNT FOR
        STA     1 COUNT ;NEXT RUN.
        LDA     2 =33   ;AC0= LAST CHAR.
        SUB#    0 2 SZR ;IS IT "ESC"?
        JMP     XOUT    ;NO, MSG FOR OUTPUT TASK.
XOUT:   JMP     INPUT   ;GO GET MORE MSGS.
COUNT:  0
TALLY:  0
TABLE:  .BLK 80.
HOME:   .SYST
        .RTN
        JSR     @ERR
LPTSK:
        MOV     1 2     ;PUT POINTER IN AC2.
        LDA     1 0 2   ;# OF BYTES TO WRITE.
        LDA     0 =2*TABLE;BYTE POINTER TO TABLE.
        .SYST
        .WRS    0       ;PRINT THE TABLE.
        JSR     @ERR
        JMP     LPTSK   ;WAIT FOR NEXT MSG.
OUTSK:
        MOV     1 2     ;PUT POINTER IN AC2.
        LDA     1 0 2   ;# TO CONVERT.
        JSR     @ABNDC  ;CONVERT TO DECIMAL.
        JSR     @.+2    ;OUTPUT CRLF.
        JMP     OUTSK   ;WAIT FOR NEXT MSG.
ATYPE:  TYPE-1

        .END    MON
```

```
        LICENSED MATERIAL - PROPERTY OF DATA GENERAL CORPORATION
 0001 BNDEC MACRO REV 02                          11:57:58 02/10/75
01
                              .TITL BNDEC
03                            .ENT BNDEC ABNDC
04                            .EXTD ERR
05
06                            .ZREL
07 00000-000000'ABNDC:        BNDEC              ;
08
09                            .NREL
10 00000'175100 BNDEC:        MOVL     3,3       ;
11 00001'054437               STA      3,SAC3    ;
12 00002'050437               STA      2,SAC3+1  ;
13 00003'044437               STA      1,SAC3+2  ;
14 00004'040437               STA      0,SAC3+3  ;
15 00005'034431               LDA      3,INST    ;
16 00006'054401               STA      3,.+1     ;
17 00007'000000 LOOP:         0                  ;
18 00010'020427               LDA      0,C60     ;
19 00011'146443               SUBO     2,1,SNC   ;
20 00012'101401               INC      0,0,SKP   ;
21 00013'147001               ADD      2,1,SKP   ;
22 00014'000775               JMP      .-3       ;
23 00015'006017               .SYST              ;
24 00016'010000               .PCHAR             ;
25 00017'006001$              JSR      @ERR      ;
26 00020'010767               ISZ      LOOP      ;
27 00021'151203               MOVR     2,2,SNC   ;
28 00022'000765               JMP      LOOP      ;
29 00023'020420               LDA      0,SAC3+3  ;
30 00024'024416               LDA      1,SAC3+2  ;
31 00025'030414               LDA      2,SAC3+1  ;
32 00026'034412               LDA      3,SAC3    ;
33 00027'175220               MOVZR    3,3       ;
34 00030'001400               JMP      0,3       ;
35
36       000012               .RDX 10
37 00031'023420 TENS:         10000
38 00032'001750               1000
39 00033'000144               100
40 00034'000012               10
41 00035'000001               1
42       000010               .RDX 8
43
44 00036'030422 INST:         LDA      2,.+TENS-LOOP
45 00037'000060 C60:          60
46 00040'000000 SAC3:         0
47 00041'000003               .BLK 3
48
49                            .END
```

```
        LICENSED MATERIAL - PROPERTY OF DATA GENERAL CORPORATION
 0001 ERROR MACRO REV 02                        18:41:13 02/10/75
01
02
                                .TITL ERROR     ;SYSTEM CALL ERROR RETURNS.
04                              .ENT ERR ERROR
05                              .EXTD APRNT ABNOC
06        000001                .TXTM 1
07
08                              .ZREL
09 00000-000000'ERR:    ERROR
10
11                              .NREL
12 00000'174400 ERROR:  NEG     3,3             ;
13 00001'174000         COM     3,3             ;DECREMENT ERROR PC BY 1.
14 00002'054420         STA     3,SAC3          ;SAVE IT.
15 00003'006002$        JSR     @APRNT          ;PRINT ERROR MESSAGE.
16 00004'000050"        MSG*2                   ;
17 00005'024415         LDA     1,SAC3          ;GET ERROR PC
18 00006'006001$        JSR     @ABNOC          ;AND PRINT IT.
19 00007'020414         LDA     0 CRLF
20 00010'006017         .SYST
21 00011'010000         .PCHAR
22 00012'006000-        JSR     @ERR
23 00013'101300         MOVS    0 0
24 00014'006017         .SYST
25 00015'010000         .PCHAR
26 00016'006000-        JSR     @ERR
27 00017'006017         .SYST                   ;
28 00020'006400         .ERTN                   ;GO TO SYS ERR ROUTINE.
29 00021'063077         HALT                    ;JUST IN CASE!
30 00022'000000 SAC3:   0                       ;
31 00023'005015 CRLF:   12B7+15
32 00024'042522 MSG:    .TXT /ERROR RETURN AT <7>/
33        051117
34        051040
35        051105
36        052125
37        051116
38        020101
39        052040
40        003400
41
42                      .END
```

