HEWLETT
PACKARD

RTE-A PROGRAMMER

&

SYSTEM MANAGER

VOLUME III

data systems
training center

STUDENT WORKBOOK

22999-90546 FEB. 1984

THE SYSTEM MANAGER'S JOB

CHAPTER 13

Table of Contents

Chapter 13
THE SYSTEM MANAGER'S JOB

MODULE OBJECTIVES

1. Be able to describe the various functions of the System Manager.

2. Learn what will be covered in the system management portion of the course.

SELF-EVALUATION QUESTIONS

13-1. List four responsibilities of the System Manager and the duties required with each function.

13-2. What is a primary system?

13.1 Responsibilities of a System Manager

The System Manager is a superuser and thus has full system capability, including read/write access to any directory or file on the system. There may be more than one superuser on the system. These will be people who have an in-depth understanding of the system operation. Typically, only the System Manager will be performing the functions listed.

Planning -- Deciding what the new system will be like.

Generation -- Creating the new system.

Installation -- Getting the new system running.

Maintenance -- Keeping the system usable.

RESPONSIBILITIES OF A SYSTEM MANAGER

1. PLANNING

2. GENERATION

3. INSTALLATION

4. MAINTENANCE

13.2 Planning your system

These things must be taken into account to insure that the system meets the needs of the users and the applications running.

PLANNING YOUR SYSTEM

* Who will be using your system? Will it be a multiuser system?

* What type of applications will be run on your system?

* What system resources and peripherals will be required?

13.3 Generating the System

The generation process creates a system which is customized for your application.

The primary system is one supplied by HP as part of the software product. It is a generic operating system that includes most common peripherals and a standard disc layout. The primary allows you to start using your computer right away and provides a starting point from which to create your customized system.

GENERATING THE SYSTEM

An RTE–A system is "generated" by using the generator program RTAGN.

The System Manager can generate the system on:

* an existing operating system
* the primary system

The primary system is a pre–generated system with:

* system utilities
* most supported I/O devices

13.4 Installing the System

BOOTEX -- A memory based system that has the sole responsibility of loading and initializing a disc based operating system.

Boot Command File -- Contains commands for the BOOTEX system to initialize the disc based system.

Welcome File -- A CI command file that is run by the RTE-A operating system to mount volumes, initialize devices, etc.

System Utilities -- These include the copy of CI that users will share (VC+) and commonly used programs (eg. WH).

INSTALLING THE SYSTEM

AFTER THE NEW RTE–A SYSTEM HAS BEEN GENERATED, YOU WILL NEED TO:

- Install the boot extension area (BOOTEX)
- Prepare the boot command file
- Prepare the welcome file
- Load system utility programs

BOOT UP THE NEW SYSTEM

13.5 Maintaining the System

MAINTAINING THE SYSTEM

AS THE SYSTEM IS BEING USED, YOU MIGHT NEED TO:

. Alter user accounts

. save/restore disc LUs and files

. Alter system parameters

. Add updated software

. Answer questions pertaining to system operation

. Act as Hewlett Packard's contact at the installation

SYSTEM DESIGN AND PLANNING

CHAPTER 14

Table of Contents

Chapter 14
SYSTEM DESIGN AND PLANNING

MODULE OBJECTIVES

1. Describe the components of physical memory and logical system memory. Be able to describe the relationship between logical and physical memory.

2. Be able to make effective system planning decisions: I/O layout, what modules to include and why.

3. Be able to describe the system tables.

4. Explain why the disc layout would be modified; be able to fill out the disc configuration worksheets for both CS80 and non CS80 discs.

SELF-EVALUATION QUESTIONS

14-1. What is the difference between reserved and dynamic memory partitions?

14-2. What is a driver partition?

14-3. Which modules are required in every system?

14-4. What is an RPL file?

14-5. How many of each of the following tables are in the system?

LUT
DVT
IFT
INT

14-6. Define the following disc configuration concepts?

Surface mode
Cylinder mode
Physical disc block
Logical disc sector
Block addressing

14.1 SYSTEM DESIGN CONCEPTS

SYSTEM DESIGN CONCEPTS

14.2 Physical Memory Map

System Base Page - Contains the A- and B-Registers, interrupt trap cells, links to other memory locations and temporary storage for the VCP/loader ROM.

System Modules - The area of memory reserved for the operating system the device and interface drivers which do not support driver partitioning.

Driver Partition - Each partition contains 1 or more drivers which will be mapped into the system as needed. keep as small as possible

System Tables - Tables containing data used by the system, such as I/O configuration, program ID segments, etc.

System Common Partition - Mapped into the user logical map when a program accesses system common.

System Message Block - A data structure which contains system messages.

System Available Memory - A block of physical memory used for buffered I/O, class I/O and program-to-program communication. SAM may be up to 32K words.

Reserved User Partitions - Blocks of physical memory reserved for programs which are assigned to run in a specific partition.

Dynamic Partition Areas - Physical memory which is allocated for programs on demand.

References: System Design Manual

PHYSICAL MEMORY MAP

DYNAMIC PARTITION AREA	HIGHEST ADDRESS
RESERVED USER PARTITION #n	
• • •	
RESERVED USER PARTITION #1	
SYSTEM AVAILABLE MEMORY (SAM)	
SYSTEM MESSAGE BLOCK	
SYSTEM COMMON	
SYSTEM TABLES	
DRIVER PARTITION #n	
• • •	
DRIVER PARTITION #1	
SYSTEM MODULES	
SYSTEM BASE PAGE	ADDRESS 0

14.3 Logical Memory Map of the System

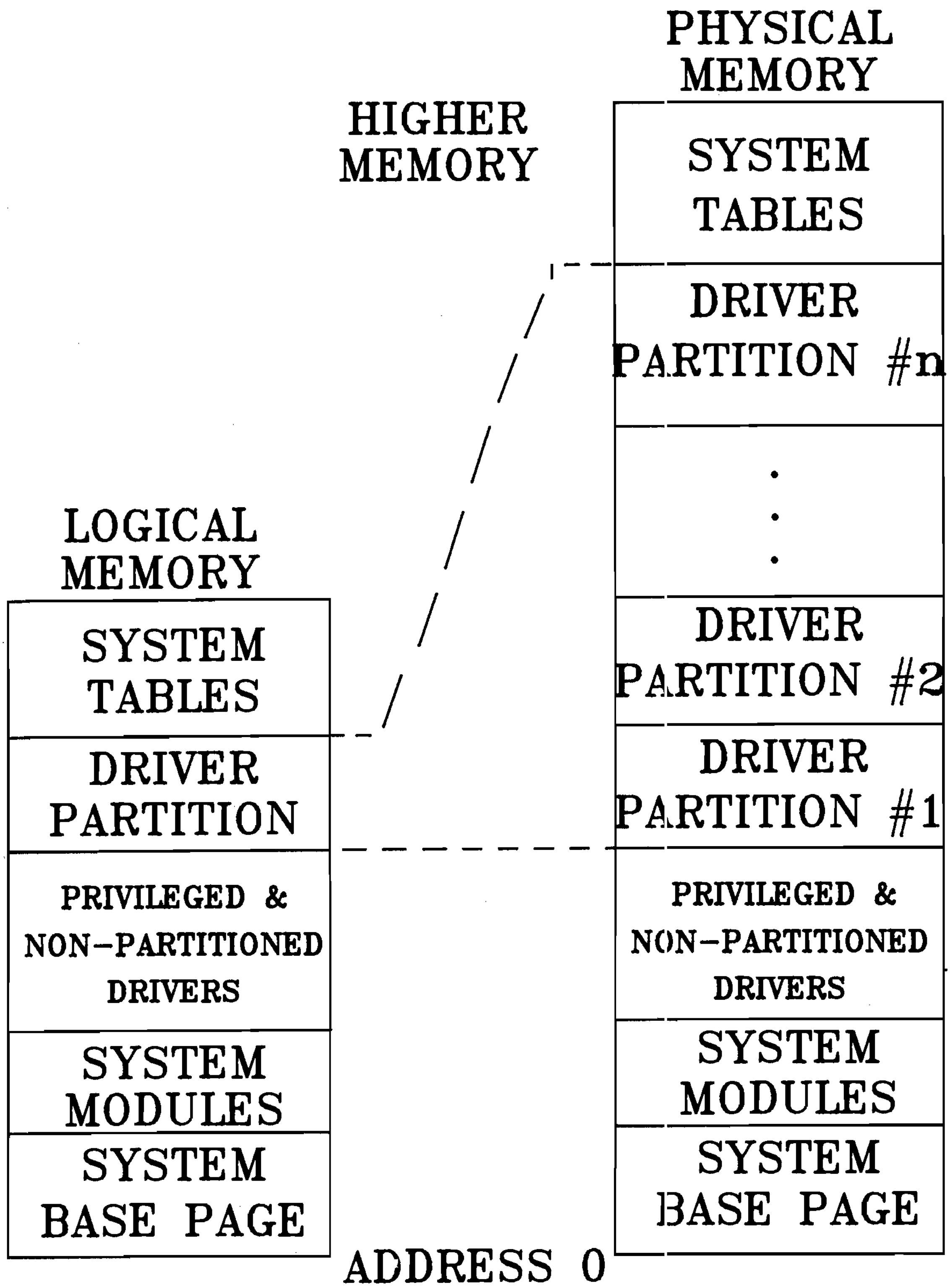
* Privileged Drivers - Drivers whose interrupts are not processed by the RTE-A operating system.

The only standard RTE-A driver which is not partitionable is ID.43, the powerfail driver.

Driver Partition - A contiguous set of pages in the system logical memory map. The size of this partition is the same as the size of the largest driver partition (defined at generation time).

References: System Design Manual

LOGICAL MEMORY MAP



14.4 Dynamic Mapping System

The dynamic mapping system consists of 32 sets of map registers, with 32 registers in each set.

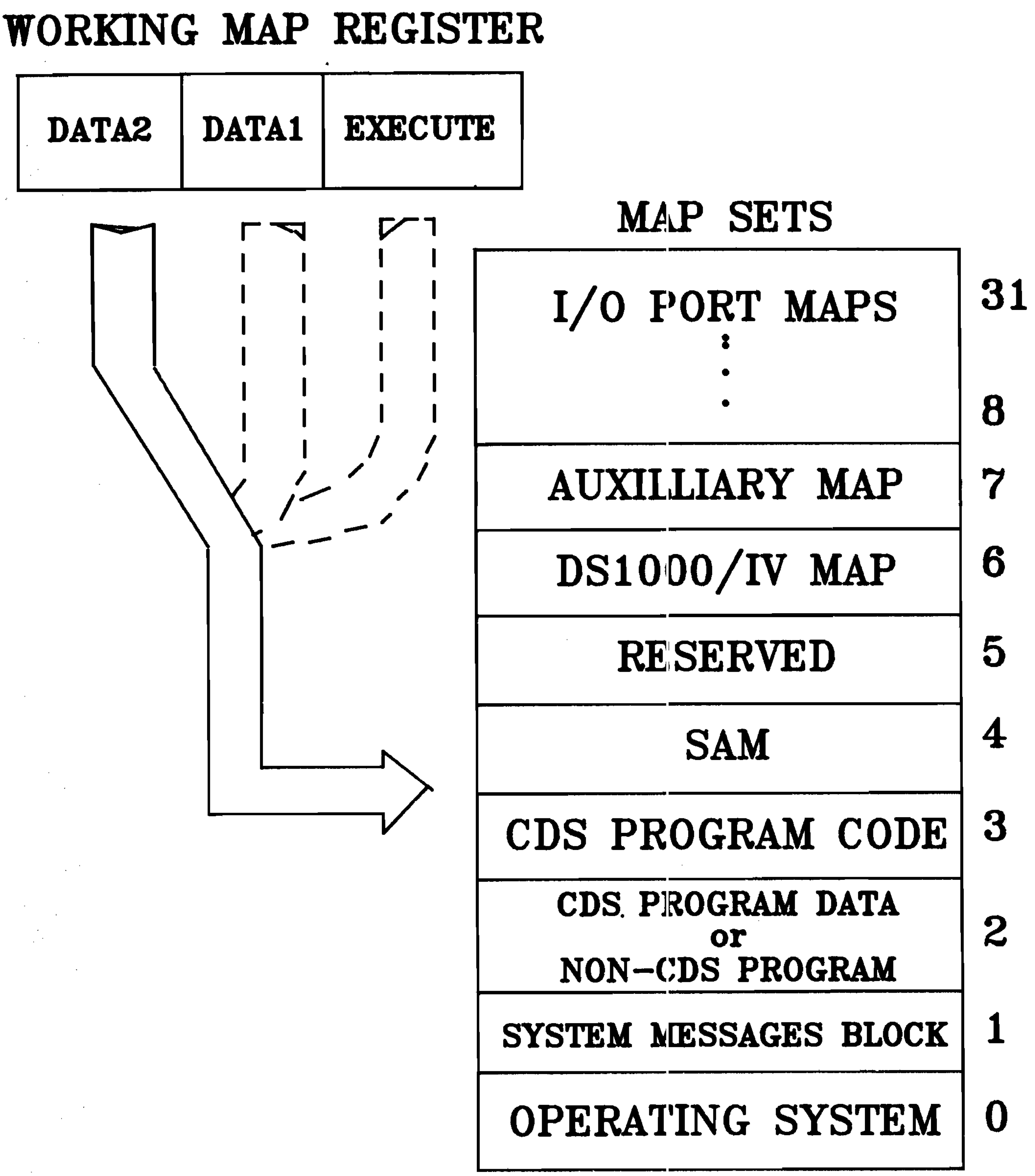
Working Map Register - A special purpose register which holds the map set number which is currently active. The EXECUTE map is used for instruction fetches. In the CDS (VC+) environment this map number also determines which map (data map = code map - 1) is used for program data. The DATA1 and DATA2 maps are used to access data in other map sets. (For example - to allow a user program to access data in the system map).

Port Map Sets (8-31) - Used individually by each select code (octal 20 through 47). There is a port map for each interface card, thus providing 24 channels of concurrent DMA.

Port map number = select code - 8

References: System Design Manual

DYNAMIC MAPPING SYSTEM



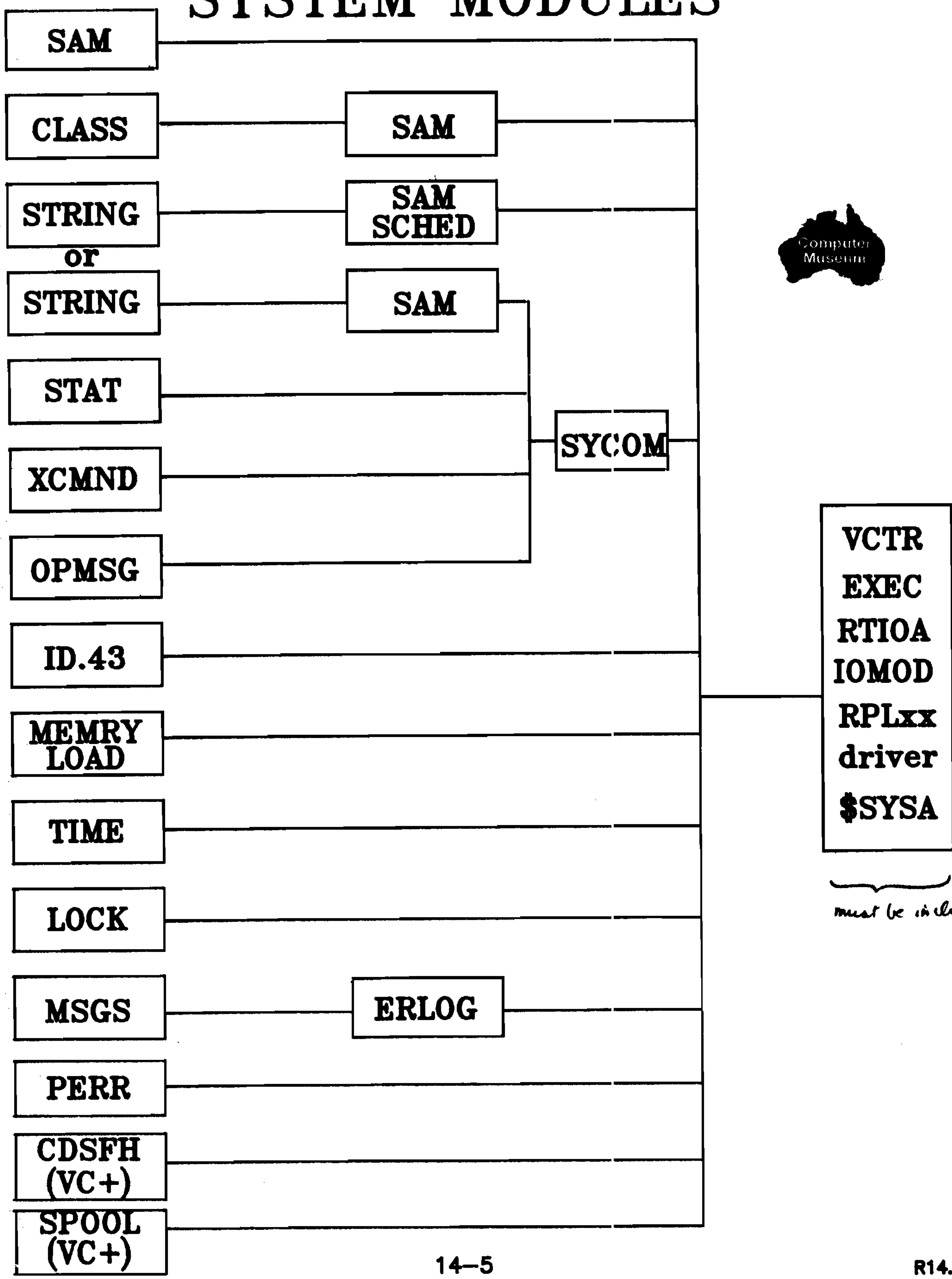
14.5 System Modules

RTE-A is a modular system; the system is built only with those pieces needed for your application. The system modules "tree" shows all the RTE-A system modules, indicating which modules require other modules in order to operate. The modules in the box at the right are required. For the program development environment it is usually a good idea to include all modules in the system.

CDSFH and SPOOL are available only with the VC+ option.

References: System Design Manual

SYSTEM MODULES



must be included

14.6 Required System Modules

VCTR - Contains system entry points. * If all system references are resolved in the VCTR module, then the program is transportable.*

RPLxx - This file defines software replacements for microcoded instructions. The particular RPL file used is based on the type of processor and the features used in your system. This can be found in the reference maunal.

EXEC - Routes all EXEC calls and processes EXEC 6, 7, and 29.

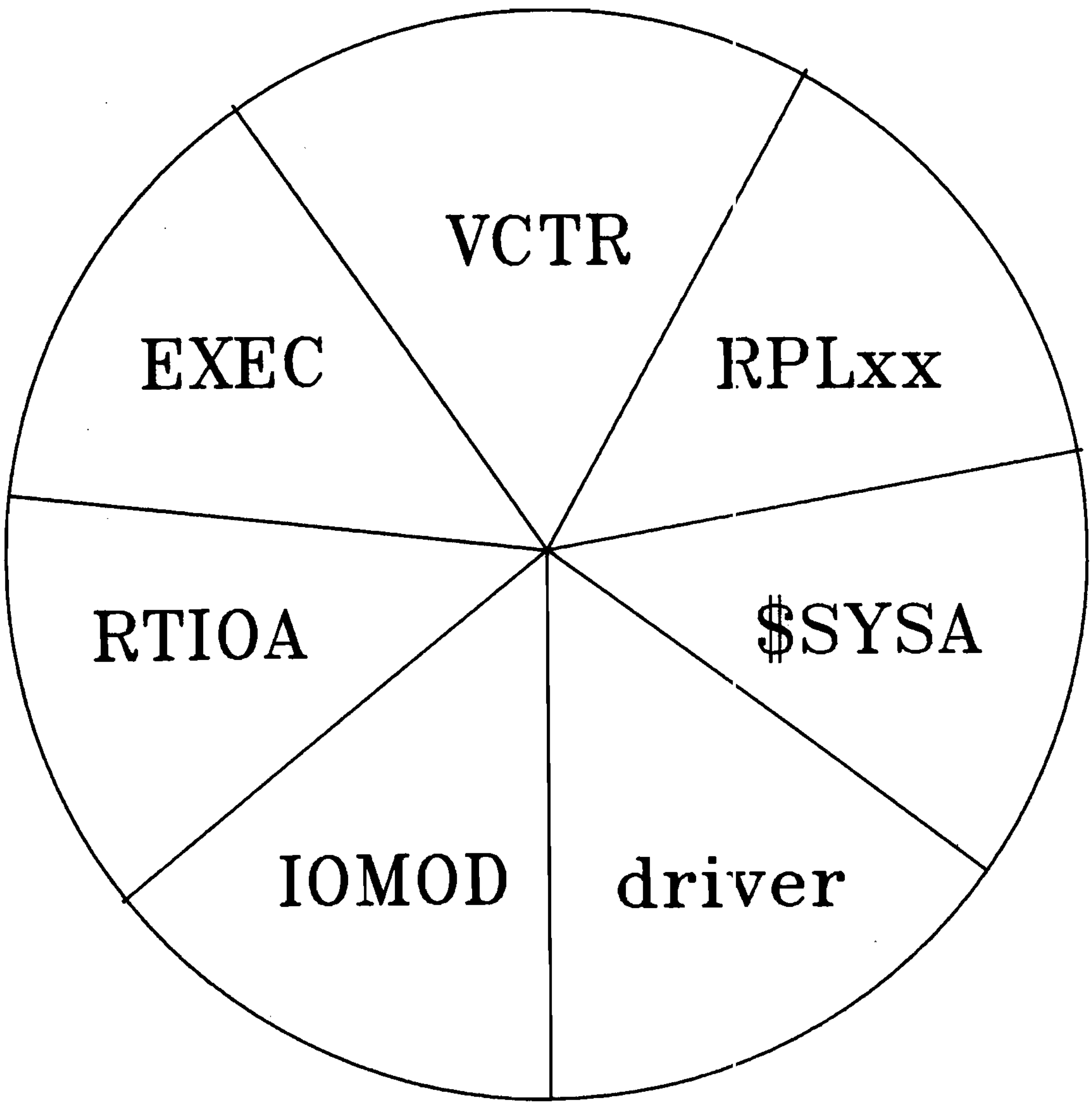
RTIOA - Handles all normal I/O requests (EXEC 1, 2, 3, 13), although IOMOD must also be present to actually perform the I/O.

IOMOD - contains subroutines for I/O and is the real time clock manager.

$SYSA - This is the dummy library - more later.

References: System Design Manual

REQUIRED SYSTEM MODULES



14.7 $SYSA

The dummy module may:

- Issue an error message
- Implement a NOP instead of the subroutine
- Abort the program with an illegal request

The action taken by a particular dummy module when invoked is described in the System Design Manual (chapter on Operating System Modules).

The name of the dummy module in $SYSA uses the first three characters of the real module followed by two periods.

At load time:

In each of the real modules, the entry point $$xxx will exist, where xxx are the first three characters of the name of the real module. This entry point will not exist in the dummy module.

At run-time:

The entry point $.xxx (where xxx are the first three characters of the real module name) will be set to 0 in the real module and will be set to -1 in the dummy module.

Note: An entry point is a location in a module which is accessible to external routines.

References: System Design Manual

$SYSA

RESOLVES REFERENCES TO MODULES WHICH HAVE BEEN OMITTED.

REAL MODULE = TIME
ENTRY POINT = $$TIM
ENTRY POINT $.TIM = 0

DUMMY MODULE = TIM..
NO ENTRY POINT $$TIM
ENTRY POINT $.TIM = −1

this tell system
the module is not
included

14.8 System Tables

System tables - In order to provide flexibility, and at the same time minimize system memory requirements, variable length tables are used to configure the operating system. The length of the tables or the number of tables is determined at generation time. The interface tables and device tables are 8 words and 25 words, respectively, but there may be extensions to these tables defined at generation time.