```
        LICENSED MATERIAL - PROPERTY OF DATA GENERAL CORPORATION
 0001 PRINT MACRO REV 02                    10:15:35 02/12/75
01
02
03                ;THE FOLLOWING ROUTINE MAY BE USED TO OUTPUT
04                ;TEXT MESSAGES PACKED LEFT TO RIGHT USING
05                ;ASSEMBLER PSEUDO-OPS.  EXAMPLE:
06                ;                .TXTM 1
07                ;        AMSG:   .TXT /MESSAGE/
08                ;
09                ;MESSAGES PACKED BY .TXT WILL AUTOMATICALLY
10                ;END WITH A NULL BYTE.  WHEN "PRINT" RECIEVES
11                ;THE NULL IT WILL AUTOMATICALLY EXECUTE A
12                ;CARRIAGE RETURN (CR) AND LINE FEED (LF).
13                ;TO PREVENT THIS AUTOMATIC CR LF, THE MESSAGE
14                ;SHOULD END WITH THE BELL CHARACTER AS FOLLOWS:
15                ;                .TXTM 1
16                ;        AMSG:   .TXT /MESSAGE<7>/
17
18
19                ;THIS ROUTINE BEGINS BY SAVING THE STATE OF
20                ;THE MACHINE (ACCUMULATORS & CARRY) BEFORE
21                ;OUTPUTTING THE MESSAGE.  AT THE COMPLETION
22                ;OF THE MESSAGE THE ORIGINAL STATE, PRIOR
23                ;TO THE CALL, IS RESTORED AND THE OUTPUT
24                ;DEVICE IS IDLED.
25
26
27                ;THIS PROGRAM IS CALLED THRU ITS PAGE 0 LINK
28                ;AS FOLLOWS:
29                ;        JSR     @APRNT
30                ;        AMSG*2
31                ;        MORE PROGRAM
32                ;
33                ;THE WORD FOLLOWING THE CALL (AMSG*2) IS A
34                ;TRAILING ARGUMENT BYTE-POINTER TO THE
35                ;MESSAGE TO BE PRINTED.
36
37
38
```

```
        LICENSED MATERIAL - PROPERTY OF DATA GENERAL CORPORATION
!0002 PRINT
01
02
                                    .TITL PRINT
04                                  .ENT PRINT, APRNT
05                                  .EXTD ERR
06
07
08                                  .ZREL
09
10 00000-000000'            APRNT:PRINT
11
12
13                                  .NREL
14
15 00000'054452 PRINT:      STA     3,SAC3   ;SAVE AC3
16 00001'050450              STA     2,SAC2   ;SAVE AC2
17 00002'044446              STA     1,SAC1   ;SAVE AC1
18 00003'040444              STA     0,SAC0   ;SAVE AC0
19 00004'101100              MOVL    0,0      ;GET CARRY
20 00005'040446              STA     0,SCRY   ;SAVE CARRY
21 00006'010444              ISZ     SAC3     ;BUMP RETURN ADDRESS
22 00007'024412              LDA     1,BELL   ;GET ASCII BELL
23 00010'031400              LDA     2,0,3    ;GET MSG ADRS
24 00011'155220 MORE:        MOVZR   2,3      ;ADRS/2, BYTE PNTR TO CARRY
25 00012'021400              LDA     0,0,3    ;GET FIRST TWO CHAR.
26 00013'101003              MOV     0,0,SNC  ;WHICH BYTE?
27 00014'101300              MOVS    0,0      ;C=0, MOV HIGH TO LOW.
28 00015'034437              LDA     3,BMSK   ;GET LOW BYTE MASK
29 00016'163400              AND     3,0      ;MASK OUT BITS 0-8
30 00017'006017              .SYST            ; = JSR @17
31 00020'010000              .PCHAR           ;PRINT THE CHARACTER.
32 00021'000007 BELL:        7                ;ASCII BELL.
33 00022'105415              SUB#    0,1,SNR  ;WAS CHAR= BELL?
34 00023'000415              JMP     DONE     ;YES, DONE
35 00024'101005              MOV     0,0,SNR  ;NO, WAS IT NULL?
36 00025'000407              JMP     DONE-4   ;YES, DONE.
37 00026'151400              INC     2,2      ;NO, BUMP CHAR POINTER.
38 00027'000762              JMP     MORE     ;GO GET MORE MESSAGE.
39 00030'020416              LDA     0,CR     ;GET ASCII CARRIAGE RETURN.
40 00031'006017              .SYST
41 00032'010000              .PCHAR           ;PRINT CRLF.
42 00033'006001$             JSR     @ERR
43 00034'101300              MOVS    0 0      ;REPOS. FOR LINE FEED.
44 00035'006017              .SYST
45 00036'010000              .PCHAR
46 00037'006001$             JSR     @ERR
47 00040'020413 DONE:        LDA     0,SCRY   ;GET CARRY,
48 00041'101200              MOVR    0,0      ;RESTORE IT.
49 00042'020405              LDA     0,SAC0   ;RESTORE AC0
50 00043'024405              LDA     1,SAC1   ;RESTORE AC1
51 00044'030405              LDA     2,SAC2   ;RESTORE AC2
52 00045'002405              JMP     @SAC3    ;RETURN TO CALLING ROUTINE.
53 00046'005015 CR:          12B7+15
54 00047'000000 SAC0: 0                       ;
55 00050'000000 SAC1: 0                       ;TEMPORARY STORAGE FOR AC'S.
56 00051'000000 SAC2: 0                       ;
57 00052'000000 SAC3: 0                       ;
58 00053'000000 SCRY: 0                       ;
59 00054'000177 BMSK: 177                     ;MASK TO SAVE BITS 9-15.
60                          .END
```

```
        LICENSED MATERIAL - PROPERTY OF DATA GENERAL CORPORATION
 0001 BNOCT MACRO REV 02                        11:59:30 02/10/75
01
02
                                        .TITL BNOCT
04                                      .ENT ABNOC BNOCT
05
06                                      .ZREL
07 00000-000000'ABNOC:  BNOCT
08
09                                      .NREL
10 00000'054426 BNOCT:  STA     3,SAC3          ;
11 00001'050426         STA     2,SAC2          ;
12 00002'151100         MOVL    2,2             ;
13 00003'050425         STA     2,SCRY          ;
14 00004'152620         SUBZR   2,2             ;
15 00005'020420 LOOP:   LDA     0,C60           ;
16 00006'146443         SUBO    2,1,SNC         ;
17 00007'101401         INC     0,0,SKP         ;
18 00010'147001         ADD     2,1,SKP         ;
19 00011'000775         JMP     .-3             ;
20 00012'006017         .SYST                   ;
21 00013'010000         .PCHAR                  ;
22 00014'000401         JMP     .+1             ;
23 00015'151220         MOVZR   2,2             ;
24 00016'151220         MOVZR   2,2             ;
25 00017'151224         MOVZR   2,2,SZR         ;
26 00020'000765         JMP     LOOP            ;
27 00021'030407         LDA     2,SCRY          ;
28 00022'151200         MOVR    2,2             ;
29 00023'030404         LDA     2,SAC2          ;
30 00024'002402         JMP     @SAC3           ;
31 00025'000060 C60:    60                      ;
32 00026'000000 SAC3:   0                       ;
33 00027'000000 SAC2:   0                       ;
34 00030'000000 SCRY:   0                       ;
35
36                              .END
```