References: System Design Manual

SYSTEM TABLES

TABLE NAME	#WORDS
SWAP DESCRIPTOR TABLE	3 per descriptor
SHAREABLE EMA TABLE	5 per entry
MEMORY DESCRIPTOR TABLE	4 per reserved 7 per dynamic
ID SEGMENTS	43 per ID segment
RESOURCE NUMBER TABLE	1 per resource #
CLASS NUMBER TABLE	2 per class #
INTERRUPT TABLE	1 per select code
LOGICAL UNIT TABLE	1 per logical unit
INTERFACE TABLES	8+ per interface card
DEVICE TABLES	25+ per device (LU)
SHARED PROGRAMS TABLE	5 per shared program
MULTI-USER TABLE	20 per user

14.9 I/O Tables

There is one LU table (LUT) in the system which contains an entry for each LU number. The length of this table corresponds to the highest LU number in the system.

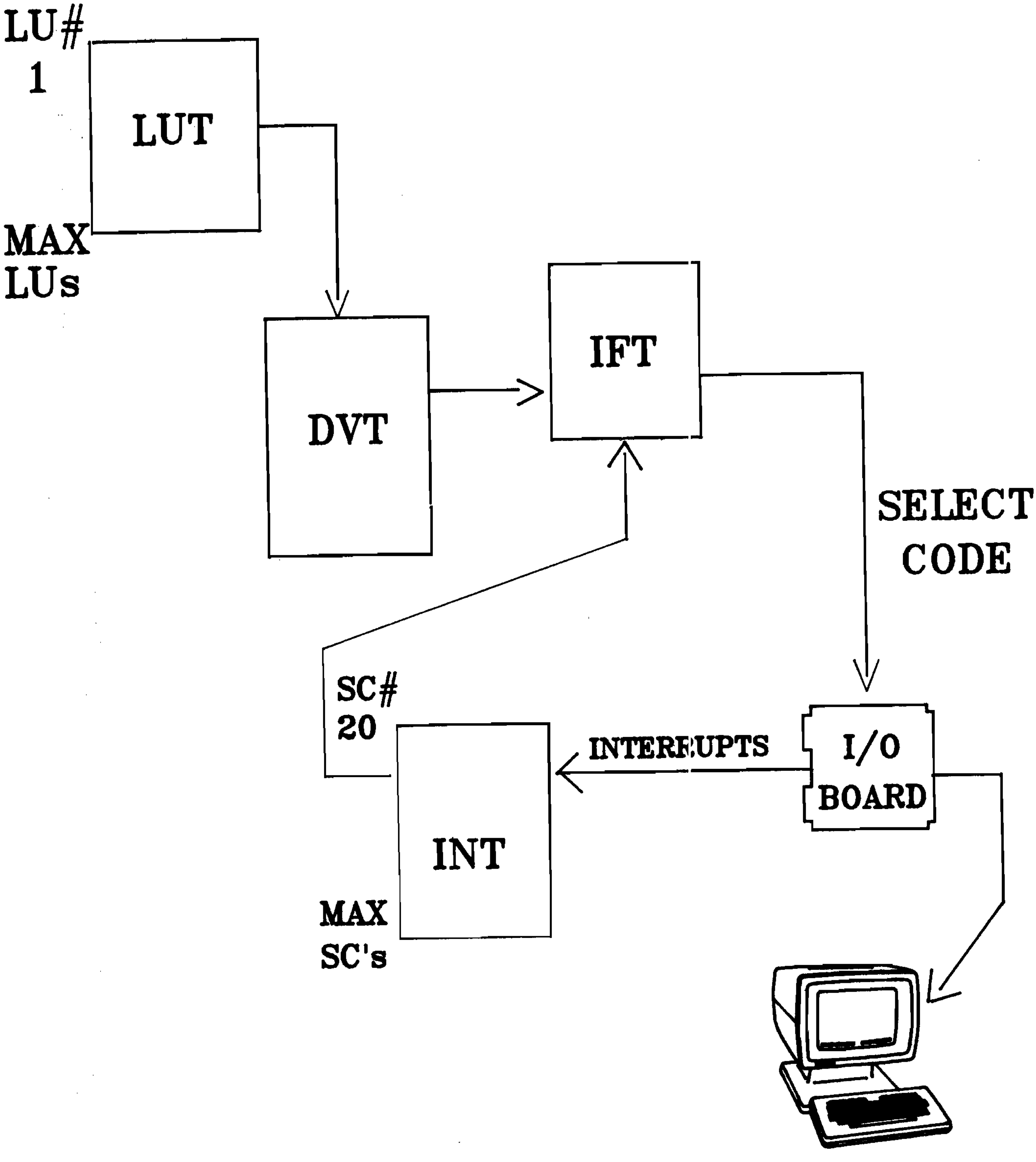
There is one device table (DVT) per LU. This is a fixed length table, although there is a corresponding driver parameter area and driver extension area which vary in length according to the driver specifications.

There is one interface table (IFT) per I/O card. This is a fixed length table with a corresponding interface extension area which varies according to the drive specifications. Several DVTs may be linked to one IFT, just as several devices may be connected to one interface card.

There is one interrupt table (INT) in the system, which contains an entry for each select code. The length of this table corresponds to the highest select code in the system.

References: System Design Manual

I/O TABLES

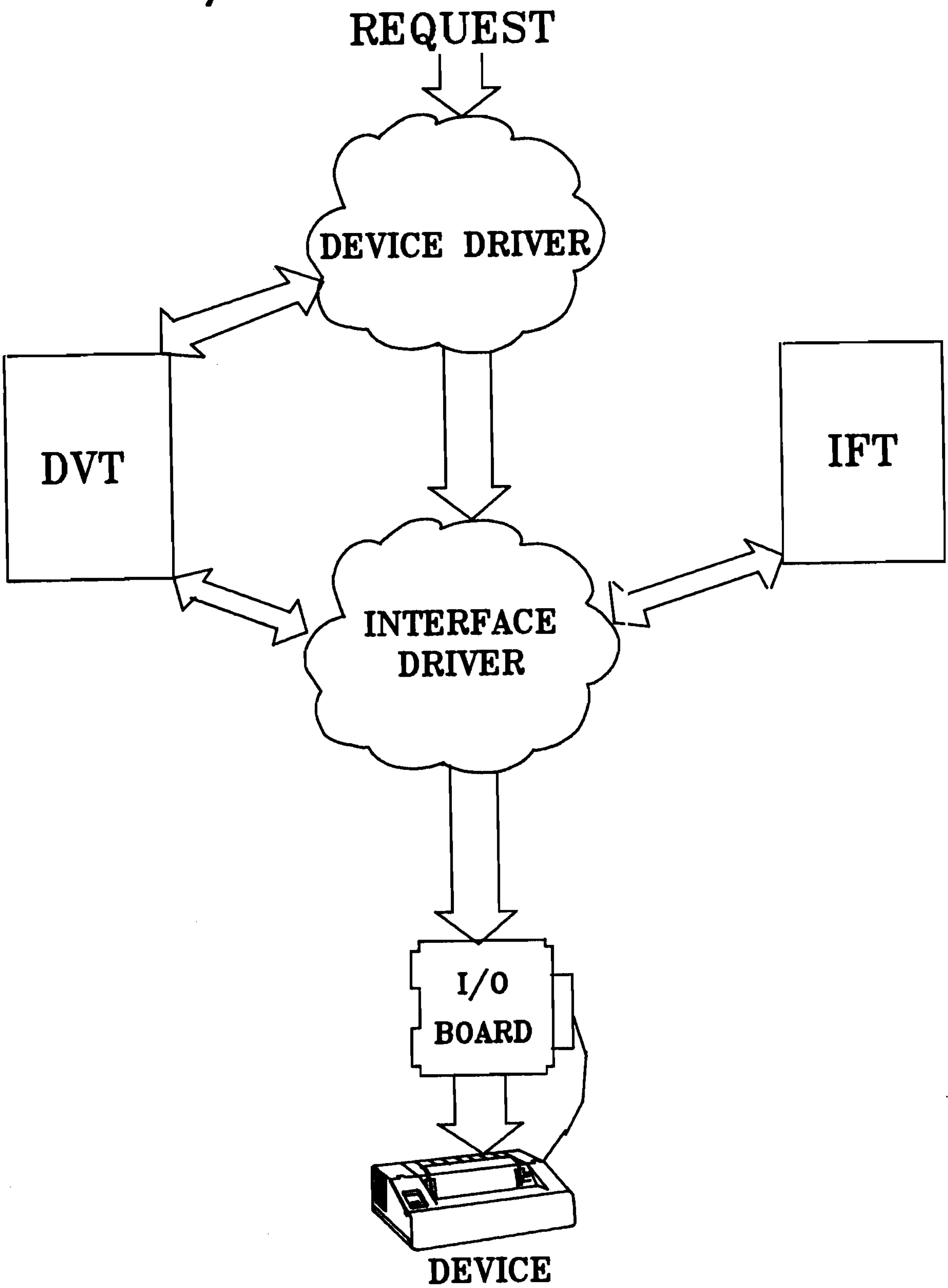

LU#
1
LUT
MAX
LUs
DVT
IFT
SELECT
CODE
SC#
20
INT
MAX
SC's
INTERRUPTS
I/O
BOARD

14.10 I/O Drivers

An I/O request is usually processed by 2 drivers - the device driver and the interface driver. The device driver is associated with a particular device (like a printer or a terminal). It formats the request for that specific device and passes the request on to the interface driver. The interface driver then converts the request into the actual I/O transfer (or transfers).

References: System Design Manual

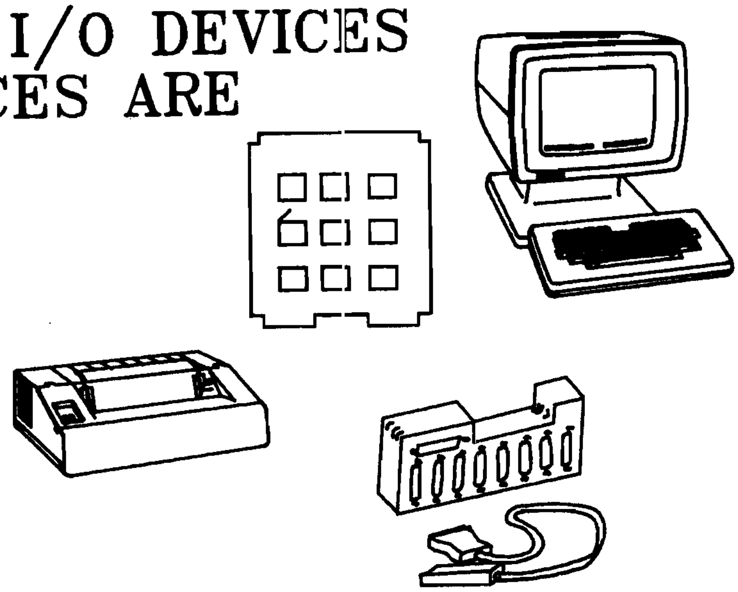
I/O DRIVERS

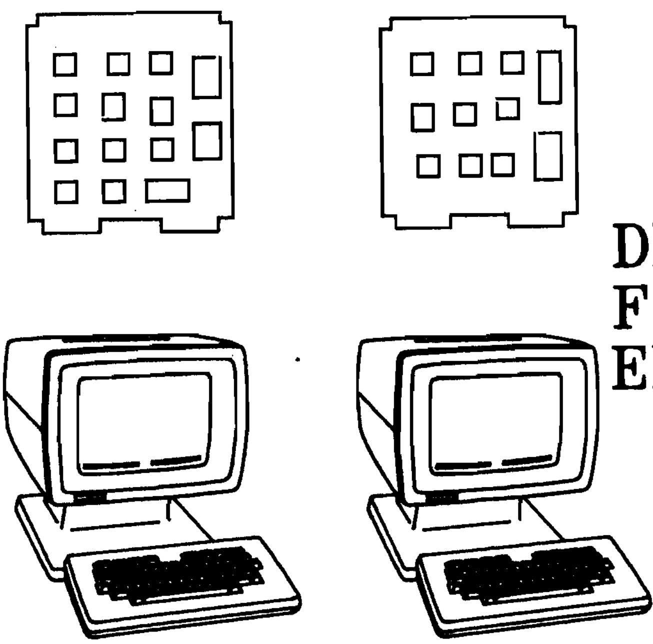

REQUEST
DEVICE DRIVER
DVT
IFT
INTERFACE DRIVER
I/O BOARD
DEVICE

14.11 I / O P L A N N I N G

I/O PLANNING

DECIDE WHAT I/O DEVICES AND INTERFACES ARE REQUIRED.





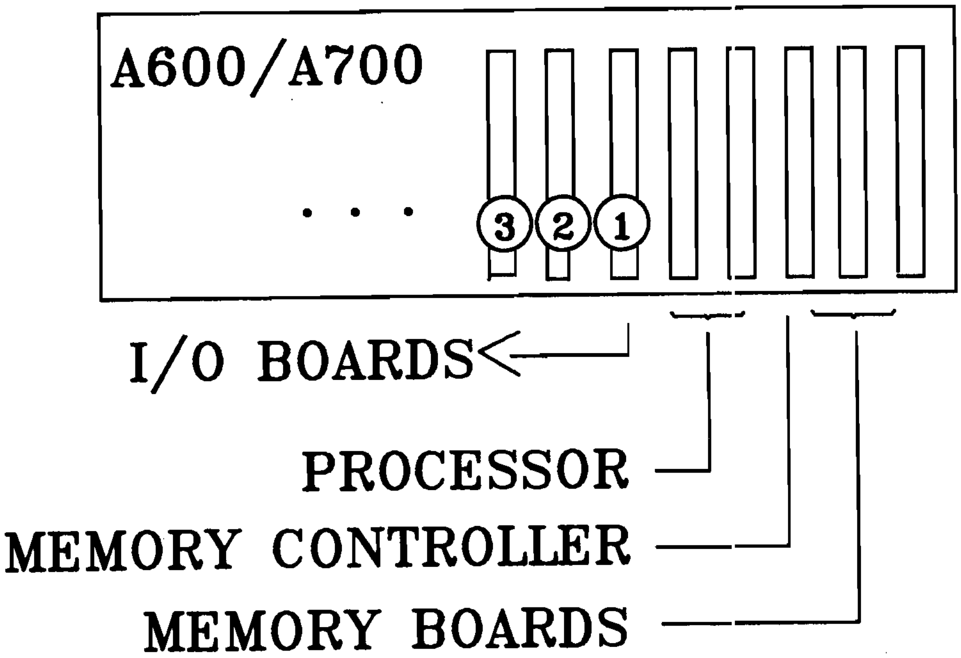
DESIGN IN EXTRAS FOR FUTURE EXPANSION.

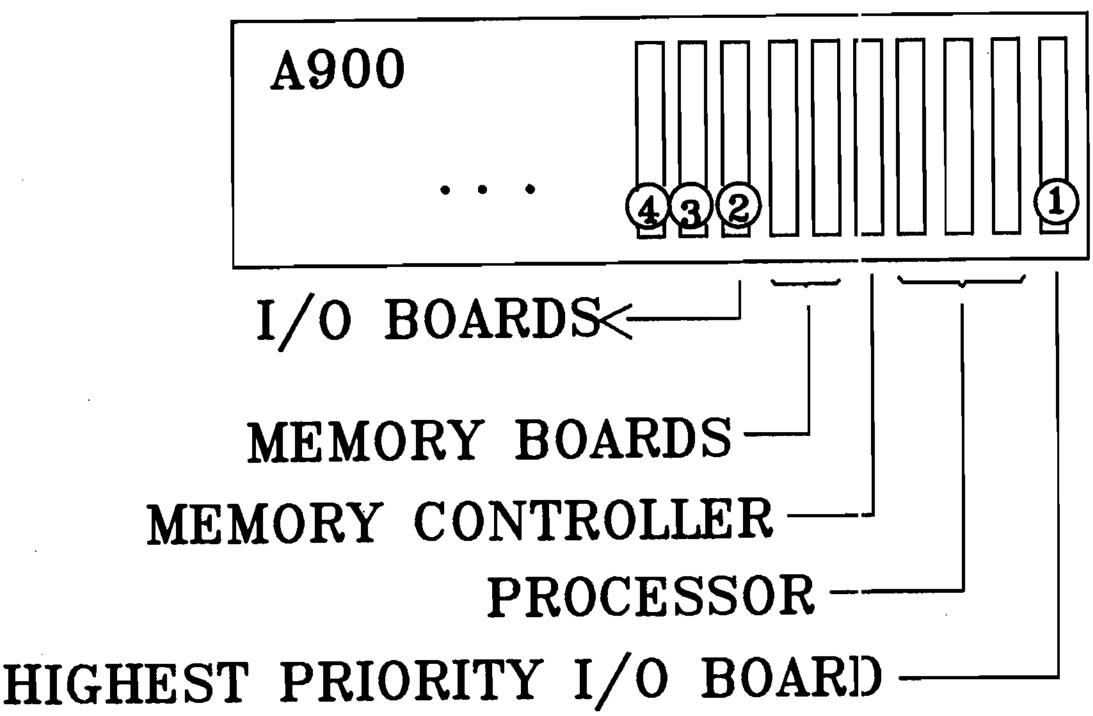
14.12 I/O Priority - 20 Slot Box

Priority of I/O cards is determined by the order of the cards in the backplane.

References: A600/A700/A900 Installation and Service Manuals

I/O PRIORITY
20 - slot box





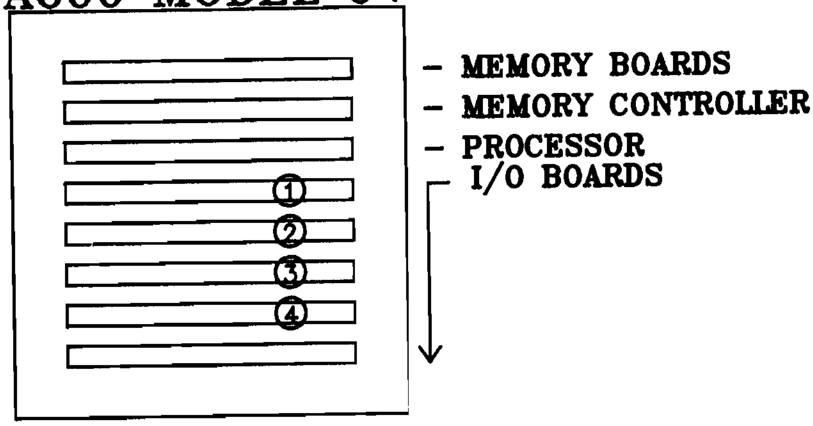
14.13 I/O Priority - Microsystems

The two bottom slots are reserved for the battery backup and 25 Khz modules and cannot be used for anything else. The battery backup takes the space of 3 slots, therefore, if it is used, the 2 slots directly above it can no longer be used for I/O cards.

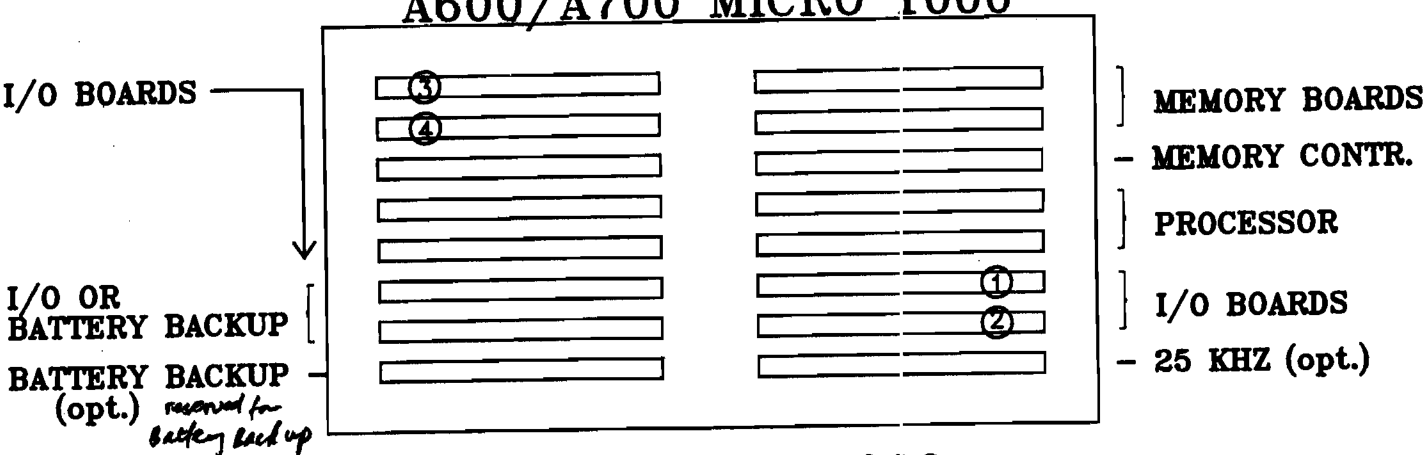
Refernces: Model 6 Instl & Serv, Micro 26/27/29 Instl & Serv Manuals

I/O PRIORITY MICROSYSTEMS

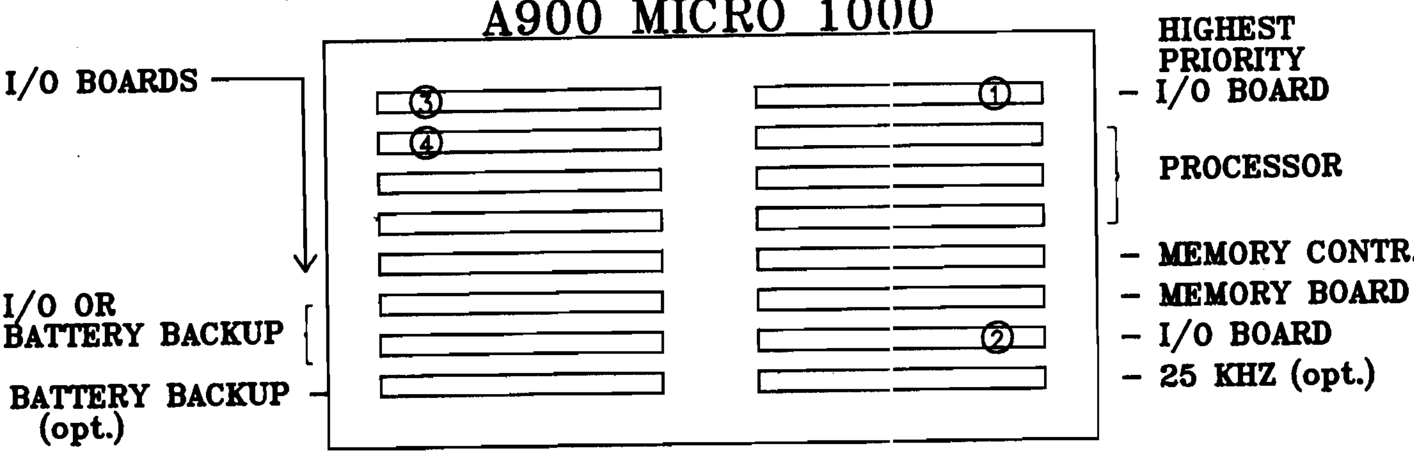
A600 MODEL 6+



A600/A700 MICRO 1000



A900 MICRO 1000



14.14 I/O Select Codes

Select code - is manually set on each I/O board. The select code provides the means by which cards in the backplane are addressed. The select code must be between 20B and 47B.