```
  0001   TOT
01                ;                    MULTITASKING EXAMPLE
02
03                              .TITL   TOT
04                              .ENT    TOT
05                              .EXTN   .PRI .TASK
06          000001              .TXTM   1
07
08                              .NREL
09  00000'020436 TOT:           LDA     0,.TTO          ;GET BYTE POINTER TO FILNAME.
10  00001'126400                SUB     1,1             ;DON'T INHIBIT CHARACTERISTICS.
11  00002'006017                .SYST
12  00003'014000                .OPEN   0               ;OPEN CHANNEL TO TTO
13  00004'004431                JSR     ERR
14  00005'020425                LDA     0,PRIOR         ;IO=0, PRI=10
15  00006'024425                LDA     1,NEWTASK       ;START ADDRESS = L
16  00007'152400                SUB     2,2             ;MSG DISPL=0
17  00010'177777                .TASK                   ;CREATE FIRST TASK
18  00011'004424                JSR     ERR
19  00012'151400                INC     2,2             ;MSG DISPL=1
20  00013'000010'               .TASK                   ;CREATE SECOND TASK
21  00014'004421                JSR     ERR
22  00015'151400                INC     2,2             ;MSG DISPL=2
23  00016'000013'               .TASK                   ;CREATE THIRD TASK
24  00017'004416                JSR     ERR
25  00020'177777                .PRI                    ;LOWER DEFAULT TO PRI=10*
26  00021'151400                INC     2,2             ;DISPL = 3
27  00022'034420 L:             LDA     3,.MESS         ;GET ADRS OF MSG POINTER TABLE.
28  00023'157000                ADD     2,3             ;ADD DISPL FOR CURRENT TASK.
29  00024'021400                LDA     0,0,3           ;GET MSG FOR " ".
30  00025'024407                LDA     1,COUNT         ;BYTE COUNT=8 for all MSGS.
31  00026'006017                .SYST
32  00027'016400                .WRS    0               ;OUTPUT THE MESSAGE TO $TTO
33  00030'004405                JSR     ERR
34  00031'000771                JMP     L               ;REPEAT FOR NEXT TASK
35
36  00032'000010 PRIOR:         10
37  00033'000022'NEWTASK:       L
38  00034'000010 COUNT:         8.
39
40  00035'000400 ERR:           JMP                     ;IF ERROR, HANG THERE
41  00036'000076".TTO:          .+1*2
42                              .TXT    /$TTO/
    00037'022124
    00040'052117
    00041'000000
43
44  00042'000043'.MESS:         .+1
45  00043'000116"               MESS0*2
46  00044'000130"               MESS1*2
47  00045'000142"               MESS2*2
48  00046'000154"               MESS3*2
49                  MESS0:      .TXT    /TASK 1<15><12>/
    00047'052101
    00050'051513
    00051'020061
    00052'006412
    00053'000000
50                  MESS1:      .TXT    /TASK 2<15><12>/
    00054'052101
```

* This moves .PRI task to end of queue and allows FIRST TASK to execute now.

The continuation of this program is:

Line #	Content
51	MESS2: .TXT /TASK 3<15><12>/
52	MESS3: .TXT /TASK 4<15><12>/
53	.END TOT

APPENDIX B

S203 PROBLEM II

MULTITASKING

Write a Fortran IV program consisting of at least 5 tasks to perform the functions described below. You may use more tasks as you see fit. (Hint: The best way to divide a program into tasks is by I/O requirements.):

Task 1 prints its name on the line printer every 10 seconds.

Task 2 prints its name on the line printer every 15 seconds.

Task 3 inputs integers from the $TTI, 10 at a time.

Task 4 sorts each group of 10 integers input by task 3.

Task 5 prints the sorted integer list on the line printer as soon as task 4 has them ordered.

MULTI-TASKING EXAMPLE #1

```
C       MAIN TASK = INITIALLIZING TASK
C
        EXTERNAL TSK5,TSK10,TSK15,TSK20,TSK30
C
        WRITE(12,5)
5       FORMAT(1H1,"MAINTASK STARTED",/)
        TYPE "MAIN TASK"
        ACCEPT "PRIORITIES ARE ",PRI5,PRI10,PRI15,PRI20,PRI30
        CALL ITASK(TSK5,1,PRI5,IER)
        CALL ITASK(TSK10,2,PRI10,IER)
        CALL ITASK(TSK15,3,PRI15,IER)
        CALL ITASK(TSK20,4,PRI20,IER)
        CALL ITASK(TSK30,5,PRI30,IER)
        IF(IER.EQ.1)GOTO 10
        WRITE(12) "ERROR IN MAIN TASK, IER = ",IER
10      WRITE(12) "MAIN DONE"
        CALL KILL
        END
```

MULTI-TASKING EXAMPLE #1 (continued)

```
C       5-SECOND TASK
C
        TASK TSK5
5       CALL WAIT(5,2,IER)
        IF(IER.EQ.1)GOTO 10
        WRITE(12) "ERROR IN TSK5, IER = ",IER
10      WRITE(12,15)
15      FORMAT(1H0,"5 SEC TASK")
        GOTO 5
        END
C       10-SECOND TASK
C
        TASK TSK10
5       CALL WAIT(10,2,IER)
        IF(IER.EQ.1)GOTO 10
        WRITE(12) "ERROR IN TSK10, IER = ",IER
10      WRITE(12) "10 SEC TASK"
        GOTO 5
        END
C       15-SECOND TASK
C
        TASK TSK15
5       CALL WAIT(15,2,IER)
        IF(IER.EQ.1)GOTO 10
        WRITE(12) "ERROR IN TSK15, IER = ",IER
10      WRITE(12) "15 SEC TASK"
        GOTO 5
        END
C       20-SECOND TASK
C
        TASK TSK20
5       CALL WAIT(20,2,IER)
        IF(IER.EQ.1)GOTO 10
        WRITE(12) "ERROR IN TSK20, IER = ",IER
10      WRITE(12) "20 SEC TASK"
        GOTO 5
        END

C       30-SECOND TASK
C
        TASK TSK30
5       CALL WAIT(30,2,IER)
        IF(IER.EQ.1)GOTO 10
        WRITE(12) "ERROR IN TSK30, IER = ",IER
10      WRITE(12) "30 SEC TASK"
        GOTO 5
        END
```

USER DEVICE DEFINITION

DEFAULT TASK:

```
        EXTERNAL WIDG,WIDGT
        .
        .
        .
        CALL FINTD(6Ø,WIDG)
        .
        .
        CALL ITASK(WIDGT,ID,NPRI,IER)
        .
        .
        .
        CALL KILL

        END
```

"WAITING" TASK:

```
        TASK WIDGT
        COMMON/INTMS/INTKEY
        COMMON/BUFFER/IRRAY(50)
        CALL REC(INTKEY,IDUM)
        TYPE "DATA IN FROM WIDG"
        GOTO 5
        END
```

"WIDG" INTERRUPT SERVICE ROUTINE:

```
        .TITLE    WIDG              .
        .EXTN     .IXMT,.UIEX       .
        .ENT      WIDG,WIDGS        . (SERVICE ROUTINE)
        .NREL                       SUBZL     1,1
        .COMM     INTMS,1           LDA       0,MADD
        .COMM     BUFFER,62         .IXMT
WIDG:   0                           SUBZL     1,1
        177777                      LDA       3,SAVE
        WIDGS                       .UIEXIT
WIDGS:  STA       3,SAVE    SAVE:   0
                            MADD:   .GADD     INTMS,0
                                    .END
```

MULTI-TASKING EXAMPLE #2

```
C    MULTITASKING DEMO
C
C
C    INITIALIZATION TASK FOR INITIALIZING THE MULTITASK ENVIRONMENT
C
C        "JSTART"
C
C    MASTER (DEFAULT) TASK HAS PRIORITY ZERO.  PERFORMS THE
C    INITIALIZATION OF THE MULTITASK ENVIRONMENT AND THEN PROMPTS
C    THE LOGGER TASK ONCE A MINUTE.
C
C    ASSUMES THE RTC IS SET TO 10 HZ.
C
C
     EXTERNAL JCLOCK,JSCAN,JALARM,JLOG
     COMMON/SWCH/IOLD(-1:15),INEW(-1:15),ICNT,TCNT
     COMMON/KEY/KEYSW,KEYA,KEYS
     INTEGER TCNT
     TYPE "THIS IS A FORTRAN IV MULTITASKING DEMO"
     CALL ITASK(JSCAN,10,1,IER)          ;ID=10,PRI=1
     CALL ITASK(JALARM,20,2,IER)         ;ID=20,PRI=2
     CALL ITASK(JLOG,30,3,IER)           ;ID=30,PRI=3
     CALL HOLD(30,IER)                   ;SUSPEND THE LOGGER TASK
     CALL DUCLK(10,JCLOCK,IER)           ;DEFINE USER CLOCK
     IF(IER-1)99,1,99
1    CALL WAIT(1,3,IER)                  ;TAKE A ONE MINUTE NAP
     CALL RELSE(30,IER)                  ;WAKE UP THE LOGGER TASK
     IF(IER-1)2,1,2
2    CALL JERROR(30,IER)
     GOTO 1
99   TYPE "DEMO ABORTED"
     DO 3 I=10,30,10
3    CALL ABORT(I,IER)                   ;DEACTIVATE THE TASKS
     CALL RUCLK                          ;REMOVE THE USER CLOCK
     CALL EXIT
     END

*****************************************************************


          SUBROUTINE JERROR(I,J)
          TYPE "TASK ID #",I," ERROR CODE=",J
          RETURN
          END
```

MULTI-TASKING EXAMPLE #2 (CONT'D)