If a device using a select code is privileged, all devices using the select codes in the same group of four must also be privileged. These groups of four are:

20B-23B	34B-37B
24B-27B	40B-43B
30B-33B	44B-47B

Some standard conventions used for select codes:

20 = VCP asynchronous interface card
22 = PROM storage card
24 = VCP DS/1000-IV (HDLC) card
27 = boot disc (HPIB)

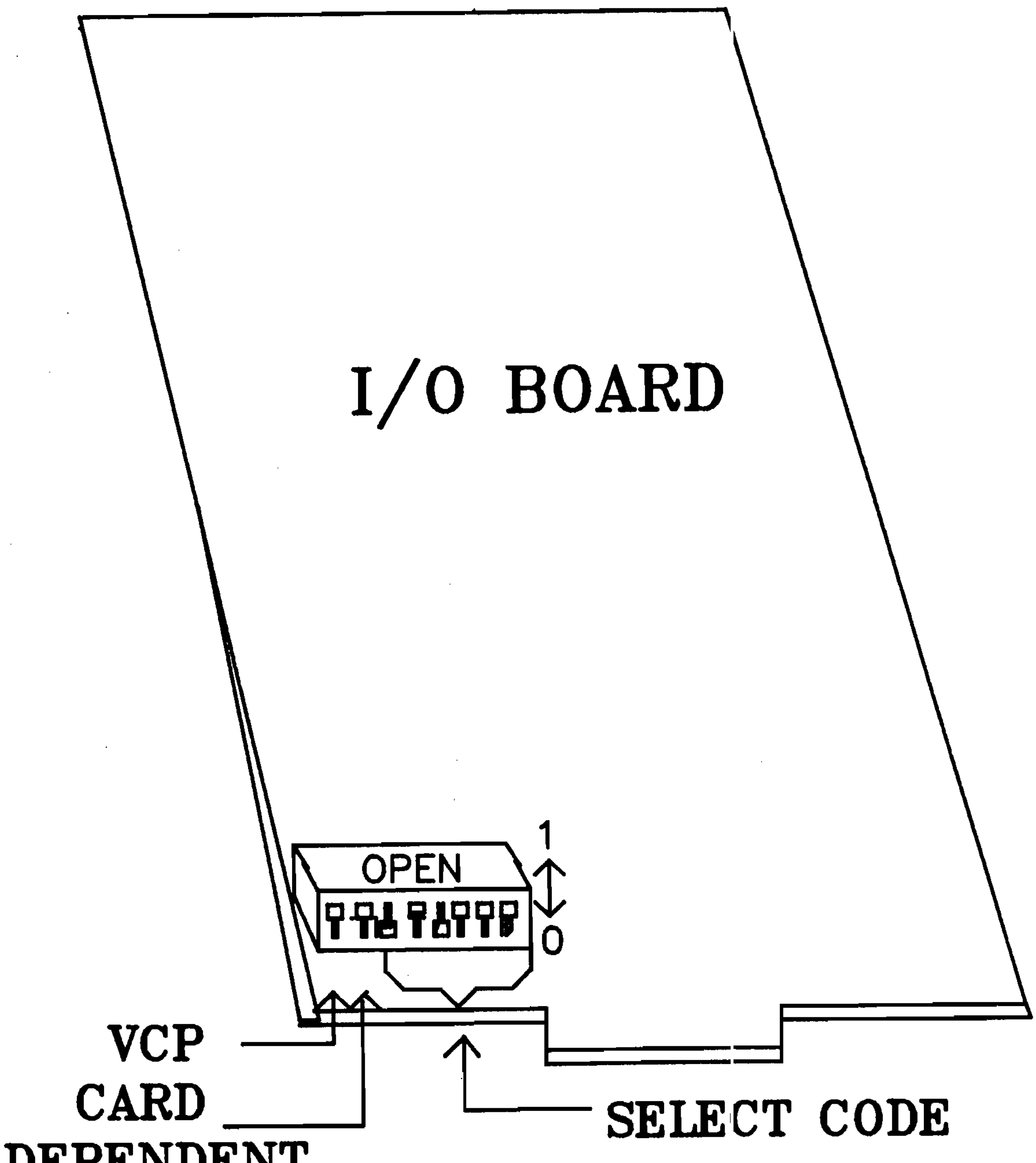
VCP - when this switch is closed (down) the device (usually a terminal) acts as the "virtual control panel" for the system.

Card dependent - The requirements for this switch are explained in the appropriate interface card manual. For example, HPIB cards use this switch to specify fast or slow transfer mode.

Note: The system requires a hard reset (that is, hitting the reset switch or cycling power) to acknowledge a change in select code.

References: System Gen. & Install. Manual, I/O installation manuals

I/O SELECT CODES



14.15 I/O Configuration Worksheet

Some interfaces cards allow only one device to be attached to them. An ASIC card, for example, only supports one terminal. Other interface cards may have more than one device attached to them.

The HPIB card supports multiple devices. Each device on the HPIB is assigned a unique address which the card uses to communicate with it. There may be up to 8 devices such as discs, printers, and mag tapes an an HPIB line. These must be assigned addresses between 0 and 7. Typically, you should not put printers on the same HPIB as your discs, as this will slow them down. There may be up to 32 instrument devices on an HPIB line, with assigned addresses between 0 and 31. Instruments may not be used on the same HPIB line as non-instrument devices. HPIB addresses are usually assigned by setting switches on the device. The HPIB address which will be associated with a specific LU is set up at generation time.

Up to 8 terminals (or other supported devices) may be connected to one terminal multiplexer interface card. Each terminal has a port number between 0 and 7. The port number which will be associated with a particular LU is set up when the LU is initialized.

References: System Generation and Installation Manual

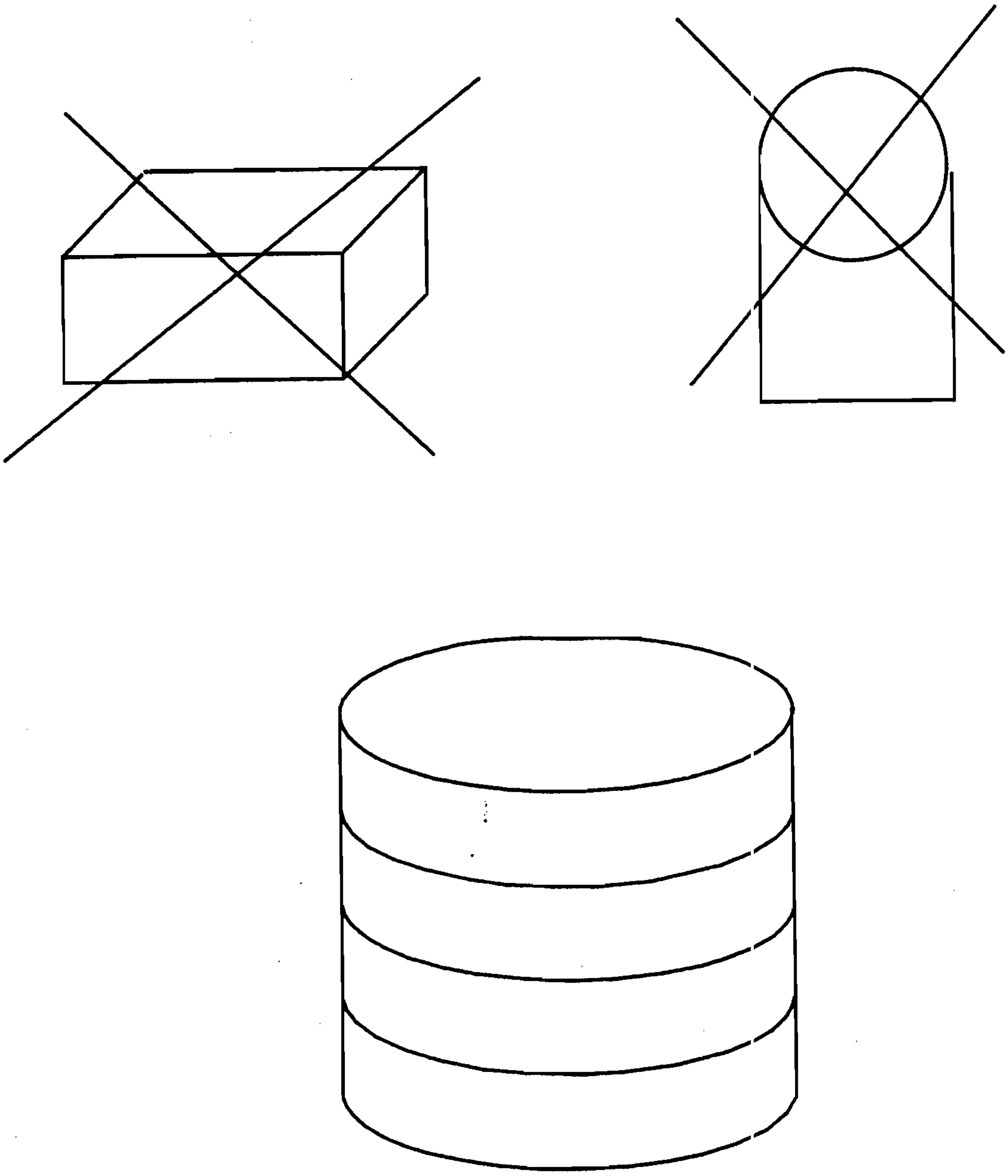
I/O CONFIGURATION WORKSHEET

DEVICE	INTERFACE	LOGICAL UNIT	SELECT CODE	HPIB ADDRESS
262X terminal	ASIC	1	20B	--
2631 printer	HPIB	6	25B	6

14.16 D I S C   C O N F I G U R A T I O N

The system manager can, at generation time, define the size and bounds of the disc LUs in the system.

DISC CONFIGURATION



14.17 File System Considerations

Under the RTE-A hierarchical file system it is generally best to have a small number of large disc LUs. Some subsystems or applications may require FMGR LUs. These are typically smaller LUs.

Advantages of larger LUs:

- More room for files and subdirectories under 1 global directory.
- Do not need to pack disc as often.
- Easier to find space for very large files.

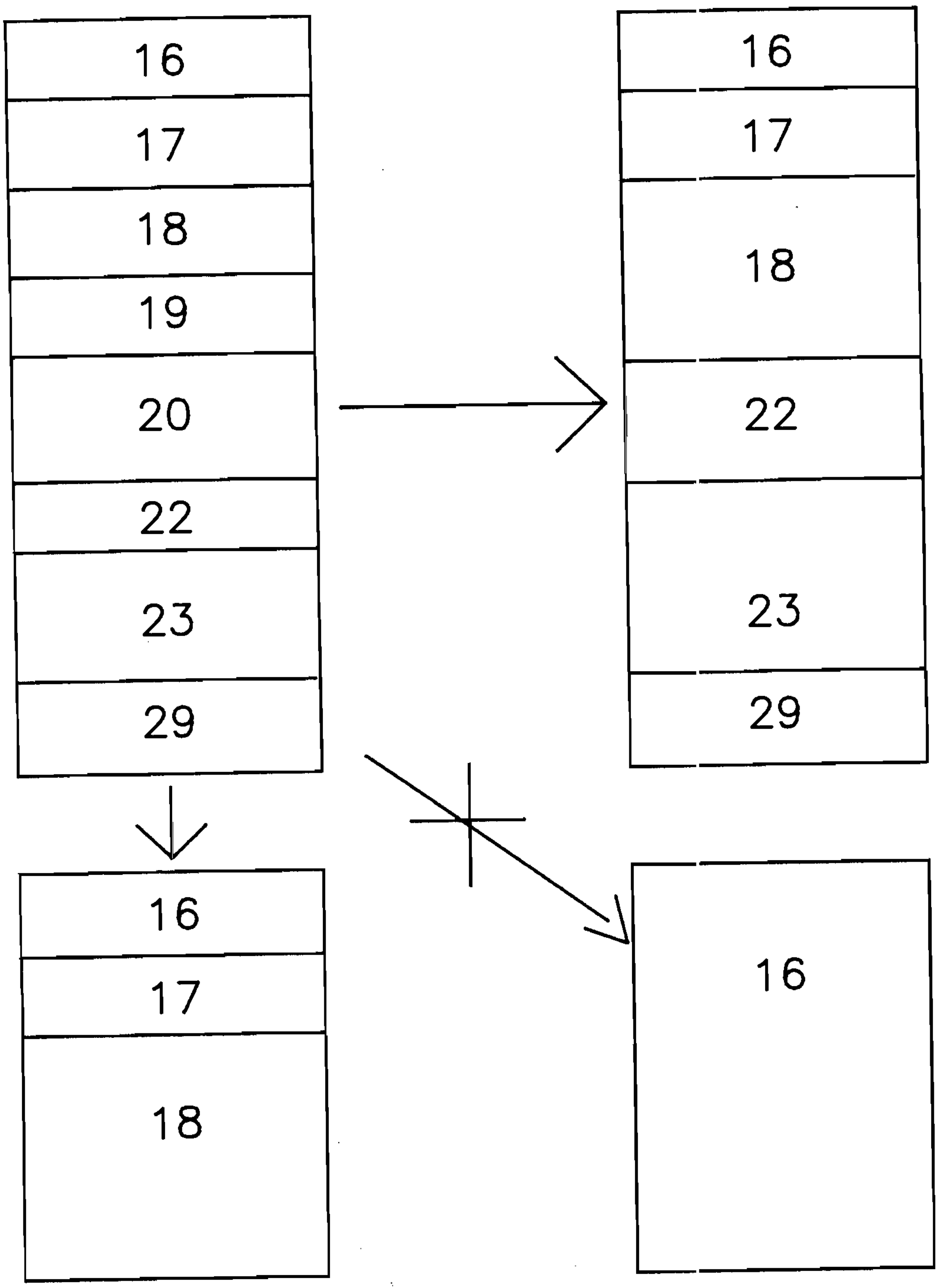
Advantages of smaller LUs:

- could restrict users to specific LUs.
- faster physical backup of individual LU.
- smaller amount of disc space can be allocated to required FMGR LUs.

It is NOT possible to create only one LU on your disc from another configuration if you only have one bootable disc. When you change the boundaries of a disc LU in a generation, all data on that LU will be lost when the system is booted!

References: System Generation and Installation Manual

FILE SYSTEM CONSIDERATIONS



14.18 Disc Allocation Units

The bit map is an area of disc space which has one bit for each allocation unit on the disc. When a user requests a file to be created, the bit map is searched by the file system for a contiguous group of free allocation units sufficiently large to satisfy the request.

Since the bit map has a fixed size allowing 128K allocation units per disc volume (LU), the size of the allocation unit is a function of the size of the LU. The size of the disc allocation unit is always a power of 2, i.e., 1, 2, 4, 8, 16, etc.

For example: The 7933 disc is 404 Mbytes or about 1540 K blocks. If the 7933 were configured to be one disc volume:

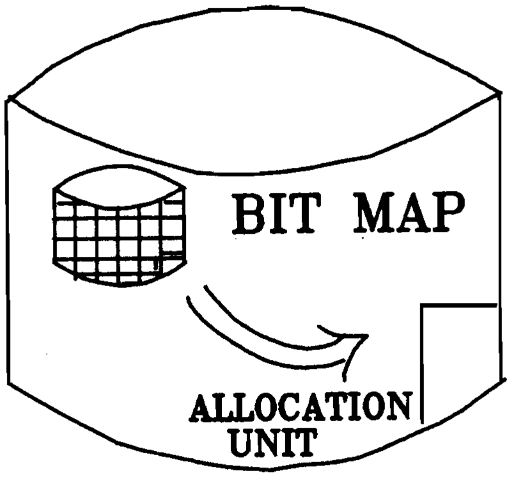
1540 K blocks / (128 K allocation units) = 12 blks/allocation unit

Rounded to the next power of 2, the disc allocation unit would be 16 blocks. Thus disc space would always be allocated in multiples of 16 blocks. If there were a large number of very small files, some disc space would be wasted. Thus, depending on the application, it may be desirable to configure the disc into several volumes.

of disc unit

References: System Design Manual

DISC ALLOCATION UNITS



BIT MAP = $\dfrac{\text{128K bits}}{\text{1 bit per allocation unit}}$

ALLOCATION UNIT = smallest amount of disc space allocated at one time.

BLOCK = 256 bytes

VOLUME SIZE	**ALLOCATION UNIT**
up to 128K blocks (33.5 Mbytes)	**1 blocks** (256 bytes)
up to 256K blocks (67 Mbytes)	**2 blocks** (512 bytes)
up to 512K blocks (134 Mbytes)	**4 blocks** (1024 bytes)

14.19 Disc Summary

The discs listed are all functionally compatible with the A-series, although not all are qualified for use on an A-series system. You should consult a current configuration guide to determine which discs are available and supported for use on an RTE-A system. Following is a summary of the capacity of the discs. This is only for your reference during class. For ordering or configuration consult the appropriate configuration guide or disc manual.

Disc	capacity
7908	16.5 Mbyte
7911	28.1 Mbyte
7912	65.6 Mbyte
7914	132.1 Mbyte
7933	404 Mbyte
7935	404 Mbyte
2480	10 Mbyte hard disc + 270 Kbyte microfloppy
5 1/4 or 3 1/2 inch floppy (9121, 9133)	\ 270 Kbyte (single) / 540 Kbyte (dual)
8 inch floppy (9895)	1.15 Mbyte (single) 2.3 Mbyte (dual)
Winchester	
9133A,9134A	5 Mbyte
9133B,9134B	10 Mbyte
7906H	19.6 Mbyte + 9.8 Mbyte removable
7920H	50 Mbyte + 50 Mbyte removable
7925H	120 Mbyte + 120 Mbyte removable

References: HP 1000 Computer Systems Peripheral Selection Guide

DISC SUMMARY

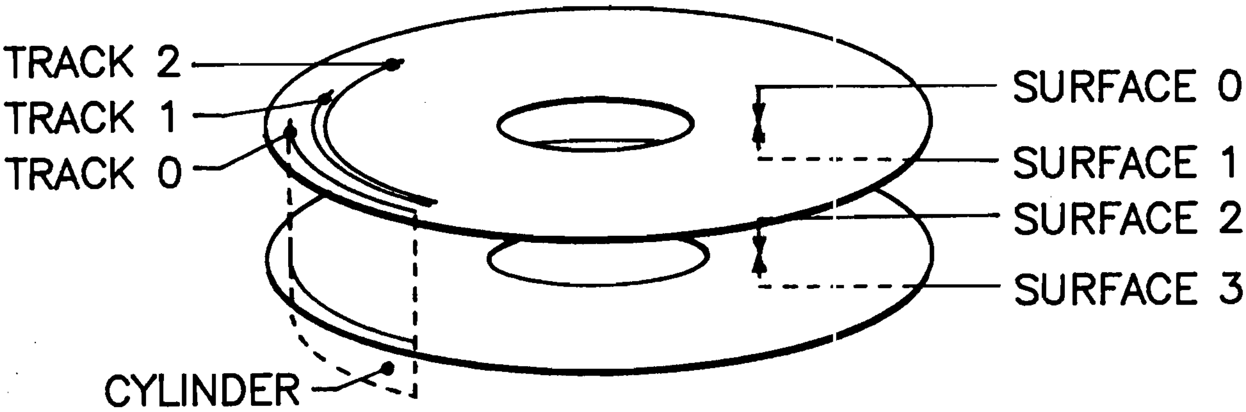
MODEL#	TYPE	NOTES
7908	CS80	integrated CTD
7911	CS80	integrated CTD
7912	CS80	integrated CTD
7914	CS80	integrated CTD
7933	CS80	
7935	CS80	
2480	integrated disc w/microfloppy	
9121	dual/single microfloppy	
9133	mini-winchester w/microfloppy	
9134	mini-winchester	not configurable by user
9895	dual/single floppy	
7906H	ICD	not qualified
7910	ICD	obsolete, not qualified
7920H	ICD	not qualified
7925H	ICD	not qualified

14.20 Physical Disc Structure

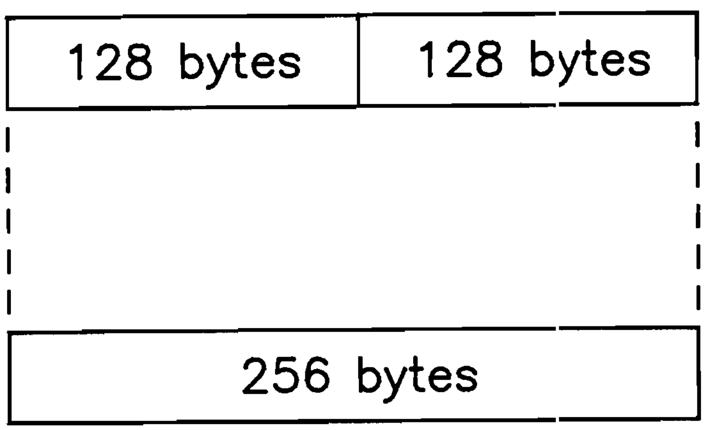
A logical disc sector is the unit addressed by the EXEC call to the driver. A physical disc block is the smallest physical unit on the disc which can be addressed. Because a physical block is equal to 2 logical sectors, only even-numbered sectors can be addressed.

References: System Generation and Installation Manual

PHYSICAL DISC STRUCTURE



FILE SYSTEM LOGICAL DISC SECTORS



PHYSICAL DISC BLOCKS

14.21 Surface Mode

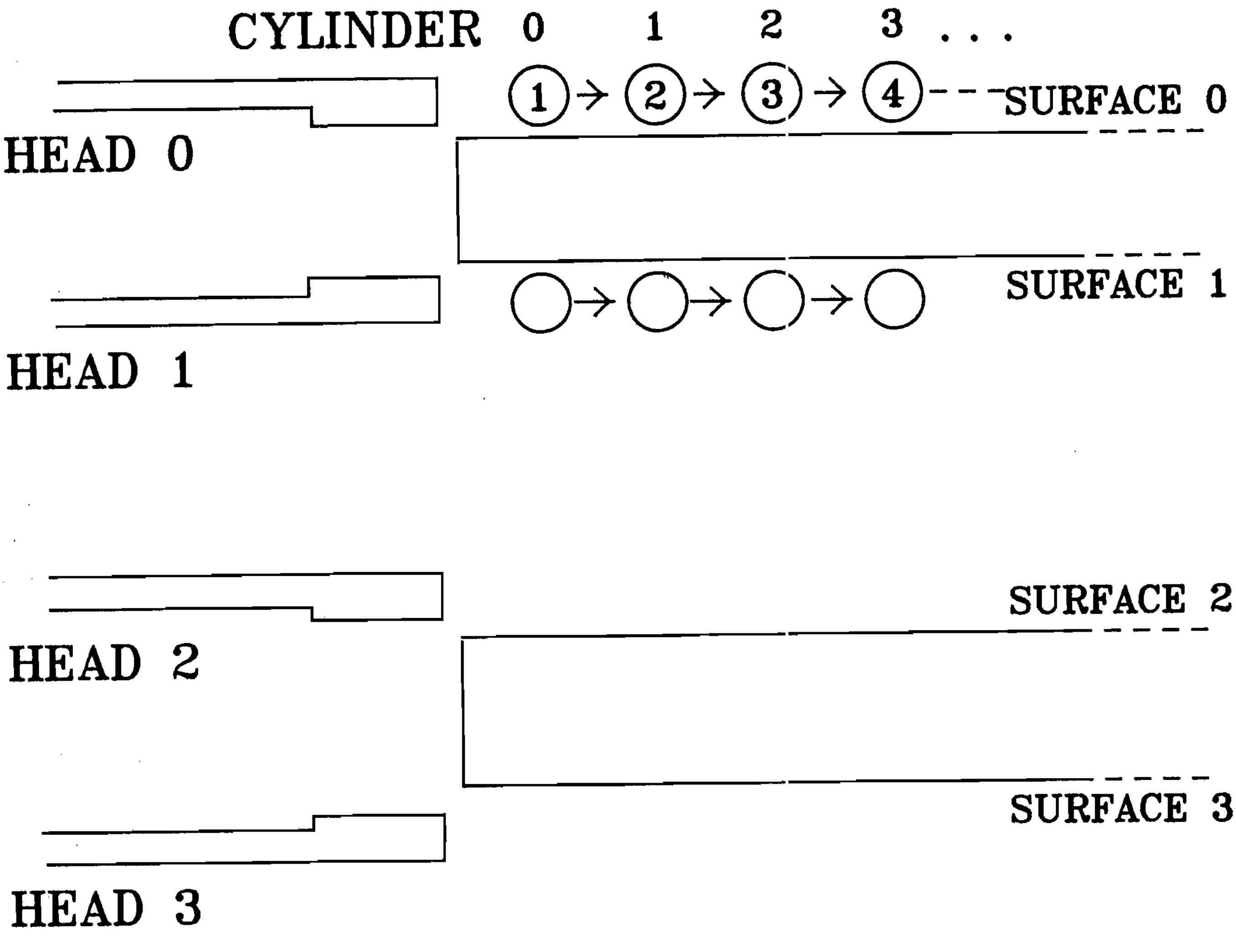
Surface mode configuration is applicable only to non-CS80 discs.

A track is an area containing a number of contiguous disc blocks. On non-CS80 discs, a track is the area contained in one cylinder, on one surface.

In surface mode, each disc LU is made up of tracks that are all on one disc surface, and tracks are accessed in serial order on that surface. An LU may NOT cross a surface boundary.