```
C          SWITCH SCANNING TASK
C
C          "JSCAN"
C
C          THIS TASK SCANS THE CONSOLE SWITCHES FOLLOWING EACH
C          USER-DEFINED CLOCK PSEUDO-INTERRUPT.  IF A CHANGE IN
C          SETTING HAS OCCURRED, THE ALARM TASK IS AWAKENED.
C
           TASK JSCAN
           COMMON/SWCH/IOLD(-1:15),INEW(-1:15),ICNT,TCNT
           COMMON/KEY/KEYSW,KEYA,KEYS
           INTEGER TCNT
1          CALL RDSW(INEW)
           IF(IOLD(-1)-INEW(-1))10,20,10     ;IF OLD SAME AS NEW,
10         CALL XMT (KEYA,1,$20)              NO ALARM
20         IOLD(-1)=INEW(-1)                 ;CALL ALARM TASK
           CALL REC(KEYSW,IDUM)              ;WAIT FOR NEXT USER
           GOTO 1                             CLOCK MSG
           END

C          ALARM TASK
C
C          "JALARM"
C
C          THIS TASK IS AWAKENED BY THE SCAN TASK WHEN A SWITCH
C          HAS CHANGED STATE. IT ALSO GENERATES AN APPROPRIATE
C          MESSAGE.
C
           TASK JALARM
           COMMON/SWCH/IOLD(-1:15),INEW(-1:15),ICNT,TCNT
           COMMON/KEY/KEYSW,KEYA,KEYS
           COMMON/TEXT/IOK,(IUP
           DIMENSION ITEXT(3),IOK(3),IUP(3)
           INTEGER TCNT
           DATA ICNT,TCNT/2*0/
           DATA IOK,IUP/"OK   15","ALARM 7"/
1          CALL REC (KEYS,IKEY)              ;WAIT TO BE AWAKENED
           DO 100 I=0,15,1
           IF(INEW(I).EQ.IOLD(I))GOTO 99
           DO 20 J=1,3
20         ITEXT(J)=IOK(J)                   ;INITIALIZE MSG TO "OK"
           IF(INEW(I).EQ.0)GOTO 40
           DO 30J=1,3
30         ITEXT(J)=IUP(J)                   ;SET MSG TO ALARM
           ICNT=ICNT+1                       ;INC THE ONE MINUTE COUNT
           TCNT=TCNT+1                       ;INC THE OVERALL COUNTER
40         WRITE(10,110) I,ITEXT
X          WRITE(12,110) I,ITEXT
99         IOLD(I)=INEW(I)                   ;UPDATE THE OLD SWITCH
100        CONTINUE                           SETTINGS
           GOTO 1                            ;GO BACK TO SLEEP
110        FORMAT(1X,"SWITCH NO",I2,2X,3A2)
           END
```

MULTI-TASKING EXAMPLE #2 (CONT'D)

```
C         LOGGER TASK
C
C         "JLOG"
C
C         THIS TASK RUNS ONCE PER MINUTE AND TYPES THE NUMBER OF
C         ALARMS OF THE PREVIOUS MINUTE AND THE TOTAL SINCE
C         PROGRAM INITIATION.
C
          TASK JLOG
          COMMON/SWCH/IOLD(-1:15),INEW(-1:15),ICNT,TCNT
          INTEGER TCNT
1         IKEY=ICNT
          ICNT=0                              ;RESET ONE MINUTE ALARM CNTR
          WRITE(10,100)IKEY,TCNT              ;OUTPUT MSG'S
X         WRITE(12,100)IKEY,TCNT
100       FORMAT(1X,I3,2X,"ALARMS SINCE LAST LOG"/
         1       1X,I3,2X,"ALARMS SINCE TEST BEGAN")
          CALL SUSP
          GOTO 1
          END

;         INTERRUPT SERVICE FOR THE USER CLOCK
;
;         "JCLOCK"
;
;         THIS IS A FORTRAN IV COMPATIBLE ASSEMBLY LANGUAGE
;         INTERRUPT SERVICE PROGRAM FOR THE USER DEFINED CLOCK.
;         IT WILL SEND A MESSAGE TO THE FIRST WORD OF A FORTRAN IV
;         LABELLED COMMON AREA CALLED "KEY".
;
          .TITL JCLOCK
          .ENT JCLOC
          .EXTN .IXMT,.UCEX
          .NREL

          .COMM KEY,3                         ;DEFINES LABELLED COMMON BLK

JCLOC:    STA   3,TEMP                        ;SAVE THE RETURN LOCATION
          LDA   0,.MESS                       ;GET THE MESSAGE ADDRESS
          SUBZL 1,1                           ;SET THE MESSAGE TO NON-ZERO
          .IXMT
          JMP   .+1                           ;IGNORE ERRORS!
          ADC   1,1                           ;FORCE RESCHEDULING ON EXIT
          LDA   3,TEMP                        ;RESTORE THE RETURN LOCATION
          .UCEXIT