References: System Generation and Installation Manual

SURFACE MODE



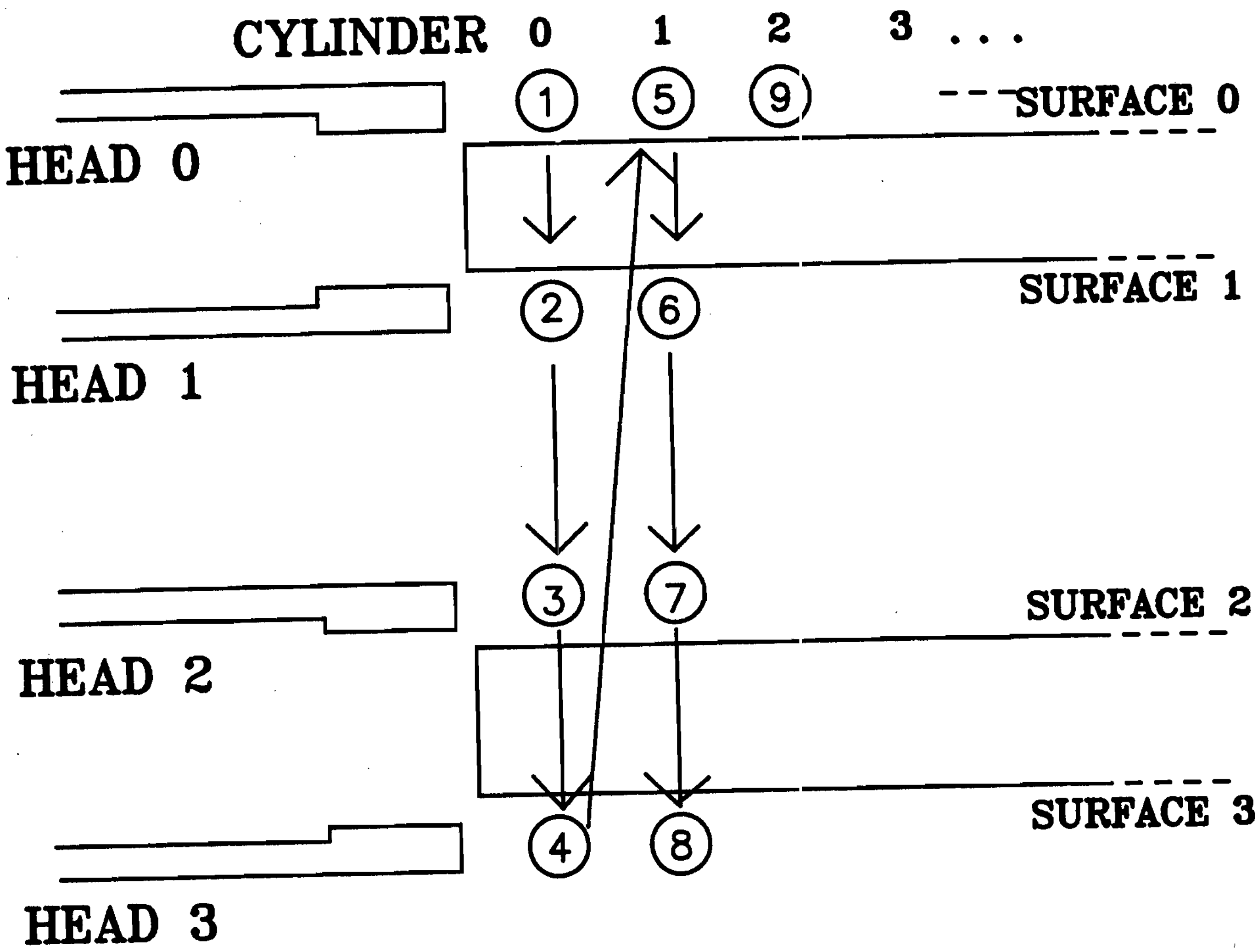
14.22 Cylinder Mode

In cylinder mode, tracks are arranged in groups of cylinders. A disc cylinder includes all tracks with a given track number on all surfaces of the disc.

For non-CS80 discs, all tracks in a given cylinder must be contained within one LU. There is no restriction for CS80 discs.

References: System Generation and Installation Manual

CYLINDER MODE



14.23 CS80 Disc Configuration

The CS80 discs have an area reserved for spares, so no spare tracks are allocated. CS80 discs do not really have tracks - each block is addressed by a 3-word address. The driver maps the track and sector into a block address. Thus, accessing CS80 disc through the driver is similar to accessing a non-CS80 disc.

References: System Gen. & Instl. Manual, disc manuals

CS80 DISC CONFIGURATION

* CYLINDER MODE ONLY
* BLOCK ADDRESSING
* NO SPARES NEEDED
* DISC CACHE FOR CTD

14.24 CS80 Disc Driver Parameters

Configuring a disc into the desired LU involves setting driver parameters in the DVT for each LU. The device driver for CS80 discs and CTDs is DD.33.

Unit/Volume number - The disc driver unit number (upper 8 bits) is a number that indentifies the drive to the disc controller. For currently supported CS80 discs this is always zero. On currently supported CS80 discs, the volume number (lower 8 bits) is always zero.

Starting block address - This is a three-word block address and can be anywhere on the disc as long as the LUs do not overlap.

Number of blocks per track for LU - This defines the "track" on the CS80 disc.

Size of LU in blocks =

number tracks for LU * number blocks per track

References: System Gen. & Instl Manual, disc manuals

CS80 DISC DRIVER PARAMETERS

DP 1	HP-IB ADDRESS	
DP 2	DISC DRIVER UNIT, VOLUME NUMBERS	
DP 3	MS WORD	$\left.\begin{matrix}\\ \\ \\ \end{matrix}\right\}$ STARTING BLOCK NUMBER (3 WORDS)
DP 4		
DP 5	LS WORD	
DP 6	NUMBER OF TRACKS FOR LU	
DP 7	NUMBER OF BLOCKS/TRACK FOR LU	
DP 8	0 (RESERVED)	

14.25 CS80 CTD Driver Parameters

The disc cache must be 256 blocks. The driver depends on this number, so any less will not work. Any more than 256 blocks will be wasted.

The CTD unit (upper 8 bits) and volume (lower 8 bits) numbers are 1 and 0, respectively, for currently supported CS80 discs.

The disc cache unit and volume number refers to the disc unit where the cache area is to be located. For currently supported disc these are both zero. The volume number is the lower 8 bits; unit number is the next 7 bits. The C bit (bit 15) must always be set.

References: System Gen. & Instl. Manual, disc manuals

CS80 CTD DRIVER PARAMETERS

DP 1 HPIB address

DP 2 CTD unit, volume number

DP 3 Cbit (bit 15) disc cache
unit, volume #s

DP 4 MS word — starting block address of disc cache (2 words)

DP 5 LS word

DP 6 0 — address of first cache block

DP 7 0

DP 8 0 (reserved)

14.26 Example 7908 Configuration

In this example there is actually 286 blocks allocated for the disc cache. This is just the amount left over after configuring the desired LUs. The extra 30 blocks could not be added to the last LU since it is less than 1 track. This becomes wasted disc space.

References: System Generation and Installation Manual

EXAMPLE 7908 CONFIGURATION

TOTAL BLOCKS: 64750

DISC LU	16	17	18	19	20
DP1 HPIB ADDR	0	0	0	0	0
DP2 UNIT, VOLUME	0	0	0	0	0
DP3 MS } START BLOCK #	0	0	0	0	0
DP4	0	0	0	0	0
DP5 LS	0	19200	29472	43872	54144
DP6 TRACKS	400	214	300	214	215
DP7 BLOCKS/TRACK	48	48	48	48	48
DP8 0	0	0	0	0	0

48 x 400
= 19200

CTD LU	24
DP1 HPIB ADDR	0
DP2 CTD UNIT/VOL	400B
DP3 C, DISC CACHE U/V	100000B
DP4 MS } START BLOCK OF DISC CACHE	0
DP5 LS	64464
DP6 0	0
DP7 0	0
DP8 0	0

START BLK= PREVIOUS START BLK + (TRACKS X BLOCKS/TRACK)

CTD DISC CACHE MUST BE ≥ 256 BLOCKS

R1426H

14.27 Non-CS80 Disc Configuration - Driver Parameters

Configuring a disc into the desired LU involves setting driver parameters in the DVT for each LU in order to define the necessary track map information. The device driver for non-CS80 discs is DD.30

Number of spares for LU - Typically, 2% of each LU is allocated for spares.

Number of tracks for LU - Total number of tracks on all surfaces, not including spares.

Number of blocks per track - Fixed value for each disc. See RTE-A Generation and Installation manual.

Number of surfaces for LU - 1 surface if in surface mode; total surfaces on disc drive if in cylinder mode. This is how the driver tells if the disc is in surface or cylinder mode.

You cannot use both surface and cylinder mode on the same disc drive. Remember, in surface mode, an LU may not cross a surface boundary and in cylinder mode all tracks within a given cylinder must be contained within one LU.

Surface mode is the only mode supported for bootup on the 7906. Cylinder mode is the only mode supported for bootup on the 7920, 7925 and 2480.

eferences: System Gen. & Instl. Manual, disc manuals

NON-CS80 DISC CONFIGURATION DRIVER PARAMETERS

1. HP-IB ADDRESS
2. DISC DRIVE UNIT NUMBER
3. STARTING HEAD FOR LU
4. STARTING CYLINDER FOR LU
5. NUMBER OF SPARES FOR LU
6. NUMBER OF TRACKS FOR LU
7. NUMBER OF BLOCKS PER TRACK FOR DISC
8. NUMBER OF SURFACES FOR LU

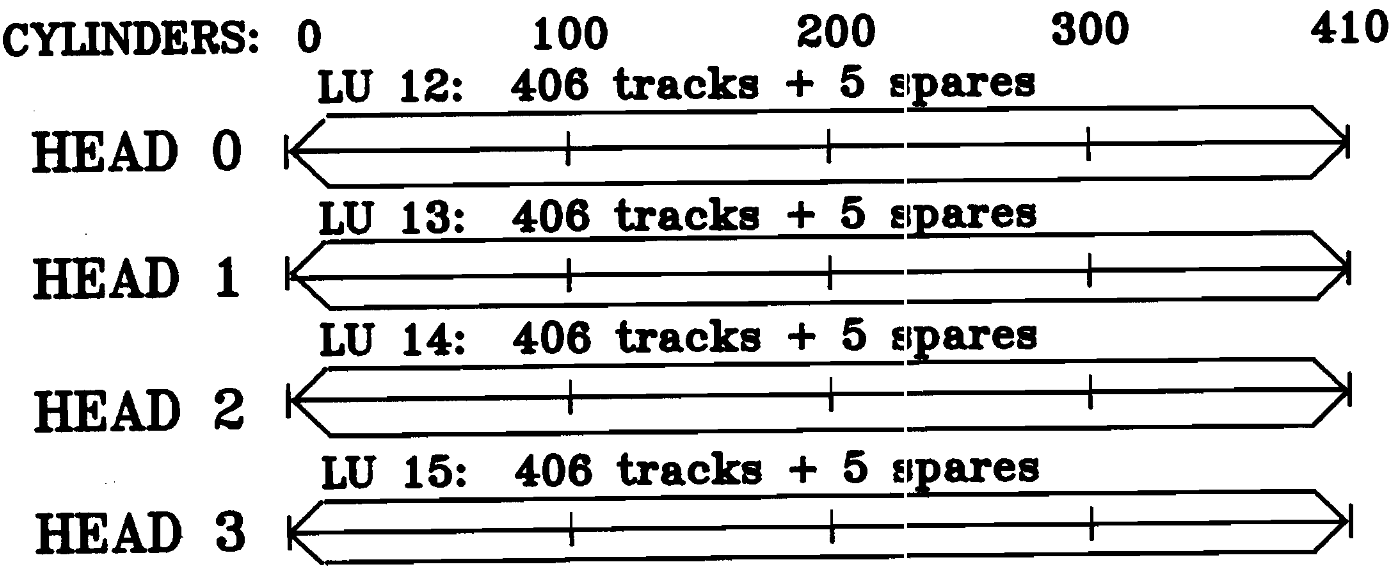
14.28 Example 7906 Configuration

A surface mode configuration is shown. Because the upper platter is removable, surface mode is usually best for the 7906.

References: System Generation and Installation Manual

EXAMPLE 7906 CONFIGURATION

7906 DISC CONFIGURATION WORKSHEET



DISC LU	12	13	14	15
DP1 HPIB ADDR	1	1	1	1
DP2 UNIT NUMBER	0	0	0	0
DP3 START HEAD	0	1	2	3
DP4 START CYLINDER	0	0	0	0
DP5 SPARES	5	5	5	5
DP6 TRACKS	406	406	406	406
DP7 BLOCKS/TRACK	48	48	48	48
DP8 SURFACES	1	1	1	1

14.29 Example 7925 Configuration

A cylinder mode configuration is shown.

References: System Generation and Installation Manual

EXAMPLE 7925 CONFIGURATION

DISC LU	10	11	12	13	14	15	16	17	18
DP1 HPIB ADDR	2	2	2	2	2	2	2	2	2
DP2 UNIT NUMBER	0	0	0	0	0	0	0	0	0
DP3 START HEAD	0	0	0	0	0	0	0	0	0
DP4 START CYLINDER	0	62	124	186	248	363	478	593	708
DP5 SPARES	9	9	9	9	11	11	11	11	11
DP6 TRACKS	549	549	549	549	1024	1024	1024	1024	1024
DP7 BLOCKS/ TRACK	64	64	64	64	64	64	64	64	64
DP8 SURFACES	9	9	9	9	9	9	9	9	9

R1429H

14.30 Example Floppy Disc Configuration

This example shows the configuration of a flexible mini-disc (5 1/4 inch). Configuration of 8 inch and 3 1/2 inch floppies is essentially the same. Floppies can only be configured in cylinder mode. Because of their size and because they are removable, it does not make sense to define more than one LU per disc.

Floppy discs have an area reserved for spares. No spares are allocated in the driver parameters.

References: System Gen. & Instl. Manual, disc manuals

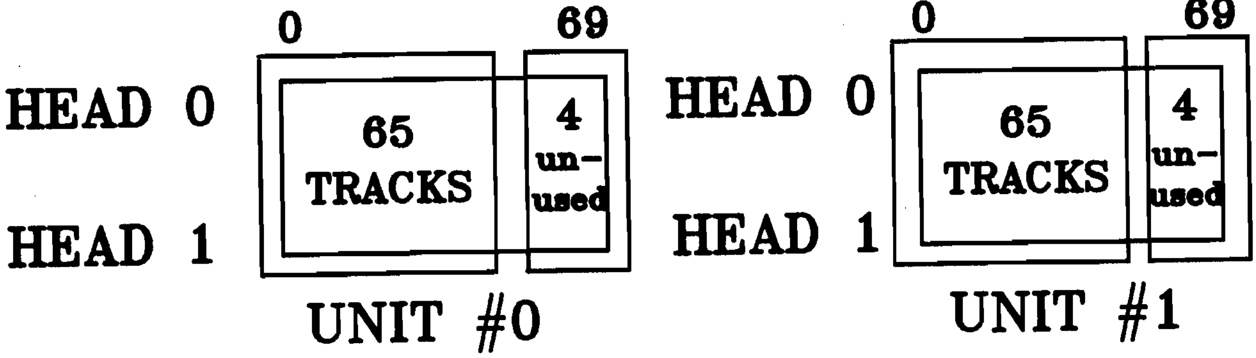
EXAMPLE FLOPPY DISC CONFIGURATION (9895)

CYLINDER MODE ONLY

DISC CONFIGURATION WORKSHEET

FLEXIBLE MINI

CYLINDERS:



TOTAL TRACKS: 140

DISC LU	20	21
DP1 HPIB ADDR	2	2
DP2 UNIT NUMBER	0	1
DP3 START HEAD	0	0
DP4 START CYLINDER	0	0
DP5 SPARES	0	0
DP6 TRACKS	65	65
DP7 BLOCKS/ TRACK	16	16
DP8 SURFACES	2	2

SYSTEM GENERATION

CHAPTER 15

Table of Contents

MODULE OBJECTIVES

1. Be able to run the RTAGN program.

2. Be able to prepare a generation answer file - what are the tradeoffs, how to choose gen parameters, what parameters can be adjusted later.

3. Be able to add I/O devices to the generation answer file.

SELF-EVALUATION QUESTIONS

15-1. What are the outputs of the RTAGN program?

15-2. What are the five phases of the generation?

15-3. Which initialization command causes links to be put on the system base page?

15-4. What do the following commands do?

RE
SE
ALIGN

15-5. What tables are defined during the table generation phase?

15-6. Which LUs must be assigned to a node list?

15-7. List four other tables for which space is allocated at generation time.

15-8. Why are so many END statements required in the generation answer file?

15.1 Review of Physical Memory

The system generation basically builds an image of physical memory for your system.

References: System Design Manual

REVIEW OF PHYSICAL MEMORY

HIGHEST ADDRESS

DYNAMIC PARTITION AREA
RESERVED USER PARTITION #n
• • •
RESERVED USER PARTITION #1
SYSTEM AVAILABLE MEMORY (SAM)
SYSTEM MESSAGE BLOCK
SYSTEM COMMON
SYSTEM TABLES
DRIVER PARTITION #n
• • •
DRIVER PARTITION #1
SYSTEM MODULES
SYSTEM BASE PAGE

↑ user area

↓ system area

ADDRESS 0

15.2 The System Generation Process

RTAGN - The RTE-A generator program uses the system relocatables and a user-prepared answer file to create the list, system and snap files.

List file - Provides documentation of what is in the system and where the modules are located. Also, the list file indicates where any errors occurred and describes them.

System file - Type 1 file that contains a memory image of the operating system.

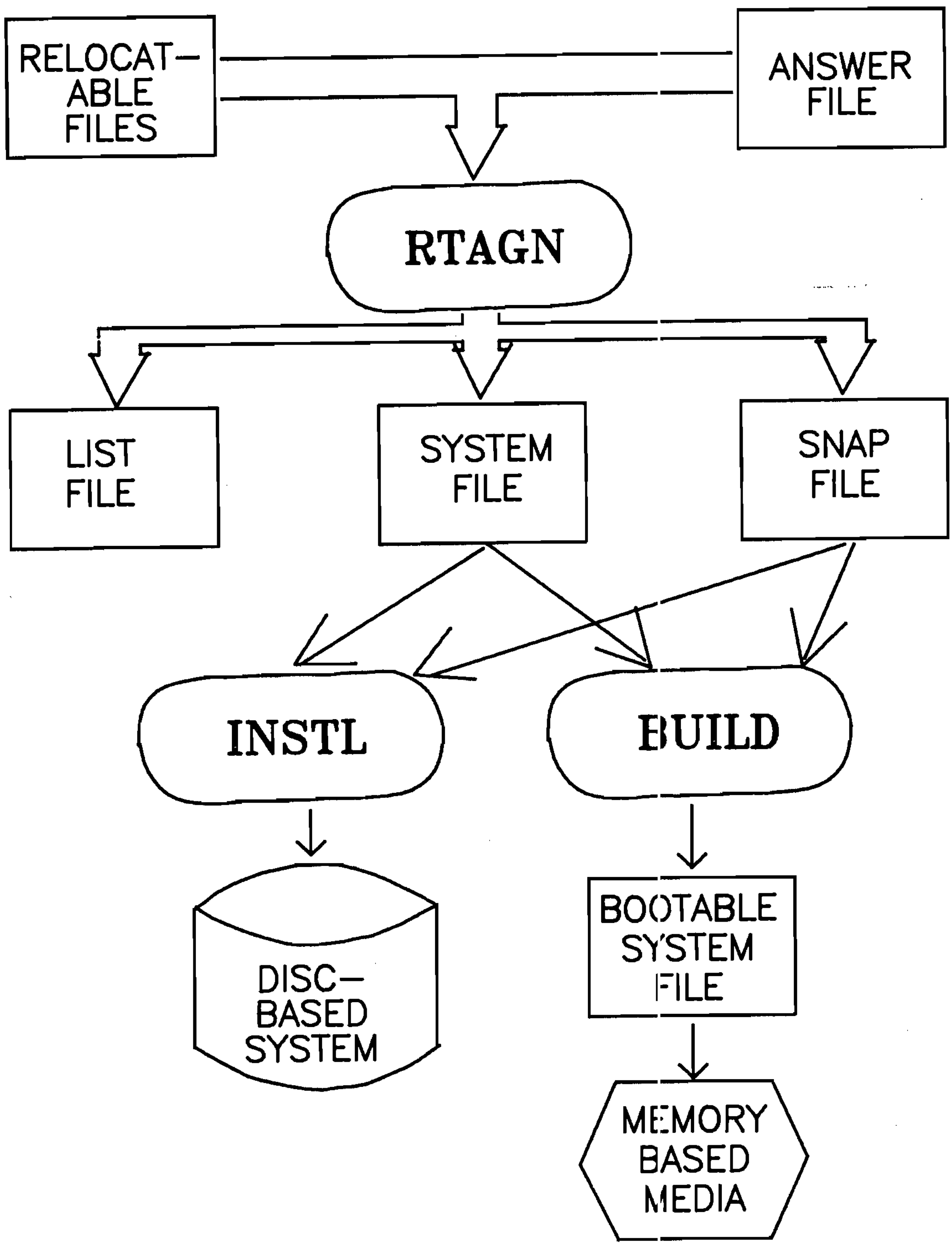
Snap file - The snapshot file contains the value of system entry points, system library names and other system information such as system checksums and system common checksum. This is used by the loader to load programs on-line.

INSTL - Installs a Boot extension for a disc-based system.

BUILD - Merges the system file with program files to create a type 1 bootable system file.

References: System Generation and Installation Manual

THE SYSTEM GENERATION PROCESS



15.3 RTAGN Generator Program

Answer file - (or command file) contains commands used by the generator to generate the operating system.

List file - output by generator; shows commands, comments, module bounds, entry points and generation errors. Error messages are indicated by:

** error **

System file - type 1 file that contains a memory image of the operating system.

Snap file -- The snapshot file contains the value of system entry points, system library names and other system information such as system checksums and system common checksum. This is used by the loader to load programs on-line.

Initialization phase - define whether current page or base page linking is used.

System relocation phase - relocate system modules and non-partitioned and privileged drivers.

Driver partition phase - relocate drivers in partitions.

Table generation phase - Define LUs for I/O devices, set up required I/O tables.

Memory allocation phase - define various system tables and other memory to be allocated.

References: System Generation and Installation Manual

RTAGN generator program

.ANS .LST .SYS .SNP

CI> rtagn <answer> <list> <system> <snap>

5 PHASES

INITIALIZATION
SYSTEM RELOCATION
DRIVER PARTITION
TABLE GENERATION
MEMORY ALLOCATION

15.4 INITIALIZATION PHASE

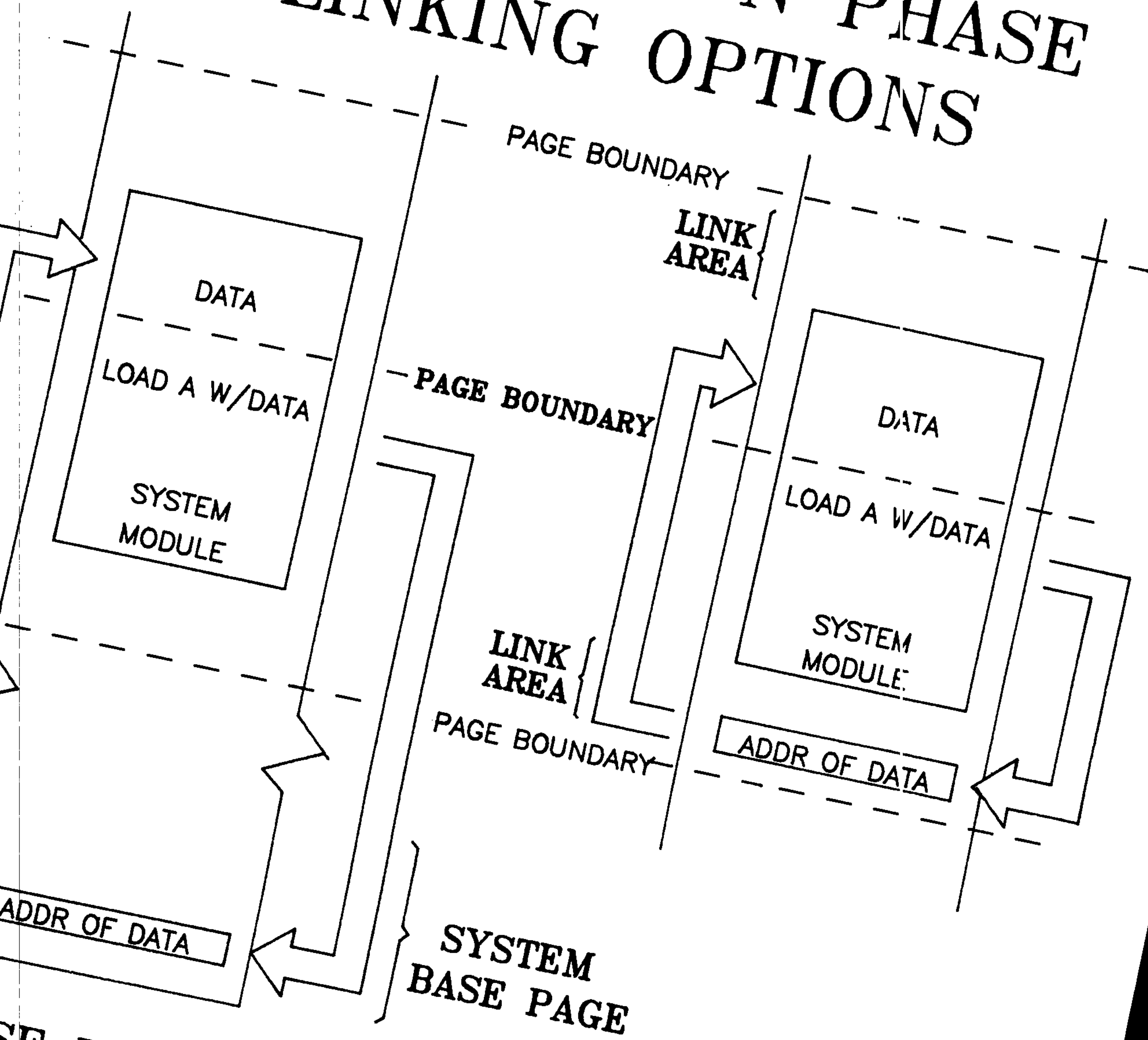
A link word is created by the generator whenever a one-word memory reference instruction references a location which is not on the same page as the instruction. This is needed because the memory reference instruction only allows ten bits for the address. An additional bit indicates whether the address is on the current or base page. ??

If <u>current page linking</u> is used, there may still be some links generated on the base page.

Using <u>base page linking</u> will reduce the overall system size, but there is a fixed amount of space on the base page available for links.

ences: System Gen. & Instl. Manual, System Design Manual

INITIALIZATION PHASE LINKING OPTIONS



SE PAGE
INKS

nds:

S, BP

CURRENT PAGE LINKS

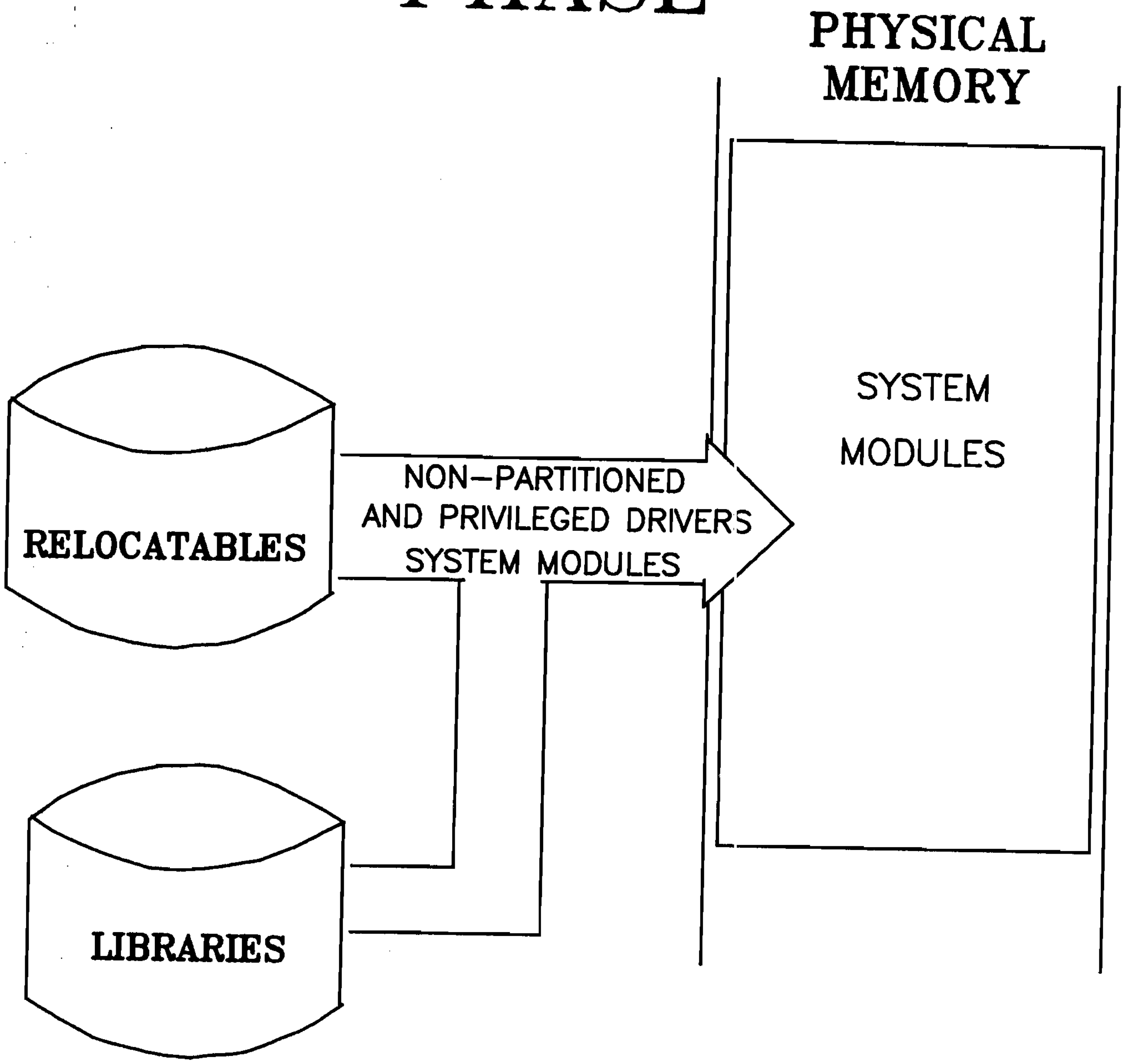
LINKS, CP

15.5 SYSTEM RELOCATION PHASE

During this phase, system modules, non-partitioned and privileged drivers are relocated. System libraries are searched to satisfy external references.

References: System Generation and Installation Manual

SYSTEM RELOCATION PHASE



15.6 System Relocation Phase Commands

These commands are used in the system relocation phase. They can also be used during other parts of the generation as indicated elsewhere. To specify the command, you can give the entire command or just the first two letters.

- REl ocate - Relocate a module as part of the op system.
- SEarch - Search a library to resolve external references.
- MSearch - Search a library multiple times.
- LOcc - Set relocation address.
- BLocc - Set base page relocation address (use to reserve an area on the base page).
- DIsplay= - Display undefined externals.
- LEntries - Turn on or off listing of module entry points.
- ALign - Set relocation address to next page boundary (typically only used in driver partition phase).
- ENd - Indicates end of phase.

References: System Generation and Installation Manual

SYSTEM RELOCATION PHASE COMMANDS

RELOCATE, file [, module name]

SEARCH, file [, module name]

MSEARCH, file

LOCC, address

BLOCC, address

DISPLAY

LENTRIES [, ON/OFF]

ALIGN

END

15.7 System Relocation Example

%VCTR must be relocated first for program transportability. The other modules may be relocated in any order. The RPL file used depends on your hardware and options - see manual. Other required system modules must be relocated in this phase. $SYSA should always be searched in case a module was left out. $SYSLB contains routines used by system modules.

%SPOOL and %CDSFH are provided only with the VC+ (92078) option.

References: System Gen. & Instl. Manual, primary answer file

SYSTEM RELOCATION PHASE EXAMPLE

```
*
*
RE, %VCTR
RE, %SPOOL
RE, %EXEC
RE, %MEMRY
RE, %CDSFH
RE, %RPL73
RE, %SAM
RE, %TIME
RE, %SCHED
RE, %STRNG
RE, %LOCK
RE, %ERLOG
RE, %OPMSG
RE, %XCMND
RE, %SYCOM
RE, %STAT
RE, %LOAD
RE, %RTIOA
RE, %IOMOD
RE, %PERR
RE, %CLASS
RE, %ID.43
*
MS, $SYSA
SE, $SYSLB
*
END
*
*
```

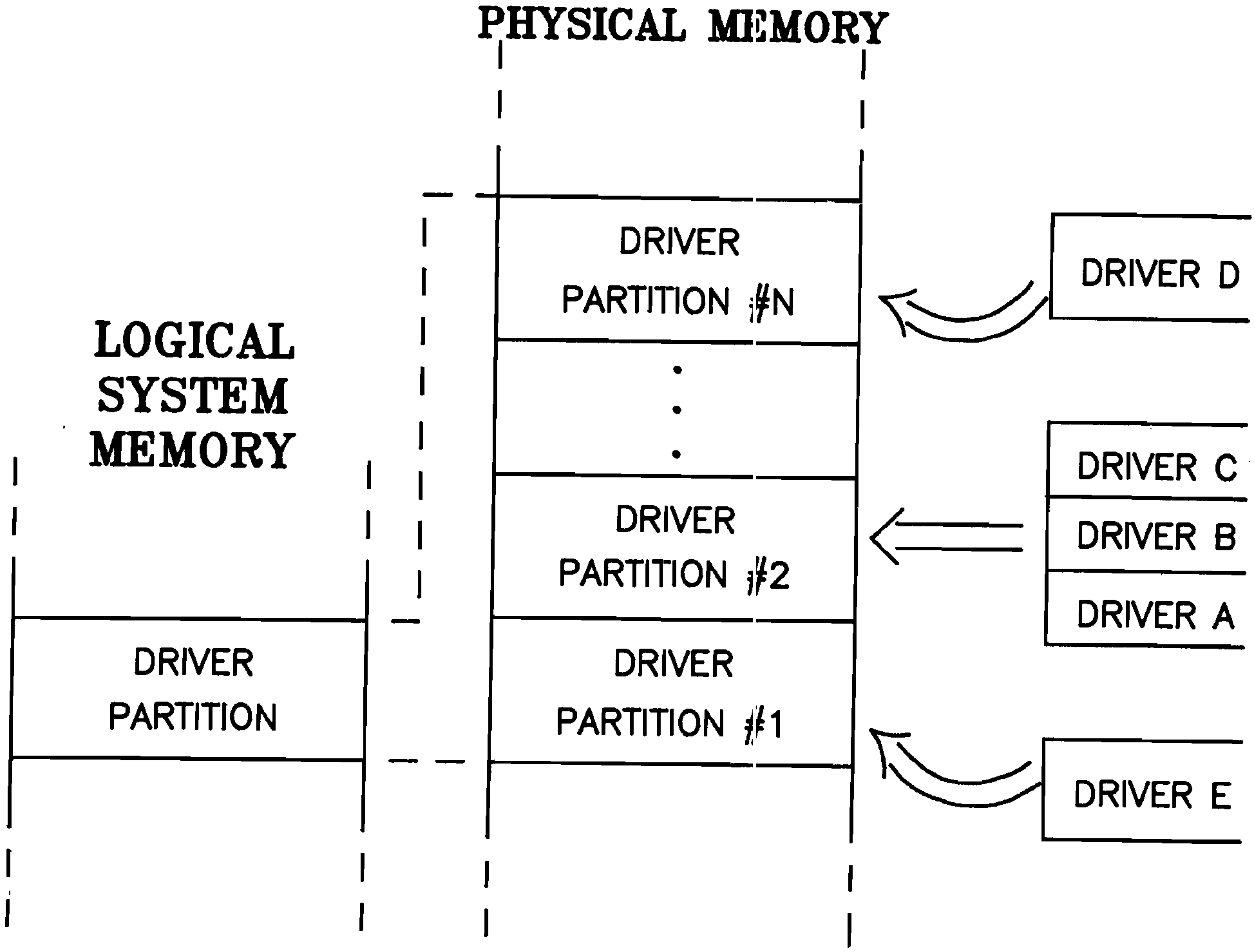
this is always the 1st module to be relocated.

15.8 DRIVER PARTITION PHASE

Any number of drivers can be relocated into one driver partition as long as it does not exceed 5 pages. The size of the driver partition in logical system memory is the size of the largest driver partition created. In general it is recommended that you relocate only one driver per partition in order to keep the size of the system driver partition small.

References: System Generation and Installation Manual

DRIVER PARTITION PHASE



COMMANDS:

same as the System Relocation Phase

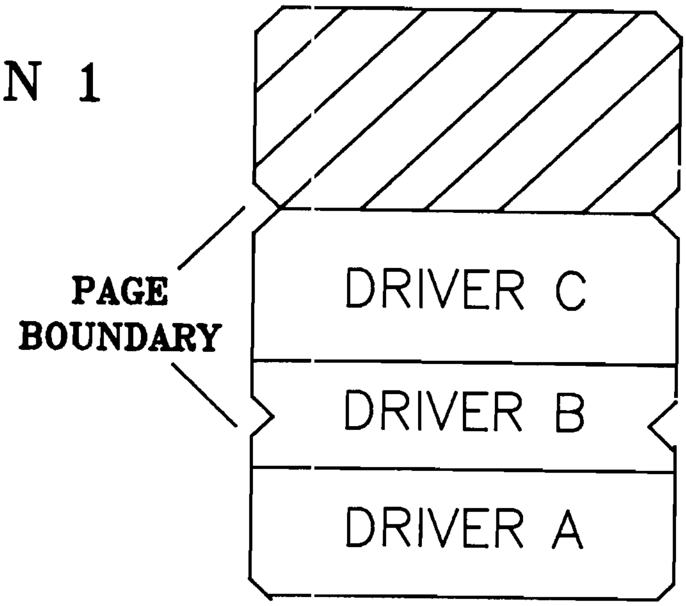
15.9 Driver Page Alignment Example

The ALIGN command is useful in this phase. Partition 1 would create many more links (potentially base page links) than Partition 2, even though it uses more memory.

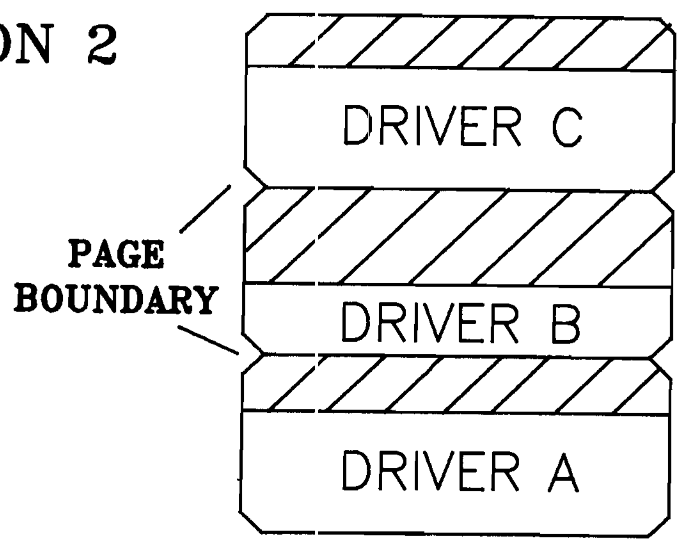
References: System Generation and Installation Manual

DRIVER PAGE ALIGNMENT EXAMPLE

```
*
*DRIVER PARTITION 1
*
RE, A
RE, B
RE, C
END
```




```
*
*DRIVER PARTITION 2
*
RE, A
ALIGN
RE, B
ALIGN
RE,C
END
*
```



15.10 Driver Partition Phase Example

Each driver partition is terminated with an END command. The driver partition phase is terminated with another END.

In this example, the system driver partition is at least as large as the largest driver (%DDC12). Therefore it makes sense to fill up the other partitions with more than one driver until they are about the same size. The System Generation and Installation Manual has a table of approximate driver sizes.

References: System Gen. & Instl. Manual, primary answer file

DRIVER PARTITION PHASE EXAMPLE

```
*
* Driver Partition Phase
*
RE, %DD.33::MS
RE, %ID.52::MS
END
*
RE, %ID.66::MS
RE, %ID.00::MS
END
*
RE, %ID.37::MS
RE, %DD.30::MS
END
*
RE, %IDM00::MS
RE, %DD.23::MS
END
*
RE, %DD.00::MS
ALIGN
RE, %ID.27::MS
RE, %ID.50::MS
END
*
RE, %DD.12::MS
RE, %ADV00::A2
RE, %DD.20::MS
END
*
RE, %DDC12::MS
END
*
*   end driver partition
END
```

15.11 TABLE GENERATION PHASE

IFT - Interface table

DVT - Device table

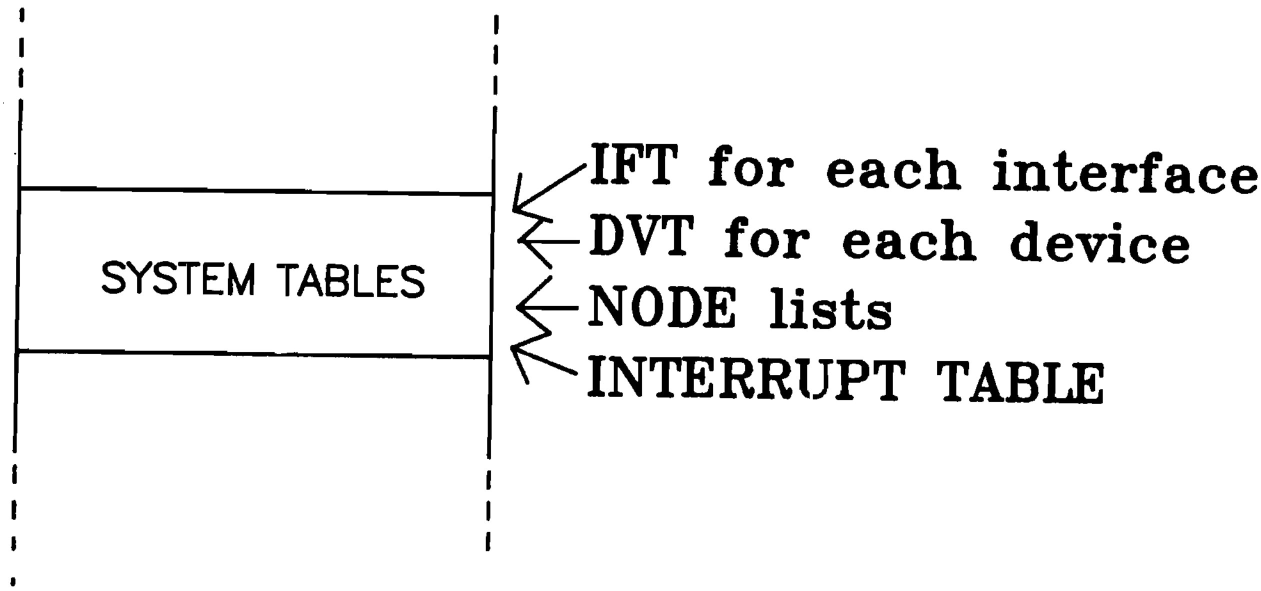
INT - Interrupt table

The IFT/DVT creation is terminated by two END statements - one to end the IFT creation and one to end the DVT creation. The node list and INT table part are each terminated with an END.

Both the DVT and NODE commands allow the use of a dash (-) to indicate that the command is continued on the next line. The dash is typed immediately following the comma separator between parameters, but never in the middle of a parameter.

References: System Generation and Installation Manual

TABLE GENERATION PHASE



COMMANDS:

```
IFT,...
DVT,...
   .
   .
IFT,...
DVT,...
DVT,...
   .
END
END
*
NODE,...

   :
END
*
INT
.
.
END
```

15.12 IFT Command

Default File - File which contains default IFT entries for the interface.

Entry Point - Entry point of the corresponding interface driver.

Select Code - Between 20B and 47B.

Queuing - FI (first in, first out) or PR (priority) queuing of devices on IFT.

Table Extension - The IFT extension area is used for temporary storage by the driver. If this area is not made large enough, the system will not operate properly! The requirements are defined by the driver - see the System Generation and Installation Manual.

Interface Type - A value which defines what type of interface card the IFT references (i.e., HP-IB, asynchronous, etc.) Used by some utility programs to determine what type of interface the LU uses.

None of these parameters can be altered on line.

Some of these parameters do not need to be specified. The generator will use default values for QU, TX and IT if none are specified. If no entry point is specified, then the generator will use the NAM record of the default file for the entry point.

References: System Generation and Installation Manual

IFT COMMAND

IFT, default file,

E entry point,

SC: select code,

QU: queuing,

TX: table extension,

IT: interface type

15.13 DVT Command

Default file - File containing default DVT entries for device.

Model Number - Model number of device, used for determining what values from the default file will be used.

LU - Between 1 and 255, between 1 and 63 for discs.

Entry Point - Entry point of device driver. If none is specified then requests will go directly to the interface driver.

Timeout - Timeout for device in 10's of milliseconds (0-255).

Buffering - Indicates whether a device is buffered (BU) or not (UN) and size of buffer limits (in multiples of 16).

Device type - Value which indicates what class of device the DVT points to (i.e., printer, disc, etc.) Used by some utility programs to determine if and how to talk to the LU. (0-77 octal).

Table extension and Parameter area - areas used by the driver for data storage; requirements are defined by driver. See reference manual for the size needed. If incorrect values are used, system will not operate correctly! Table extension is 0-511, driver parameter area is 0-127.

DP - Specifies parameters to be entered at generation time, when applicable for the specific driver. The first parameter specifies which DVT parameter to start entering; subsequent parameters are values to be entered. See Generation and Installation manual and/or the Driver Reference Manual for requirements.

Queuing - FI (first in, first out) or PR (priority) queuing of requests on DVT. Programs with priority of 40 or less will use priority queuing anyway.

Priority - priority (between 0 and 63) of device on IFT

TO can be changed on-line with CI. BU can be changed via FMGR.

A dash (-) indicates a continuation to the next line. It can only be used following a parameter and all its subparameters.

Some parameters do not need to be specified. The generator will use default values for TO, BU, DT, TX, DX, QU and PR if none are specified. If model number, entry point or driver parameters are not specified, none will be used.

References: System Generation and Installation Manual

DVT COMMAND

DVT, default file,

M model #:subchannel,

LU: lu,

E entry point,

TO: timeout,

BL: buffering: lower limit:
: upper limit,

DT: device type,

TX: table extension,

DX: parameter area,

DP: start pram#:value:value...,

QU: queuing,

PR: priority

– (continuation)

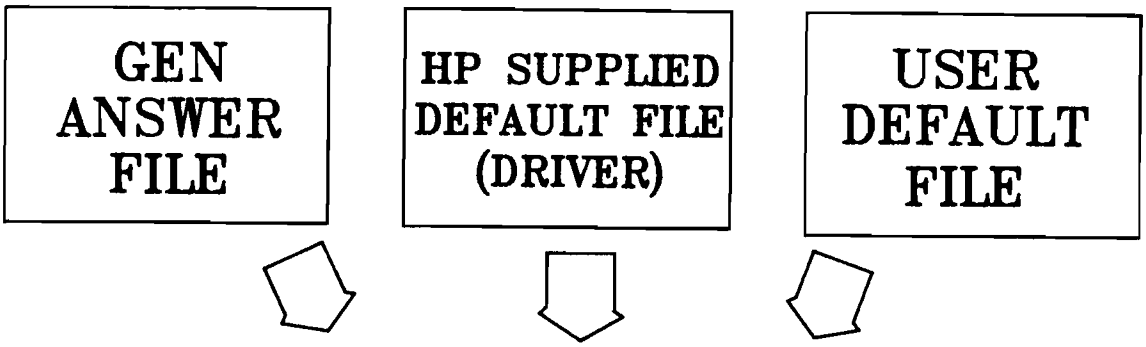
15.14 Default Files

For most drivers the default file is the driver relocatable file. See the reference manual for default file contents.