.MESS:    .GADD KEY,0                         ;GET ADDRESS OF FIRST ELT OF
                                              ;COMMON AREA "KEY"
TEMP:     0

          .END
```

MULTI-TASKING EXAMPLE #2 (CONT'D)

```
;          SUBROUTINE TO READ THE CONSOLE SWITCHES
;
;               "RDSW"
;
;          THIS IS A FORTRAN IV CALLABLE ASSEMBLY LANGUAGE SUBROUTINE
;          WHICH IS USED TO READ THE CONSOLE SWITCHES INTO A 17-WORD
;          INTEGER ARRAY. THE FIRST WORD, IARRAY(-1) IS THE SWITCH
;          POSITIONS AS READ. THE NEXT 16 WORDS, IARRAY(0) THRU
;          IARRAY(15) ARE THE BINARY VALUES OF EACH SWITCH (WHERE:
;          0=OK OR OFF, 1=ALARM OR ON).
;
;          CALLING SEQUENCE:
;               CALL RDSW(IARRAY)
;
;
          .TITLE    RDSW
          .NREL
          .ENT      RDSW
          .EXTD     .CPYL,.FRET

          AR=-167
          .FS=1

          .FS

RDSW:     JSR       @.CPYL        ;COPY ARGUMENT LIST AND SET UP RTN
          LDA       2,AR,3      ;FETCH IARRAY'S ADDRESS
          LDA       0,C16       ;INITIALIZE THE LOOP COUNTER
          STA       0,CNTR
          READS     0           ;READ THE CONSOLE SWITCHES AND
          STA       0,0,2       ;STORE IN THE FIRST WORD OF IARRAY
LOOP:     INC       2,2         ;TALLY ARRAY ELEMENT ADDRESS
          SUBO      1,1         ;INITIALIZE SWITCH VALUE TO ZERO
          MOVZL     0,0,SZC     ;SET VALUE ACCORDING TO THE CARRY
          INC       1,1         ;BIT SET SWITCH VALUE TO ONE
          STA       1,0,2       ;STORE SWITCH VALUE(0 OR 1)IN ARRAY
          DSZ       CNTR        ;TALLY THE LOOP COUNTER AND
          JMP       LOOP        ;CONTINUE THE LOOP
          JSR       @.FRET      ;RETURN TO THE CALLING FORTRAN
                                ;PROGRAM