The user can create a default file with IFT or DVT paramaters using the macroassembler, MACRO/1000.

References: System Generation and Installation Manual

DEFAULT FILES



IFT PARAMETERS
DVT PARAMETERS

FOR MOST INTERFACES YOU NEED ONLY

DEFAULT FILE
SELECT CODE

FOR MOST DEVICES YOU NEED ONLY

DEFAULT FILE NAME
MODEL NUMBER
LU

15.15 IFT/DVT Worksheet

The IFT/DVT worksheet is useful for determining what commands to enter during the table generation phase. In most cases you will not need to enter all the parameters for the IFT or DVT commands. Typically, the values in the default file will be used - indicate this on the worksheet. Sometimes, the generator defaults will be used. For your application, your may want to override particular values in the default files. For example, if you are going to use a disc configuration which is different from that in the default file, you would specify your own DP values in the generator answer file (remember the disc configuration section).

The System Generation and Installation Manual has an appendix which shows the default file values for all devices. It also shows standard entries for the answer file for these devices, when the default files are used. Another appendix describes the driver parameters for disc devices. You may also need to consult the Driver Reference Manual to find what values to enter into the driver parameter area for other devices. In addition, the primary answer file is an excellent reference for examples of table entries for most supported devices. The primary answer file is in an appendix of the System Generation and Installation Manual and is provided on your primary system.

References: System Generation and Installation Manual

IFT/DVT WORKSHEET

INTERFACE NAME :

I/O SLOT #:

IFT, ____________ ,SC: ________ ,E __________ ,QU: ______ ,TX: __________ ,IT: ________

Interface Driver Namr (Default file)	Select Code	Entry Point	Queuing	Table Extension	Interface Type
Device Name: Device Driver: Defaults File: Model Number: Logical Unit:	M LU:	M LU:	M LU:	M LU	M LU
Device Type: Device Priority: Time Out: Buffer Limits: Table Extension: Driver Extent: Driver Prams:	DT: PR: TO: BL: : : TX: DX:	DT: PR: TO: BL: : : TX: DX:	DT: PR: TO: BL: : : TX: DX:	DT: PR: TO: BL: : : TX: DX:	DT: PR: TO: BL: : : TX: DX:
start # 1 2 3 4 5	DP:1	DP:1	DP:1	DP:1	DP:1
start # 6 7 8 9 10	DP:6	DP:6	DP:6	DP:6	DP:6
start # 11 12 13 14 15	DP:11	DP:11	DP:11	DP:11	DP:11
Queuing: Node 1: Node 2: Node 3:	QU:	QU:	QU:	QU:	QU:

15.16 IFT/DVT Examples

Parameters specified in answer file will override those in default file, thus you can specify your own parameters where desired and use the default file values for other parameters. Note that in these examples, the default file is always specified, although sometimes the default parameters are overridden.

The first device is a terminal in the non-VC+ environment. The driver parameters are used to enable CI as the primary program and CM as the secondary program at gen time. (This could also be done online with a CN command).

For the 7908 disc, the default file contains the standard disc configuration. The subchannels with the model number are used to differentiate the areas of the disc for different LUs. DP:1:0 designates an HPIB address of 0 for the disc drive.

The printers are configured using the parameters from the default files. DP:1:2 designates an HPIB address of 2 for the 2608S. DP:1:6 designates an HPIB address of 6 for the 2631B.

Note that for both of the HPIB examples, there are multiple DVTs for one IFT, since there will be multiple devices connected to one HPIB interface card.

References: System Gen. & Instl. Manual, primary answer file

IFT/DVT EXAMPLES

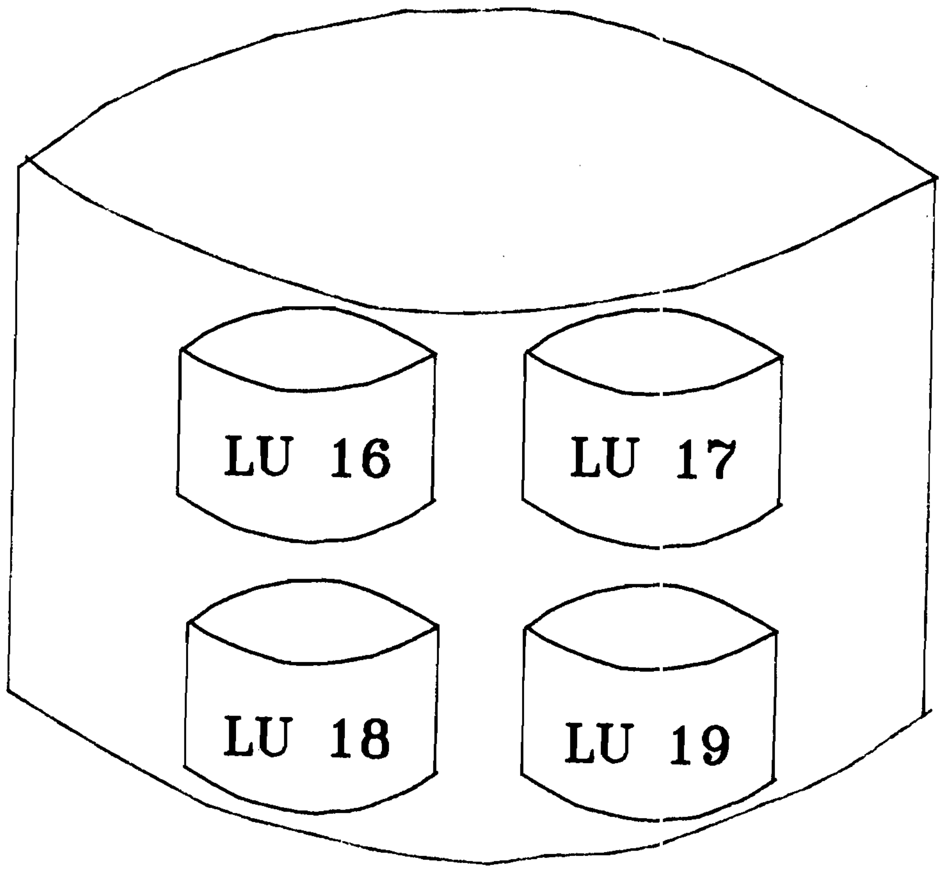
```
*
*  ASIC FOR 26XX SYSTEM CONSOLE
*
IFT,%ID.00, SC:20B
*
DVT,%DD.00,M26XX,LU:1,QU:FI,-
  DP:5:CI:20040B:20040B:0-
  DP:9:CM:20040B:20040B:CM
*
*  HPIB #1 DISC CONTROLLER
*
IFT, %ID.37,SC:27B
*
*  7908 DISC WITH CTD - HPIB ADDR 0
*
DVT, %DD.33,M7908_LF:0,LU:16,DP:1:0
DVT, %DD.33,M7908_LF:1,LU:17,DP:1:0
DVT, %DD.33,M7908_LF:2,LU:18,DP:1:0
DVT, %DD.33,M7908_LF:3,LU:19,DP:1:0
DVT, %DD.33,M7908_LF:4,LU:20,DP:1:0
*
DVT, %DD.33,MTAPE,LU:24,DP:1:0


IFT, %ID.37,SC:25B
*
*  2608S LINEPRINTER HPIB ADDR 2
*
DVT, %DDC12,,LU:85,DP:1:2
*
*  2631B LINEPRINTER HP

DVT, %DD.12,,LU:6,DP:1:6
*
END
END
*
```

15.17 Node Lists

A node list tells the system that when one LU on a node is busy, the others can not be accessed because they use the same controller. There is a place on the IFT/DVT worksheet to indicate nodes. The appendix of the generation manual contains a table of standard IFT/DVT entries for the answer file. This table indicates which devices would go on the same node list. A dash (-) can be used after a comma to continue a node list on the following line.

References: System Generation and Installation Manual

NODE LISTS



A NODE LIST CONTAINS LUs WHICH USE THE SAME PHYSICAL CONTROLLER.

COMMAND:

these LU use same physical controller ∴ only one can be accessed at one time

NODE, lu1, lu2,...,lu n
END

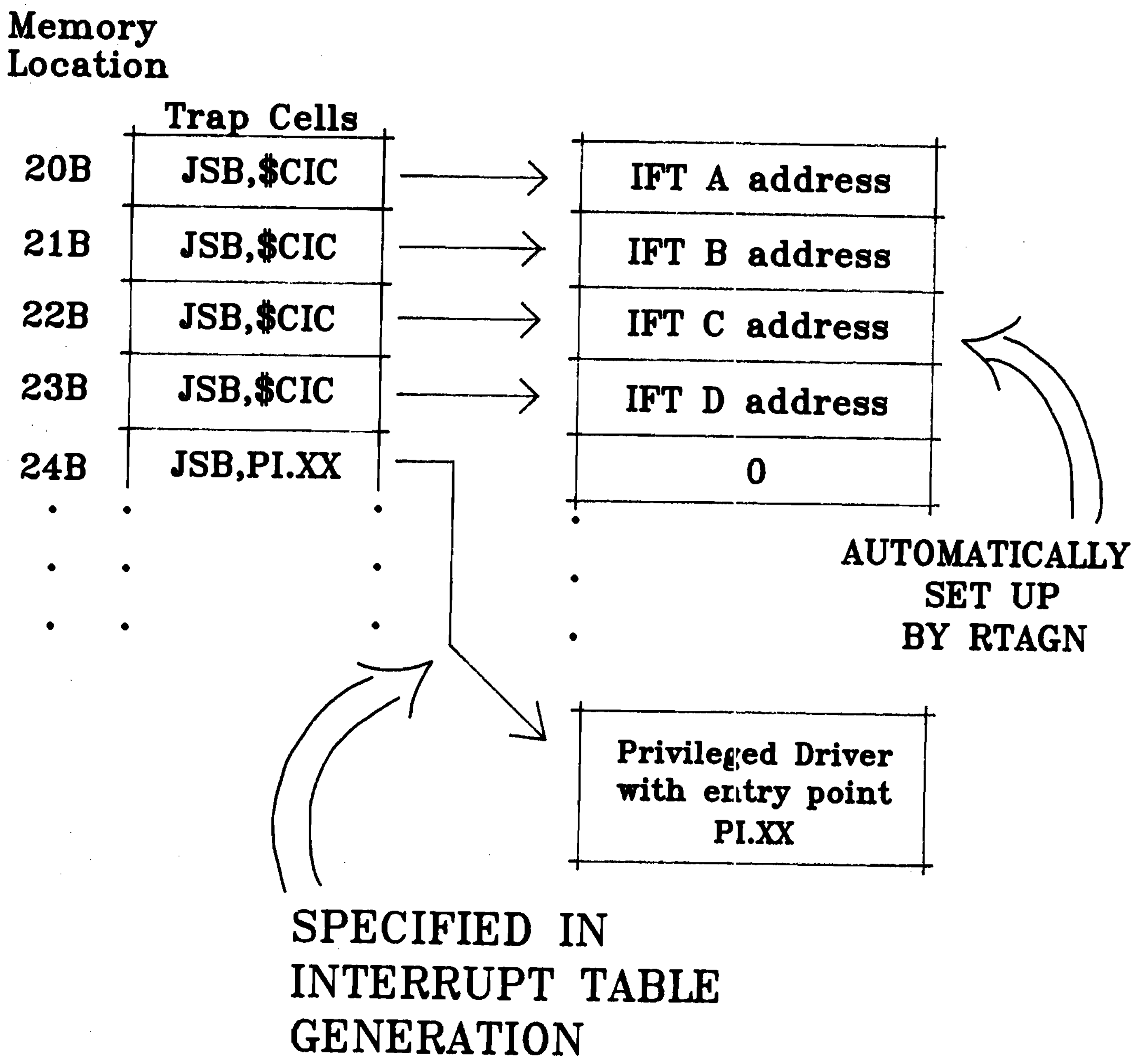
15.18 Interrupt Table

On interrupt, a location in memory called the trap cell, which corresponds to the interrupting select code, will be executed. This location normally contains a JSB to $CIC (central interrupt handler). The interrupt handler will eventually access the corresponding location in the Interrupt table. For a privileged driver, the trap cell contains a JSB to the privileged driver entry point, bypassing the system interrupt handler. This entry point is specified with the INT command.

∴ quicker as it is hardware.

References: System Gen. & Instl. Manual, System Design Manual

INTERRUPT TABLE



COMMANDS:

INT, select code, entry point
END

15.19 Node List and Interrupt Table Example

This example shows most of the devices which will require a node list. If you are unsure as to whether device LUs should go in a node list, consult the generation manual. You could also use the primary answer file for examples.

In this example all the interrupt table entries are generated automatically by the generator.

References: System Gen. & Instl. Manual, primary answer file

NODE LIST AND INTERRUPT TABLE EXAMPLE

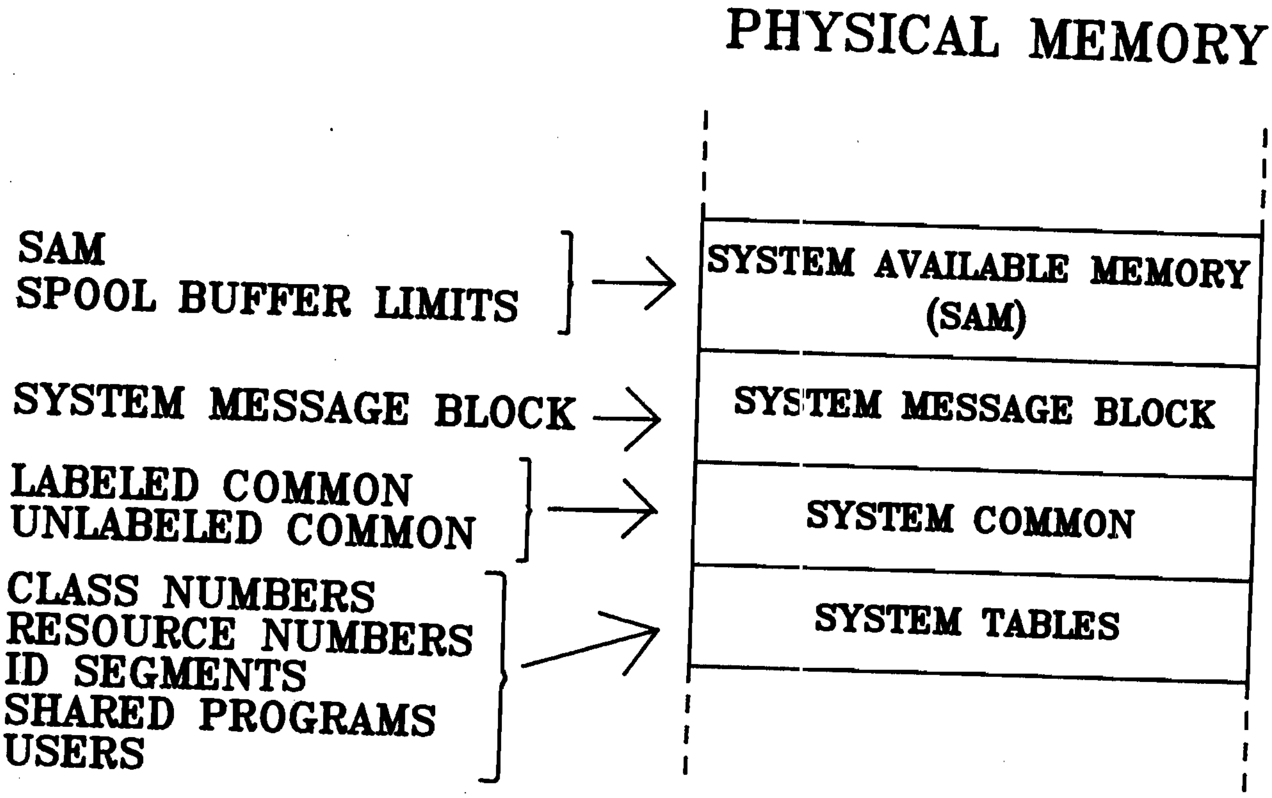
```
*
* DEFINE NODE LIST
*
* 264X SYSTEM CONSOLE WITH TWO TAPE DRIVES
NODE,1,64,65
*
* 2635 AUXILIARY CONSOLE/PRINTER
NODE,66,67
*
* TWO 8" FLEXIBLE DISCS
NODE 10,11
*
* FOUR 7906 LU'S
NODE,12,13,14,15
*
* FOUR 7910 LU'S
NODE,40,41,42,43
*
* THIRTEEN 7908/11/12/14/33/35 AND CTD
NODE,16,17,18,19,20,22,23,24,29,30,31,34,35
*
* TWO 3.5" OR 5.25" FLEXIBLE DISCS
NODE,32,33
*
* FOUR 5.25" FIXED DISC LU'S (9134 FOUR VOL.)
NODE,48,49,50,51
*
* THREE 5.25" FIXED DISC LU'S (9134 A/B SINGLE VOL.)
NODE,52,53,54
*
* FOUR 248X INTEGRATED DISC LU'S
NODE,36,37,38,39
*
END,    NODE LIST
*
END,    INTERRUPT TABLE
*
```

15.20 MEMORY ALLOCATION PHASE

The commands which define system tables will allocate table space. Nothing is put in these tables at generation time.

References: System Generation and Installation Manual

MEMORY ALLOCATION PHASE



ALSO:

BACKGROUND PROGRAM PRIORITY
QUANTUM TIMESLICE VALUE
SYSTEM MEMORY BLOCK (DS 1000-IV)
DEFAULT LIBRARIES

15.21 Memory Allocation Phase Commands

Commands must be given in the order indicated. The generation manual contains detailed information on the commands including suggested formulas for allocating table space and how much memory the tables use.

CLAS, RESN, ID, and RS - these allocate table space and can only be adjusted at gen time.

SAM - the size of SAM can be increased at ~~bootup time~~.

SL - Spool buffer limits can only be set at gen time. This command must always be entered, but for non-VC+ systems enter 0 as the upper and lower limits.

BG - Background program priority limits, QU - Quantum timeslice and timeslice priority limit; can be changed at bootup time.

SP - For non-VC+ systems, set the number of shared programs to 0.

MB - System memory blocks are used by DS. Refer to the DS-1000/IV manual set for more information.

US - For non-VC+ systems, set to 0. For VC+ systems, be sure this includes programmatic and background sessions (i.e., for DS).

Immediately after the US command is labeled system common relocation. The commands available here are the same as the system relocation phase. An END terminates labeled common relocation. Only non-CDS modules can be relocated into system common.

COM - Allocates memory space to unlabeled common. At bootup it is blank and must be initialized by the first program that uses it.

The system message block is relocated into physical memory and has its own map. It is not in system logical memory.

LIB - specifies default library files to be searched whenever a program is loaded by LINK. Typical default libraries are: $FNDLB (non-DS systems) or $FDSLB (DS systems) for FORTRAN programs, $PLIB (PASCAL.LIB) and $SHSLB (PASCAL_SHS.LIB), for PASCAL programs, $BIGLB, the system library which contains several other libraries. These might have different names on your system.

The END command is used to terminate labeled common relocation, system message module relocation, and library specification.

References: System Generation and Installation Manual

MEMORY ALLOCATION PHASE COMMANDS

CLAS, class numbers
RESN, resource numbers
ID, id segments
RS, #reserved partitions
    & bad memory pages
SAM, minimum size of SAM
SL, lower buffer limit, upper buffer limit
BG, priority boundary
QU, quantum, time-slice fence
SP, shared programs
MB, size of system memory block
US, # of concurrent users
RE, module (labeled common)
END
COM, minimum size of unlabeled common
RE, system message block
END
LIB, library file
END

15.22 Memory Allocation Phase Example

This example is for a VC+ system. For non-VC+, the parameters for SL, SP, and US should be set to 0.

References: System Gen. & Instl. Manual, primary answer file

MEMORY ALLOCATION PHASE EXAMPLE

```
*
* VC+SYSTEM
*
CLAS,40
RESN,20
ID,40
RS,0
SAM,2048
SL,200,1048
BG,30
QU,300,50
SP,1
MB,500
US,5
*
END,,,,LABLED COMMON RELOCATION
COM,10
RE, %MSGS
END
*
LIB,$FNDLB
LIB,$BIGLB
LIB,$PLIB
*
END
*
```

DISC BASED INSTALLATION

CHAPTER 16

Table of Contents

Chapter 16
DISC BASED INSTALLATION

MODULE OBJECTIVES

1. Be able to use INSTL and know when to use it.

2. Understand what BOOTEX is and why it is needed, be able to prepare a boot command file and boot the system.

3. Create a system WELCOME file.

4. Execute start-up procedures: USERS program, install system utilities, initalize spool system.

SELF-EVALUATION QUESTIONS

16-1. What files are required on the boot disc LU in order to boot the system?

16-2. When do you need to use INSTL to install a new BOOTEX?

16-3. What does the boot command file do?

16-4. What does the welcome file do?

16-5. How would you boot from a CS80 disc at HPIB address 3, select code 27 with a boot command file called BOOTME?

16-6. What program is used to create user accounts?

16.1 Disc Based Installation Process

The required items for disc-based installation are:

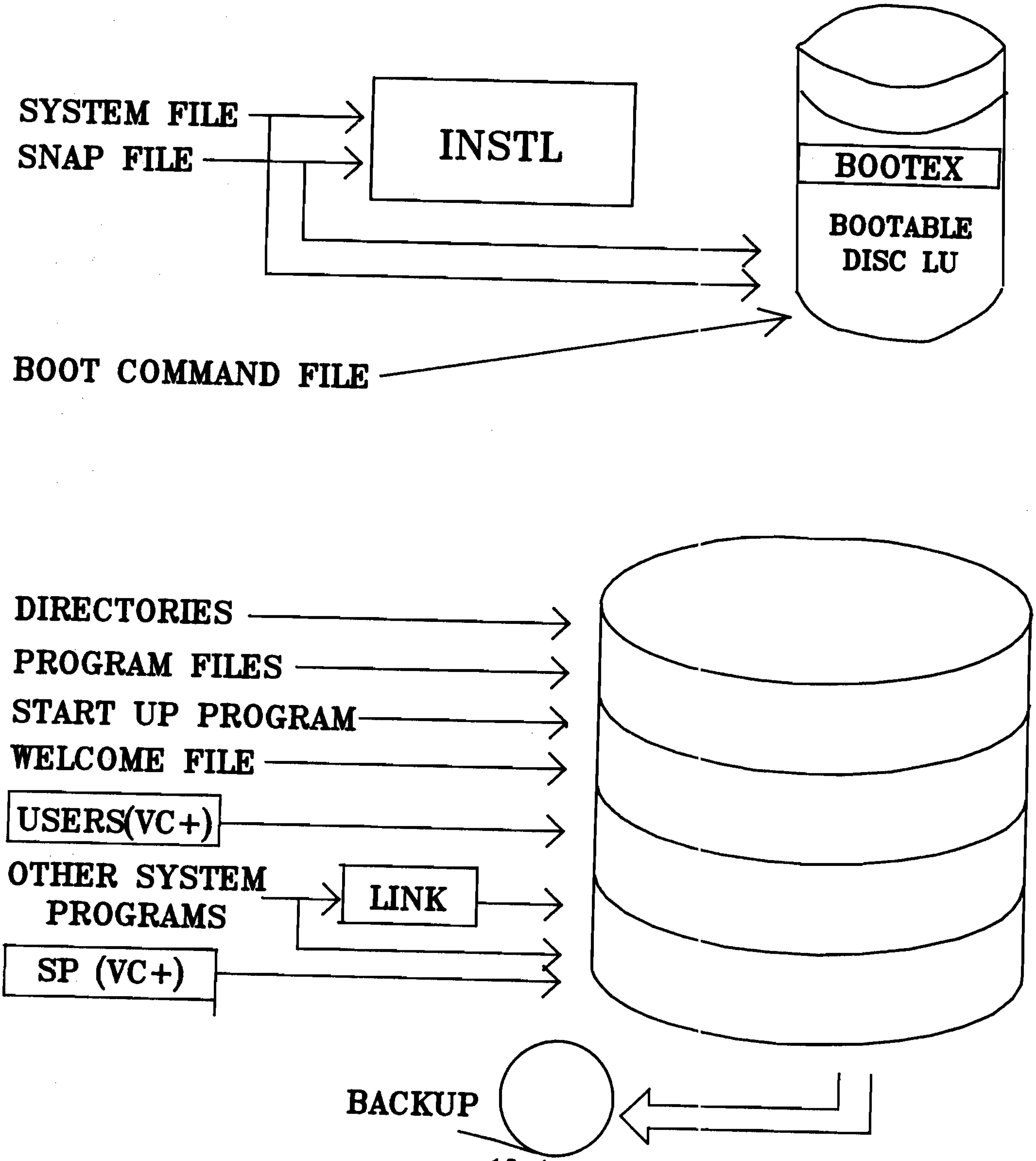
system file (on bootable disc LU)
snap file (on bootable disc LU)
type 6 program files
boot command file (optional) (on bootable disc LU)
welcome file (optional) always under /systems/ in CI

The steps in the installation procedure are:

prepare the boot LU by creating a BOOTEX area, if necessary
install the BOOTEX, if necessary
make the system and snap files available on the boot LU
prepare the boot command file
prepare the welcome file
create the required directories and program files
boot and initialize the system
verify operation and backup the new system

References: System Generation and Installation Manual

DISC-BASED INSTALLATION PROCESS



16.2 Boot Process

The boot ROM loader brings the bootstrap extension, BOOTEX, into memory. BOOTEX is a system which contains a boot program. The boot loader passes a string to BOOTEX which contains the name of the boot command file as specified in the VCP command string.