C16:      20
CNTR:     .BLK      1

          .END
```

APPENDIX C

INDEX FOR

C L I

REFERENCE

MANUAL

INDEX TO "RDOS COMMAND LINE INTERPRETER REFERENCE MANUAL" -93-109

COMMAND	PAGE	FUNCTION
DUMP	3-49	Dump files
EDIT	3-15	Edit in the background
ENDLOG	3-51	Close the log file
EQUIV	3-75	Assign a new name to a directory specifier
EXFG	3-86	Execute in the foreground
FILCOM	3-52	Compare two files
FORT	3-16	Compile and assemble a FORTRAN IV
FORTRAN	3-18	Perform a FORTRAN 5 compilation
FPRINT	3-53	Print a disk file in bytes, decimal, hexidecimal or octal
GDIR	3-76	Print the current directory device name on $TTO or $TTO1
GMEM	3-87	Get the foreground/background memory size in a mapped system
GSYS	3-102	Get the name of the current operating system
GTOD	3-103	Get time and date
INIT	3-77	Initialize a directory device, mag tape or cassette
LFE	3-19	Update library files
LINK	3-79	Create a link to file on the same or another directory
LIST	3-81	List file directory information
LOAD	3-54	Reload dumped files
LOG	3-56	Record all CLI dialogue
MAC	3-22	Perform a MACRO Assembly
MCABOOT	3-114	Transmit a program via an MCA line
MDIR	3-84	Get the name of the current master directory

COMMAND	DATE	FUNCTION
SQUASH	3-92	Prepare a system save file for use in bootstrapping
STOD	3-105	Set the time
SYSGEN	3-90	Generate a new operating system
TPRINT	3-96	Print the timing file
TUOFF	3-95	Turn off system timing facility
TUON	3-94	Turn on system timing facility
TYPE	3-68	Output the contents of a file on the system console
UNLINK	3-69	Delete a link entry name
XFER	3-70	Copy the contents of a file to another file

APPENDIX D

INDEX FOR TROUBLESHOOTING RDOS CRASHES

TROUBLESHOOTING RDOS CRASHES

A. Symptoms

1. RDOS PANIC

   a. Five octal numbers typed on $TTO, then HALT. Record these #s; last # = panic code.
   b. If panic code has 1BØ; consult Appendix G of RDOS Reference Manual.
   c. If panic code has ØBØ user system call caused panic. Consult Appendix A of RDOS Reference Manual.

2. RDOS Deadlock (Spinning its wheels)

   a. RUN light on but no recognition of $TTI.
   b. Hit STOP and record PC value. Compare this value to SMON and PENTR (on load map).
   c. If PC is between these values it indicates that the system is looking for something to do; probably waiting for a resource (stack, buffer, cell) to become available.
   d. If PC = 12_8 , user program probably did a JMP Ø and worked its way up to 12/JMP self.

3. System Died

   a. RUN light on or off? If on, probably a deadlock, go back to step 2.
   b. If off, record PC, compare to SYST (on loader map).
   c. If PC SYST, halted in RDOS.
   d. If PC SYST, halted in user.

B. Diagnosis

1. RDOS page Ø.

   a. Record the octal content of LOC Ø - 15_8 + $40\text{-}47_8$
   b. Locate these values on the system load map and compare them

2. Other key locations - Record on attached sheet.

   a. OVTAB - The start address of a table of overlay status information. One entry per system overlay; content of CRSEG (LOC 14) minus value of OVTAB equals the overlay number of the most recently executing overlay.

b. Content of CC (LOC 5) is the address of the current cell where:

displacement	meaning
Ø	Address of user TCB
4	Entry point to RDOS routine currently processing
11	Pointer to PT1 (if FG) or PT2 (if BG)

c. BCEC - This value (taken from load map) indicates the number of system buffers that were asked for during sysgen.
d. MNSTK - (taken from load map. This is the number of stacks that were asked for during sysgen.
e. BQ - This value is the address of the first system buffer.
f. RTCI - Real Time Clock Increment: 1 = 10Hz. etc.

3. Check the Interrupt World

a. INTLV - The content of (not the value) INTLV indicates the level of nested interrupts. The eigth level will cause an RDOS panic (RDOS Ref. Man., G-1, code 3).
b. LINT - The content of this address is the code of the last device whose interrupt was acknowledged; should never be RTC (code 14).
c. SS - This value is the Starting address of the interrupt Stack. (RDOS uses a separate stack for the processing of interrupts versus the processing of system calls).

4. Check the System World

a. SYSIN - The content of this address will = Ø if the crash occurred while executing user code or, = 1 if executing system code. Any other value is garbage.
b. INTSK - The content of this address will = Ø if RDOS is trying to schedule a system task or, = 1 if RDOS found something to do and is now doing it (or at least trying to do it).
c. CQ - (LOC 13) The content of this address should correspond to a symbol (on the load map) consisting of a device mnemonic and ending with the letter Q. For example: TTIQ, PTPQ, LPTQ. Possible exceptions also considered legitimate include DSQ1, DSQ2, SPOLQ, PT1, PT2 and SYSQ.

d. SACHN - For each parallel operation involving user program and/or I/O device there is an associated Queue Control Block. These QCBs are linked together and the content of SACHN (Start of the Active CHaiN) points to the first one in the chain.

5. The System Active Chain

a. As stated above, the active chain will consist of a number of QCBs linked together; but will always end with the same three: SPOLQ, PT1 and PT2. From this chain there is certain information which is helpful to you (or your local A.E.) in troubleshooting the RDOS crash.

1. QSTAT - At displacement 4 is status information about this queue entry such as whether or not it is pended. Based on the type of information found here, displacement 11_8 may provide further status information.
2. QALNK - At displacement 5 is a pointer to the next QCB in the chain. This address should be on the load map corresponding to the rules of 4C above.
3. QCURR - At displacement 10_8 is the address of the current cell associated with this queue entry.
4. QKEY - At displacement 11_8 is additional status information which may or may not be meaningful depending upon the content of displacement 4.

b. After this information has been recorded for SPOLQ, the last two entries (PT1 and PT2) are slightly different. PT1 is for a foreground program and PT2 is for the background program.

1. PSTAT - At displacement 4 of a program table is status information about the program.
2. POLNK - At displacement 5 is the pointer to the other program table. In displacement 5 of PT2 should be a -1 indicating the end of the chain.
3. PPC - At displacement 7 is a pointer to PENTR (program entry routine). PENTR is the last routine in RDOS before going to the user task scheduler.
4. PTSPN - At displacement 20_8 is the number of program swaps that this program has done. For PT2 (BG) this should be in the range of Ø-4. For PT1 (FG) it should be in the range of 4-10. The initial values (Ø&4) indicate no swaps have taken place.