BOOTEX modifies the system file by setting up ID segments for the RPed programs and some other system tables. The modified system file is then loaded into memory and the start-up program is executed.

References: System Generation and Installation Manual

BOOT PROCESS

BOOT LOADER PUTS
BOOTEX INTO MEMORY.

BOOT LOADER PASSES NAME OF
BOOT CMD FILE TO BOOTEX

BOOTEX MODIFIES SYSTEM FILE

SYSTEM IS LOADED INTO MEMORY

STARTUP PROGRAM EXECUTES
WITH WELCOME FILE

16.3 CREATING A BOOTEX AREA

The LU you boot from must be located at cylinder 0, sector 0 of one of the disc surfaces. For CS80 discs, there will be only one such LU; 7906 discs with removable platters have several. The correct LU is typically the one with the lowest number (in the Primary answer file).

On a CI volume, the 512 block area reserved for BOOTEX cannot be accessed as a CI file and does not appear in any directory. Once the space is created, it will remain there until the LU is re-initialized. Note: IN destroys all the data on the disc!

The FMGR IN command automatically creates a 512 block BOOTEX file at the beginning of the LU. This is a type 1 file with security code of -32767. The BOOTEX file will remain on the LU unless it is purged or the LU is re-initialized as a CI volume. For a description of the FMGR IN command see the RTE-A Utilities Manual.

References: System Gen. & Instl Manual, Utilities Manual

CREATING A BOOTEX AREA

FOR A CI VOLUME:

CI> IN <lu> 512
Re–initialize valid directory [N] ? Y
Initializing disc

FOR A FMGR LU:

FMGR: IN, <msc>,<old crn>,
<new crn>,<label>,
[<opt. prams>]

FMGR 060 DO YOU REALLY WANT TO PURGE DISC? (YES OR NO) YES

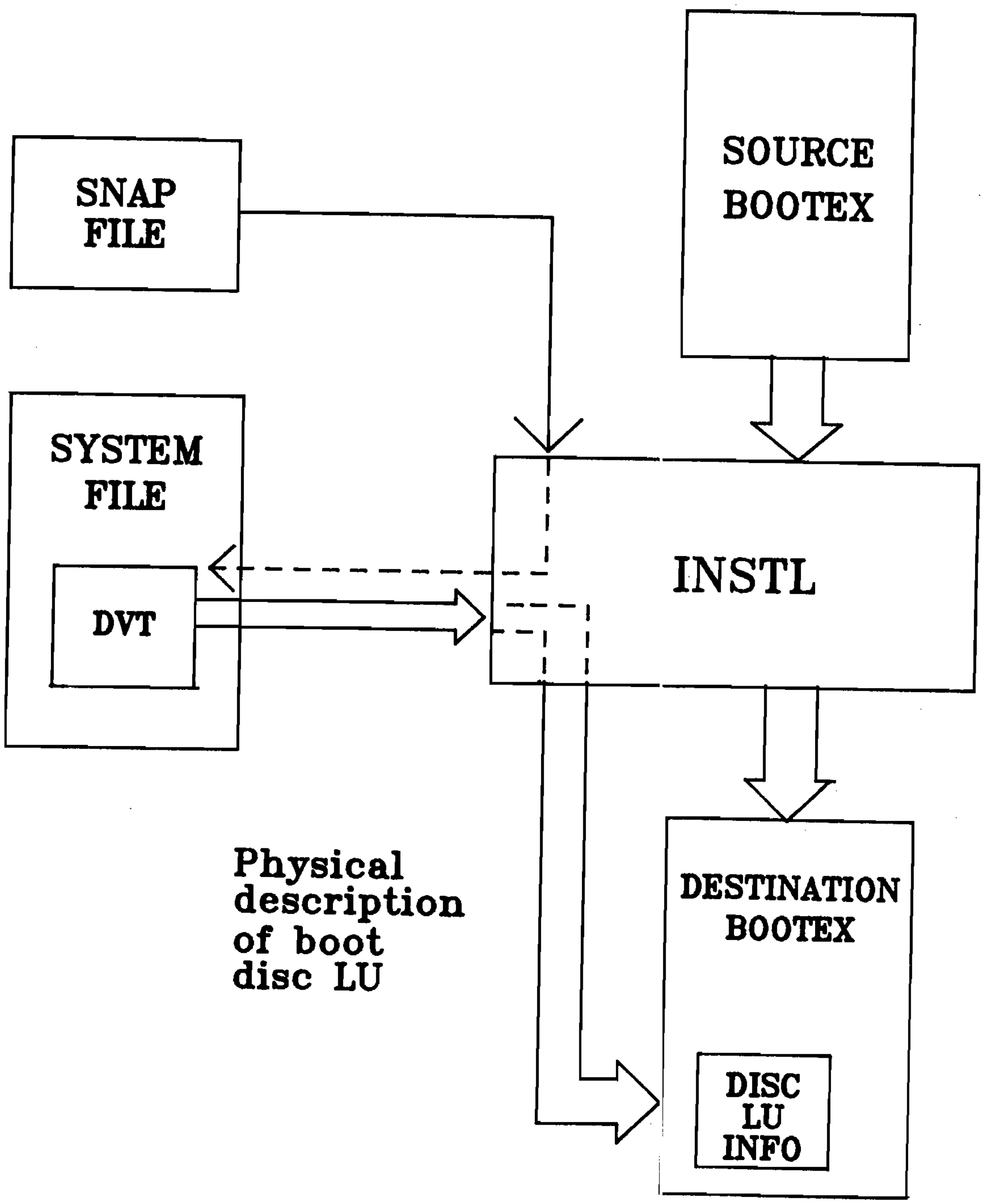
16.4 The INSTL Program

When BOOTEX is loaded into memory, it needs to know what disc LU it was on. The INSTL program puts the required information about the disc LU where BOOTEX can find it. INSTL retrieves driver parameters 1 through 8 for the boot LU and places them in a table within the BOOTEX file. The SNAP file provides the entry points which allow INSTL to find this information.

References: System Gen. & Instl Manual, Utilites Manual

THE INSTL PROGRAM

CI>INSTL <snap> <system> <boot dest> <lu> <boot source>



16.5 Running INSTL - Same Disc Configuration

In these examples, the boot LU, 16, in the target system is the same as LU 16 in the host system

To run INSTL, you always need a source BOOTEX. This could be a previously installed BOOTEX on a FMGR cartridge or a BOOTEX file provided with the RTE-A master software. In these examples, bootex::master is a previously installed BOOTEX file.

CI volume example:

Newsys is a directory containing the system and snap files. Since the BOOTEX area is not a file, a '0' for destination file causes the BOOTEX to be installed at the beginning of the specified LU.

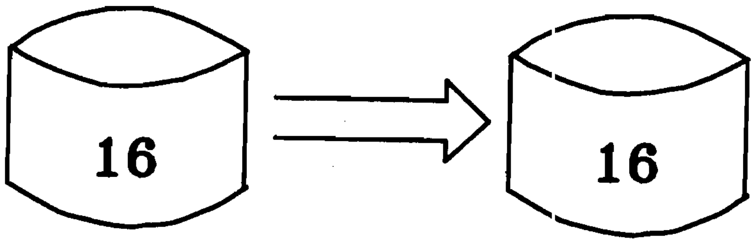
FMGR example:

The BOOTEX on a FMGR cartridge is accessible as a file, so it is specified as the destination file.

Once you have installed BOOTEX on the boot media, it is not necessary to re-install BOOTEX for new generations, unless the disc configuration or LU assignment of the boot LU changes!

RUNNING INSTL

– same disc configuration



FOR A CI VOLUME:

```
CI> wd /newsys
CI> instl
Enter snap file, system file, destination file, lu, and source file
snap.snp,prmsys.sys,0,16,bootex::master
INSTL end.  Your boot extension has been installed at
boot block 0, on LU 16
CI>
```

FOR A FMGR CARTRIDGE:

```
CI> instl
Enter snap file, system file, destination file, lu, and source file
snap::16,prmsys::16,bootex:-32767:16,16,bootex::master
INSTL end. BOOTEX:-32767:16:1:512 is your boot extension file.
warning: boot file must be at cylinder 0 sector 0
CI>
```

16.6 Running INSTL - New Disc Configuration

In these examples, the area described by LU 16 in the host system is LU 14 in the target system. Bootex::master is a previously installed BOOTEX file.

NOTE: In order to boot from a disc LU, it must cover the same physical area in the target system as in the host system. When a disc LU is re-configured to cover a different area, all data on the disc is effectively lost.

CI example:

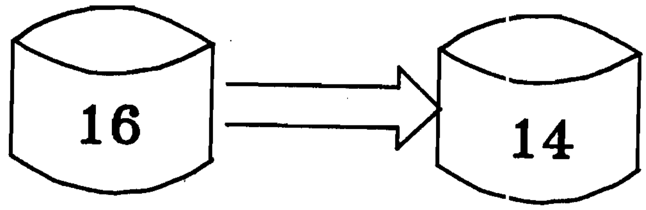
Newsys is a directory containing the system and snap files. Since the target LU is not the same as the current LU, there is no way to access the BOOTEX area. Thus the destination is a file called bootex::newsys. We then use a program called FPUT, which puts the BOOTEX into the reserved area. The offset parameter indicates where on the LU to put the file. It also indicates the bootable file number as given in the boot command string (discussed later). An offset of 0 starts the file at block 0; an offset of 1 starts the file at block 256, etc.

FMGR example:

Here we can directly reference the BOOTEX file on LU 16.

References: System Gen. & Instl Manual, Utilites Manual

RUNNING INSTL
– new disc configuration



FOR A CI VOLUME:

```
CI> wd /newsys
CI> INSTL
Enter snap file, system file, destination file, lu, and source file
snap.snp,prmsys.sys,bootex,14,bootex::master
INSTL end.  BOOTEX::NEWSYS:1:512 is your boot extension file.
warning:  boot file must be at cylinder 0 sector 0

CI> fput
Usage: RU,FPUT,filename,lu, offset
CI> fput bootex::newsys 16 0
CI> _
```

FOR A FMGR CARTRIDGE:

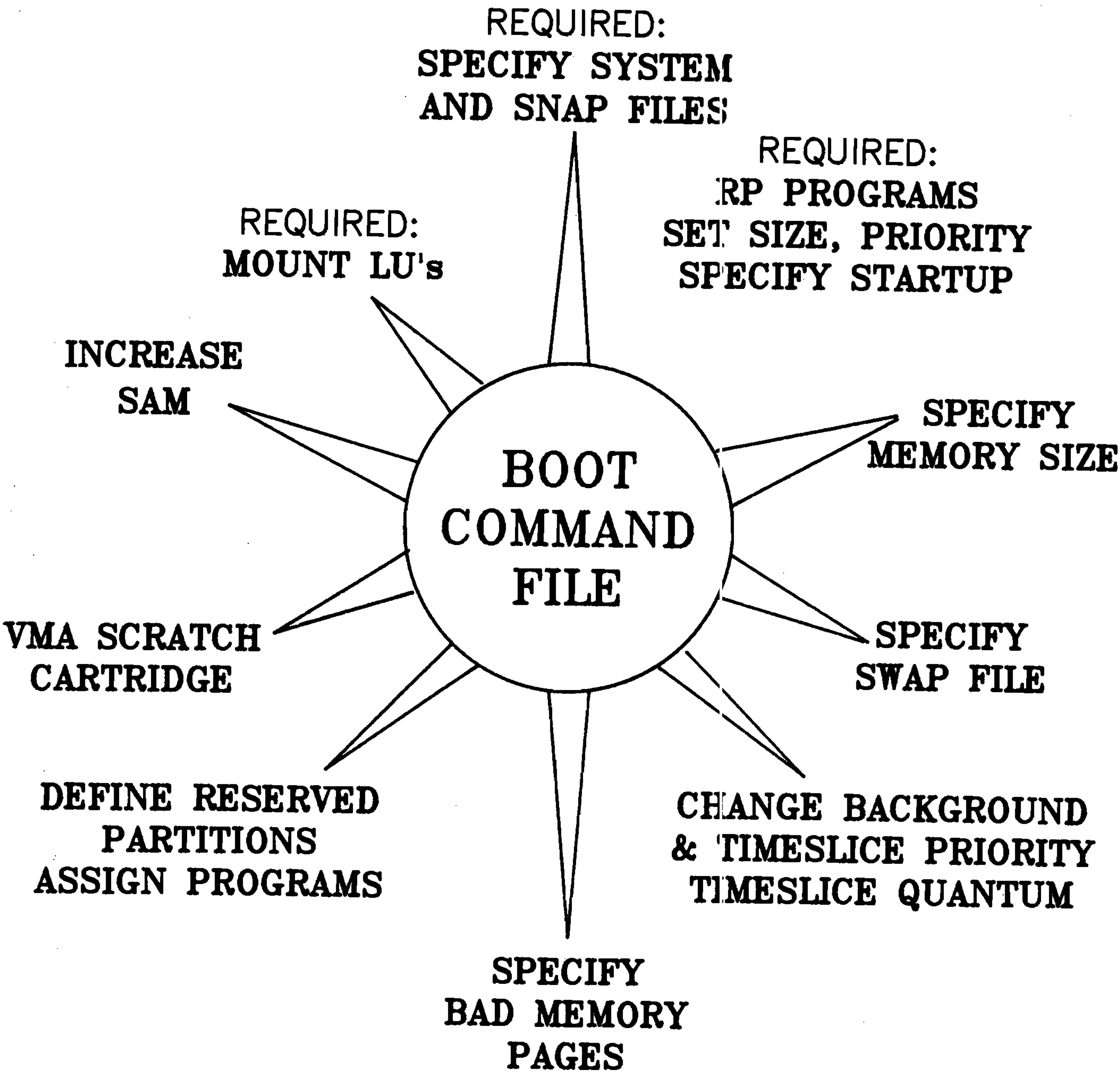
```
CI> instl
Enter snap file, system file, destination file, lu, and source file
snap::15,prmsys::16,bootex:-32767:16,14,bootex::master
INSTL end. BOOTEX:-32767:16:1:512 is your boot extension file
warning:  boot file must be at cylinder 0 sector 0
CI> _
```

16.7 THE BOOT COMMAND FILE

The boot command file is a file passed to the BOOTEX program which controls the boot process. It specifes the system and snap files and indicates how the system file should be modified for execution. The default names for the boot command file are: BOOT.CMD::SYSTEM for a CI volume and SYSTEM for a FMGR cartridge. BOOTEX will prompt for commands interactively if no file is specified and the default file is not found. BOOTEX commands are described in the reference manual.

References: System Generation and Installation Manual

THE BOOT COMMAND FILE



16.8 Boot Command File - Non-VC+ Example

In this example, the boot LU is a FMGR cartridge.

Some Bootex commands:

EC - echo commands
SN - specifies snap file
SY - specifies system file
RP - restore program (create ID segment)
ST - indicates that previously RPed program is startup program
SW - specifies swap file
AS - assign to reserve partition (creates partition)
END - end of boot commands

The system and snap files must be specified first. You must mount the LUs which contain the welcome file and any of the programs which are RPed here. The boot LU is automatically mounted. DRTR must be RPed as D.RTR. The ST command following an RP command makes a program the startup program. An additional copy of CI must be RPed for the startup program, because when CI is the startup program, it will release its ID segment when it exits. The second parameter of the ST command indicates the name of the welcome file which will be passed to the startup CI as follows:

```
ST,,n     ==>  WELCOMEn.CMD::SYSTEM
```

Thus the welcome file here is WELCOME1.CMD::SYSTEM

The SW command will create the swap file if it does not already exist. The swap file is where executing programs are saved when they are swapped out of memory. In the example, a file of size 3000 would be created if it did not exist. If no size is specified, Bootex calculates the default size as follows:

32K words x # of ID segments in system

This is often much larger then you would really need.

The disc directory program, D.RTR, is being assigned to a reserved partition so that it will never be swapped out of memory. This will increase its performance. Note that in order to assign a program to a reserved partition here, there must be sufficient reserved partition table space (allocated at generation time).

References: System Generation and Installation Manual

BOOT COMMAND FILE
non-VC+ example

```
EC
*
*
SY,PRMSYS
SN,SNAP
*
MC,-18
MC,-19
*
*
*
RP,DRTR::PROGRAMS,D.RTR
*
RP,CI::PROGRAMS,CI
*
RP,CI::PROGRAMS,CM
*
RP,CI::PROGRAMS,START
ST,,1
*
*
END
*
*
SW,SWAP:SW:3000
*
AS,D.RTR
*
END
```

16.9 Boot Command File - VC+ Example

In this example the boot LU is a CI volume.

The main difference here is that we only need to RP one copy of CI. This is the startup copy of CI which will go away (release its ID segment) when it exits (typically at the end of the welcome file).

The PROMT program will be RPed in the welcome file. This will take care of scheduling CM, and LOGON as required. CI will typically be scheduled by LOGON, when each user logs on at a terminal. The welcome file here will be WELCOME2.CMD::SYSTEM.

References: System Generation and Installation Manual

BOOT COMMAND FILE
VC+ EXAMPLE

```
EC
*
*
SY,PRMSYS.SYS
SN,SNAP.SNP
*
MC,-18
MC,-19
*
*
*
RP,DRTR::PROGRAMS,D.RTR
*
RP,CI::PROGRAMS
ST,,2
*
END
*
*
SW,SWAP.SWP::SYSTEM::3000
*
AS,D.RTR
*
END
```

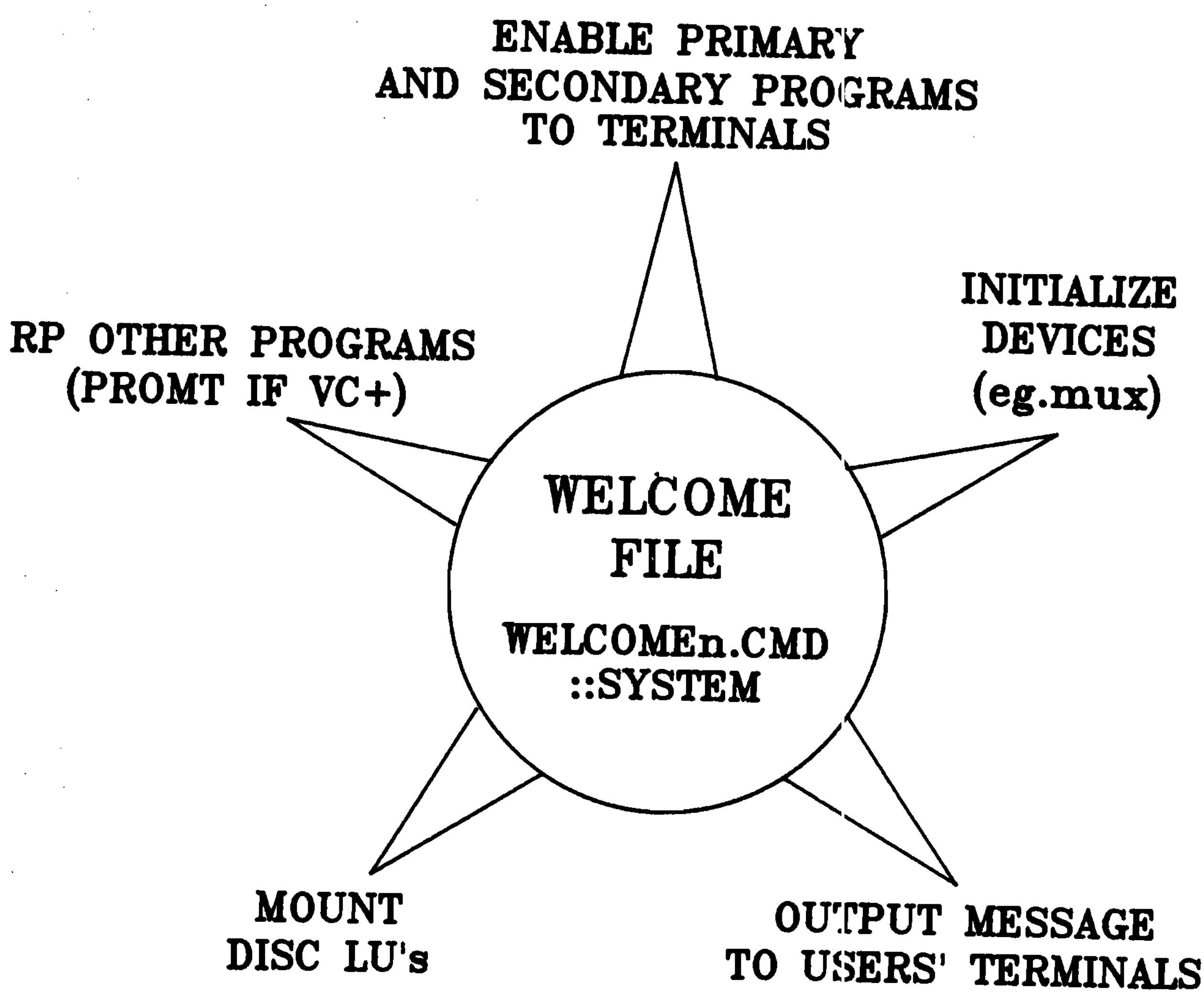
16.10 T H E   W E L C O M E   F I L E

The welcome file is merely a transfer file which is executed by the startup CI. Note that you can mount disc LUs and RP programs here as well as in the boot command file. In the boot command file you must at least mount the LUs and RP the programs you need to run the startup program. Typically, the welcome file would mount the other LUs and RP any other programs as required.

Remember, the primary program is scheduled by the driver when an unexpected interrupt is received from the terminal (i.e., a key is struck). The secondary progam is scheduled if the primary program is busy.

References: System Generation and Installation Manual

THE WELCOME FILE



16.11 Welcome File - Non-VC+ Example

In the non-VC+ environment, you must RP CI with different names to serve as the primary and secondary programs at different terminals. This is because the driver, in order to schedule a program on interrupt, requires that there be an ID segment for the program and that the program is not busy. To insure that the program is not busy when we get an asynchronous interrupt, there must be multiple copies of the program RPed. The CN command is a control request to the driver - this is equivalent to an EXEC 3 control request. The format of the CN command is:

```
CI> cn <lu> <function> <prams>
```

For terminal drivers:

```
20B = schedule primary program on interrupt
40B = schedule secondary program on interrupt
praml = name of program to be scheduled
```

The command:

```
cn 1 40b cm,,,CM
```

will cause the value 'CM' to be passed to the program when it is scheduled by the driver. This tells it that it is really the CM program. That is, it should process one command and then exit. (Remember, CM is really just CI in disguise).

References: System Gen. & Instl Manual, Driver Reference Manual

WELCOME FILE
NON-VC+ EXAMPLE

```
wd /programs
*
mc 22
mc 23
mc 29
*
* Enabling terminals
*
cn 1 20b CI
cn 1 40b CM,,,CM
*
rp CI ci68
rp CI cm68
rp CI ci66
rp CI cm66
*
cn 68 20b ci68
cn 68 40b cm68,,,CM
cn 66 20b ci66
cn 66 40b cm66,,,CM
*
* Send message to terminals
*
co mess.txt::system 1
co mess.txt::system 68
co mess.txt::system 66
*
ex
```

16.12 Welcome File - VC+ Example

It is not necessary to RP additional copies of CI or even to schedule secondary programs. PROMT, when enabled as the primary program, will handle the scheduling of LOGON and CM. LOGON will (typically) schedule CI. Note PROMT looks for the program:

    CI.RUN::PROGRAMS

PROMT is never busy because it will schedule the appropriate program and then exit.

The control request 30B initializes the MUX ports. This sets the baud rate, the port number and some handshake information.

References: System Gen. & Instl Manual, Driver Reference Manual

WELCOME FILE
VC+ EXAMPLE

```
wd /programs
*
mc 22
mc 23
mc 29
*
cn PROMT
*
* Enable terminals
*
cn 1 20b PROMT
*
* Enable muxs
*
cn 71 30b 142330b
cn 72 30b 152331b
cn 73 30b 152332b
cn 74 30b 152333b
cn 75 30b 152334b
cn 76 30b 152335b
cn 77 30b 152336b
cn 78 30b 152337b
*
cn 71 20b PROMT
cn 72 20b PROMT
cn 73 20b PROMT
cn 74 20b PROMT
cn 75 20b PROMT
cn 76 20b PROMT
cn 77 20b PROMT
cn 78 20b PROMT
*
ex
```

16.13 System Utilities and Directories

The directory /SCRATCH is also useful. It should have r/w access for all users.

The programs shown must be available for the target system prior to bootup. They can either be loaded or transported. Typically, a program will be transportable if it does not use system common.

Reload or transport ?

If your new system uses the same RPLs as your old system, then you can transport programs. If the RPLs for new system vary from your current system only in the use of CDS then you will not need to reload programs, except for CI, which is available in both CDS and non-CDS. CDS programs will not run in a non-CDS system. For any other situation where you change RPLs, system programs should be reloaded.

The first time you run a program under a new system with different RPLs, the system will print a message:

changed RPL checksum <program>

When you run the program again, it will run and the message will not be printed.

A program can be loaded for use on a different system then the current one by specifying the target snap file when LINK is run (SN command).

There are many intrinsics of the RTE-A that expect the disc directory program to be called "D.RTR". Since this is not a legal file name, the program file is called "DRTR.RUN". It is always RPed as "D.RTR".

References: System Generation and Installation Manual

SYSTEM UTILITIES AND DIRECTORIES

REQUIRED DIRECTORIES

- /SYSTEM
- /PROGRAMS
- /LIBRARIES
- /HELP

SYSTEM PROGRAMS REQUIRED AT BOOTUP

- CI
- DRTR (D.RTR)
- DL
- LINK
- PROMT (VC+)

16.14 SYSTEM BOOTUP & INITIALIZATION

SYSTEM BOOTUP AND INITIALIZATION

16.15 Disc Boot Command String Examples

The disc boot command string is entered at the VCP terminal. The %BDC command is used to boot an HPIB disc. The %BDI command is used when booting from a 2480 integrated disc (either the hard disc or the microfloppy).

After the system has been checked for proper operation, you may configure the BOOT SELECT switches on the processor or frontplane for automatic bootup.

References: Sys. Gen. & Instl Manual, A600/A700/A900 Comp. Ref. Manual

DISC BOOT COMMAND STRING EXAMPLES

direction of reading by Boot Rom ←

% B D C 0 0 2 7 S Y S T E M

- % — bootup
- B D C — from HP-IB disc
- 0 — disc HP-IB address
- 0 — unit or head number
- 2 7 — select code of disc's HP-IB interface card
- S Y S T E M — File name boot command file for disc-based system

% B D I 0 0 3 2 S Y S T E M

- % — bootup
- B D I — from 248x integrated disc
- 0 — drive address
- 0 — unit number
- 3 2 — disc interface card select code
- S Y S T E M — Filename boot command file for disc-based system

16.16 Load or Transport Other Programs

It may be desirable to make these programs available for the target system prior to bootup.

LOAD OR TRANSPORT OTHER SYSTEM PROGRAMS

	for VC+
EDIT	SP
WH	OUTPT
IO	SMP
FMGR	SPGET
TF	LOGON
	USERS

PLUS

OTHER UTILITIES
COMPILERS
APPLICATION PROGRAMS

16.17 User Accounts (VC+)

The directory ::USERS contains all the information used to describe the user accounts on the system. LOGONPROMPT is a text file containing the logonprompt. This can be changed by using EDIT. Only the first 16 characters of the file will be used. There is a file per user (file name = logon name) which contains a complete description of the user account information.

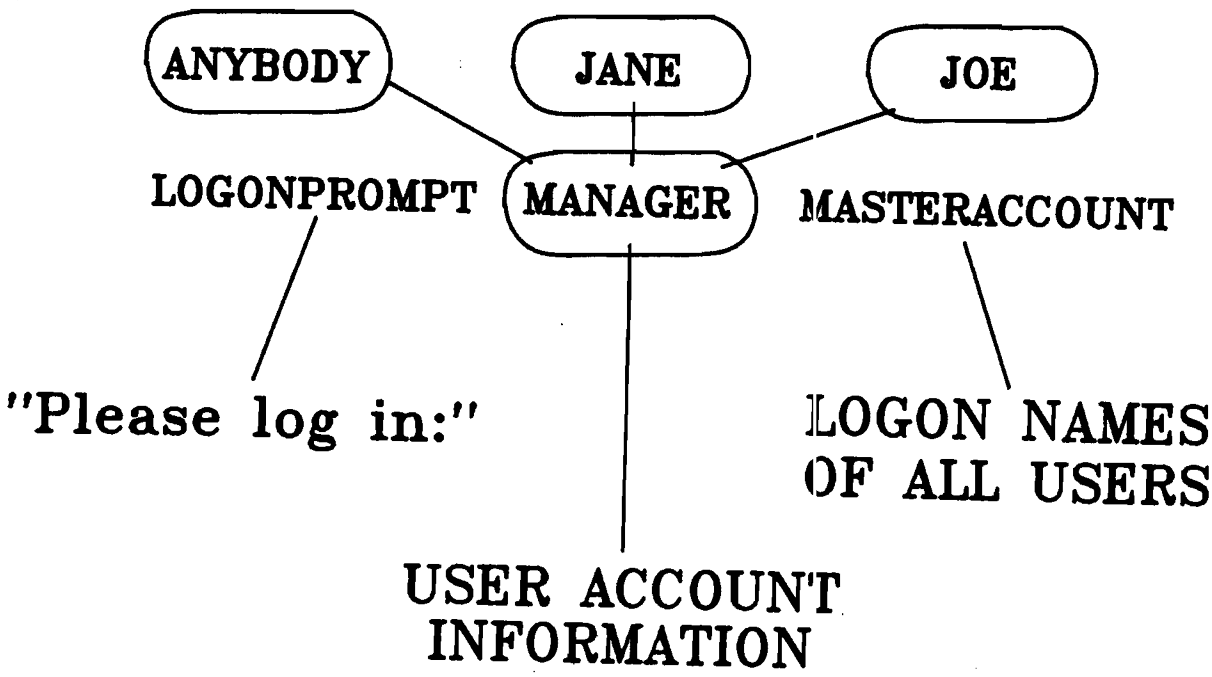
If you boot a VC+ system for the first time with no user accounts information on the system, CI will be scheduled without the need to logon. The USERS program will set up ::USERS with everything needed to run in multi-user environment.

References: System Generation and Installation Manual

USER ACCOUNTS (VC+)

CI> dl ::users

directory :: users



16.18 Creating a User Account (VC+)

USERS is an interactive program used to create or modify user accounts. It prompts you for user information, giving default values in [brackets] where applicable. This example shows the creation of an account which is not the first.

Note that in this example, most of the entries were defaulted. The default LU of 0 for working directory indicates the first CI volume in the cartridge list.

The creation of the first account is slightly different. For the first account, you will specify which LU ::USERS will go on and the logon prompt. The superuser flag is always set for the first user.

To create a user account, you need read/write access to the users directory. Creating accounts is typically done only by a superuser. A non-superuser cannot create a superuser account.

References: System Generation and Installation Manual

CREATING
A USER ACCOUNT (VC+)

CI>users

This program creates or modifies user accounts. Use carriage return to take the choice in [brackets] . Use <CNTL-D> to quit early.

Creating a user

Enter your logon name: CUTHBERT

Enter your real name: Cuthbert Q Divine

Enter your password: WOLF

Set superuser flag? (Yes or No) [No]

Enter your working directory:
[::CUTHBERT]

Enter your start-up command:
[RU CI.RUN::PROGRAMS]
RU CI HI::CUTHBERT

Create Directory ::CUTHBERT ? [YES]

What LU should the directory go on?
0 29

Created user CUTHBERT

16.19 Modifying a User Account (VC+)

Running USERS with an account name causes the account to be modified. The program will indicate the existing values as the defaults in brackets and prompt for new values. You need only enter new values for those entries that are to be changed. Note that there is no way to see the existing password.

To modify an existing account, you need read/write access to the MASTERACCOUNT file and the account file.

References: System Generation and Installation Manual

MODIFYING A USER ACCOUNT (VC+)

```
CI> users CUTHBERT

This program creates or modifies user accounts.
Use carriage return to take the choice in
[brackets].  Use <CNTL-D> to quit early.

Modifying user CUTHBERT

Your current logon name is CUTHBERT
Enter your new logon name: [CUTHBERT] WOLF
Your current real name is Cuthbert Q Divine
Enter your new real name: [Cuthbert Q Divine] _
Enter your password: [cr] _
Change password to no password (Yes or No) [Yes] _
Set superuser flag? (Yes or No)[NO] YES
        .
        .
        .
Created user WOLF
```

TO DELETE A USER ACCOUNT

```
CI> pu wolf::users
```

16.20 Initialize Spool System (VC+)

The command SP IN sets up the SPOOLINFO file and allocates resource and class numbers.

The second command turns on error logging and specifies the file to which the errors will be logged. To turn error logging off, use:

```
CI> sp lo of
```

You may want to put these commands in the welcome file since it must be done whenever the system is rebooted.

References: System Generation and Installation Manual

INITIALIZE SPOOL SYSTEM (VC+)

CI> sp in

CI> sp log on errors::log

16.21 DS Transparency Software Installation

The DS transparency software allows access to files on other RTE-A systems connected via the DS/1000-IV network. Installation of the DS network is described in the DS manuals. The DS transparency software installation follows the DS network installation.

DSRTR and TRFAS are DS monitors provided with the RTE-A software. DINIT is a DS initialization program.

References: System Gen. & Instl Manual, DS/1000-IV Manuals

DS TRANSPARENCY SOFTWARE INSTALLATION

* IF YOU HAVE THE DS/1000-IV PRODUCT

- LOAD DSRTR AND TRFAS ON YOUR SYSTEM
- RP DSRTR AND TRFAS
- EDIT THE DINIT COMMAND FILE SO THAT TRFAS IS SCHEDULED BY DINIT
- INITIALIZE DS BY RUNNING DINIT
- VERIFY THAT DS TRANSPARENCY IS SET UP

16.22 System Verification and Backup

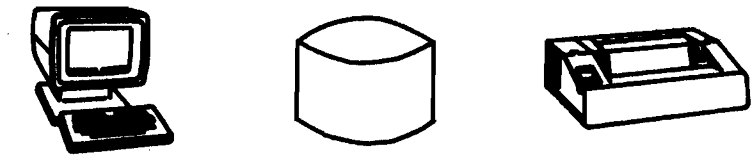
After everything is loaded and initialized, you should check for proper operation.

The FTEST program functionally tests each peripheral device on the system.

References: System Gen. & Instl Manual, Primary System Install. Manual

SYSTEM VERIFICATION & BACKUP

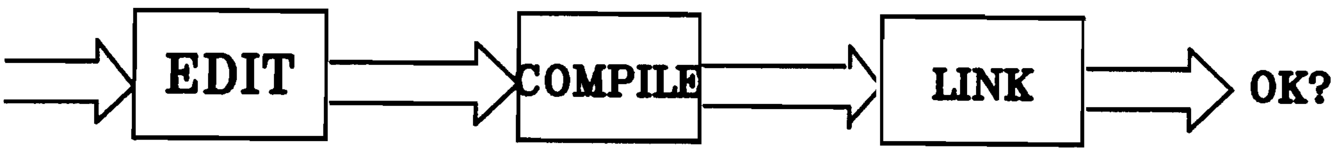
FTEST

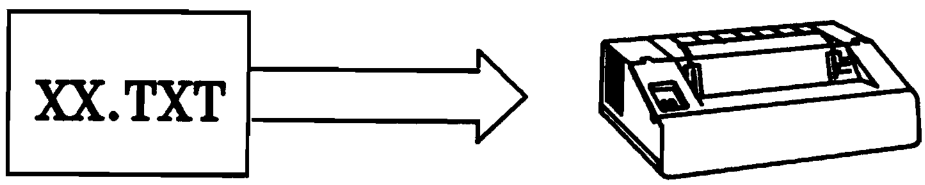


CI> WH

CI> DL

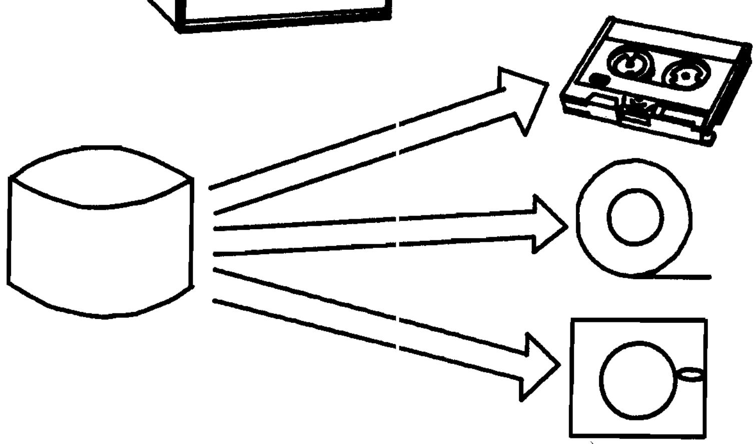
CI> LI AFILE





RUN APPLICATION PROGRAMS

BACKUP



MEMORY BASED SYSTEM INSTALLATION

CHAPTER 17

Table of Contents

Chapter 17
MEMORY BASED SYSTEM INSTALLATION

MODULE OBJECTIVES

1. Understand what a memory based system is and what the considerations are for memory based systems.

2. Be able to generate and install a memory based system.

3. Be able to run BUILD.

4. Be able to install and boot from various media - CTD, disc, PROM, DS/1000.

SELF-EVALUATION QUESTIONS

17-1. How are programs loaded into memory from disc when you are running a memory based system?

17-2. Indicate the utilities required to install a memory based system on the following media:

CTD
Magtape
Disc
DS1000/IV
PROM

17-3. What are the outputs of the build program?

17-4. Which VCP command is used to boot from the following media?

Magtape
CTD
Disc
DS-1000/IV
PROM

17.1 WHAT IS A MEMORY BASED SYSTEM.

All programs reside in reserved partitions in memory. Non-CDS program segment overlays are not allowed. If you have the VC+ option, CDS segmented programs are allowed as long as all the segments can reside in memory at the same time.

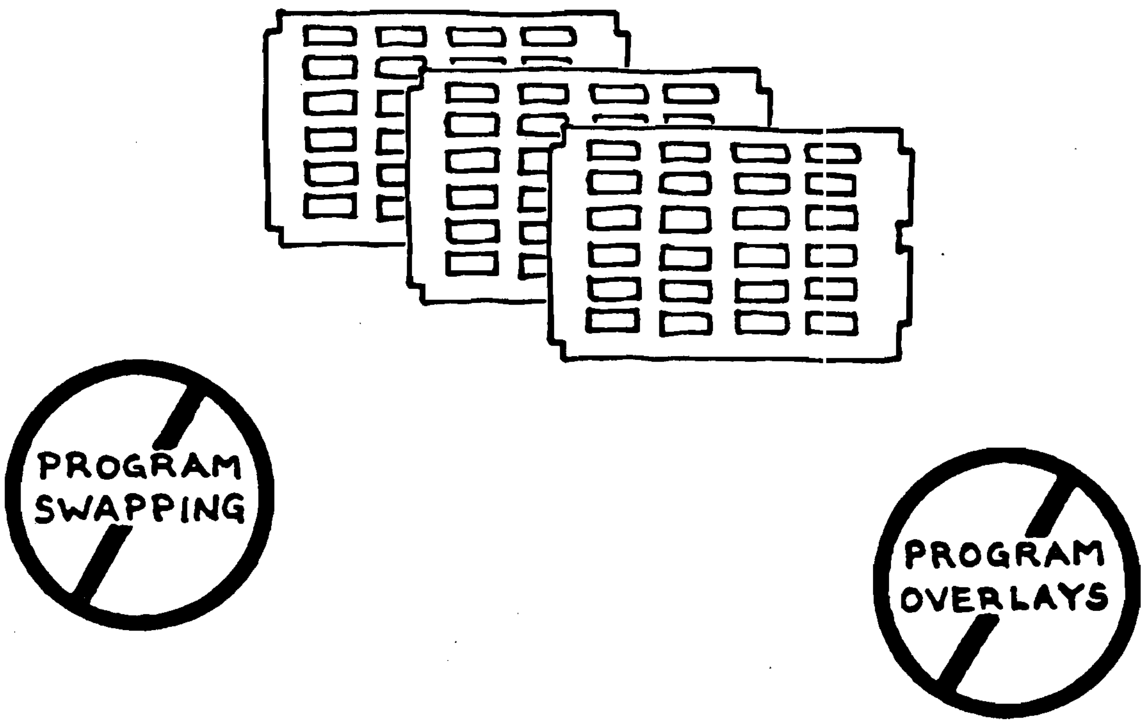
References: System Generation and Installation Manual

17.1 W H A T I S A M E M O R Y B A S E D S Y S T E M .

All programs reside in reserved partitions in memory. Non-CDS program segment overlays are not allowed. If you have the VC+ option, CDS segmented programs are allowed as long as all the segments can reside in memory at the same time.

References: System Generation and Installation Manual

WHAT IS A MEMORY BASED SYSTEM?



ALL PROGRAMS MUST RESIDE IN MEMORY

17.2 Boot Media

Magnetic Tape

Cartridge tape (CTD)

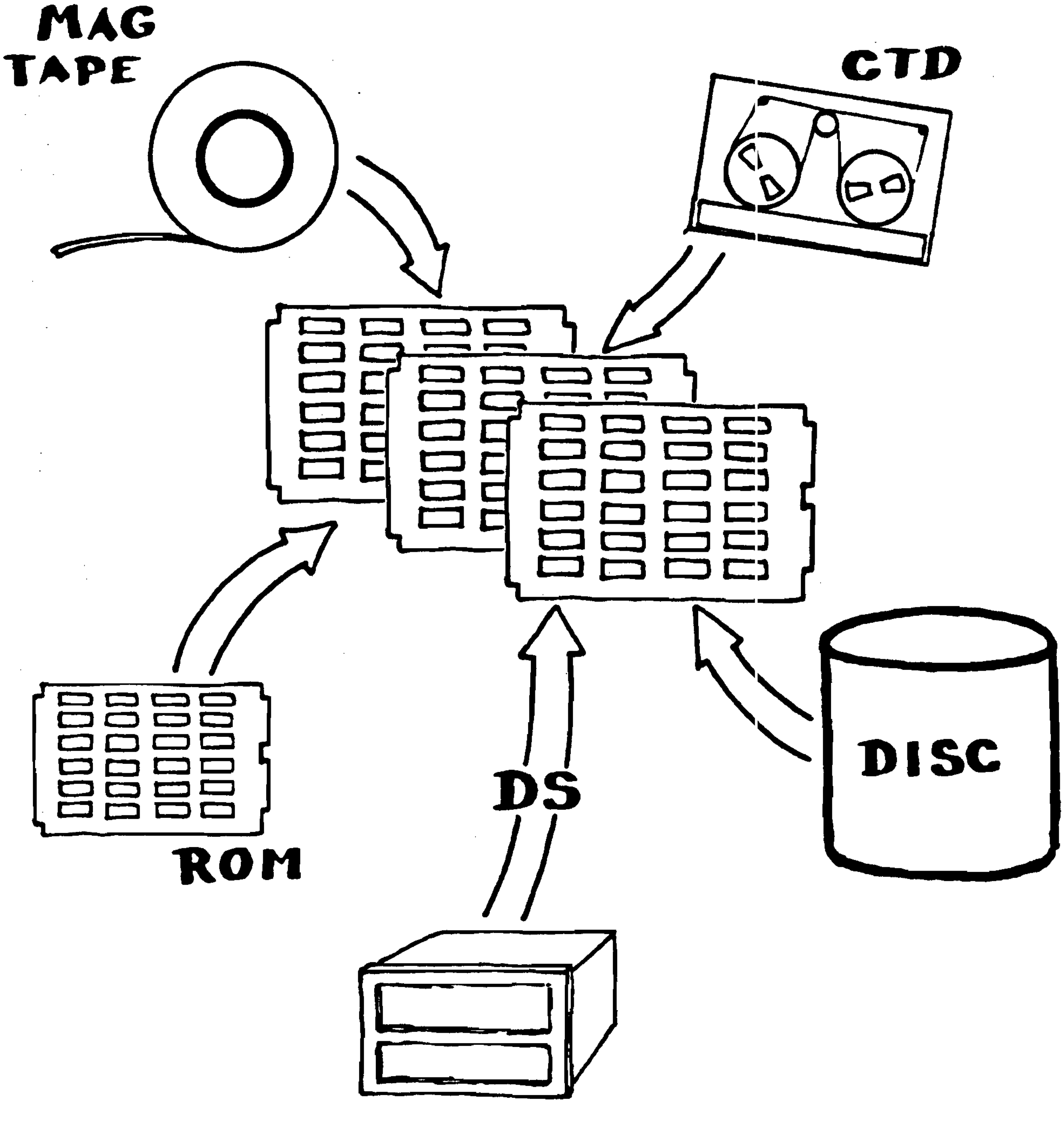
Disc

DS/1000-IV (from a remote disc based system)

PROM module

References: System Generation and Installation Manual

BOOT MEDIA



17.3 Memory Based System Examples

ARSTR - Physical restore utility

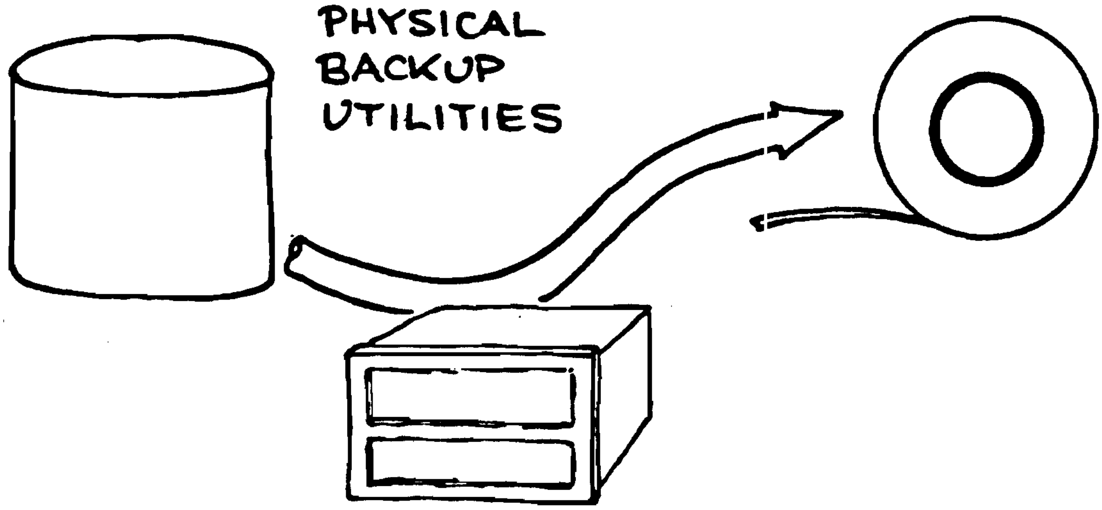
PBV - Verifies a pushbutton backup or restore

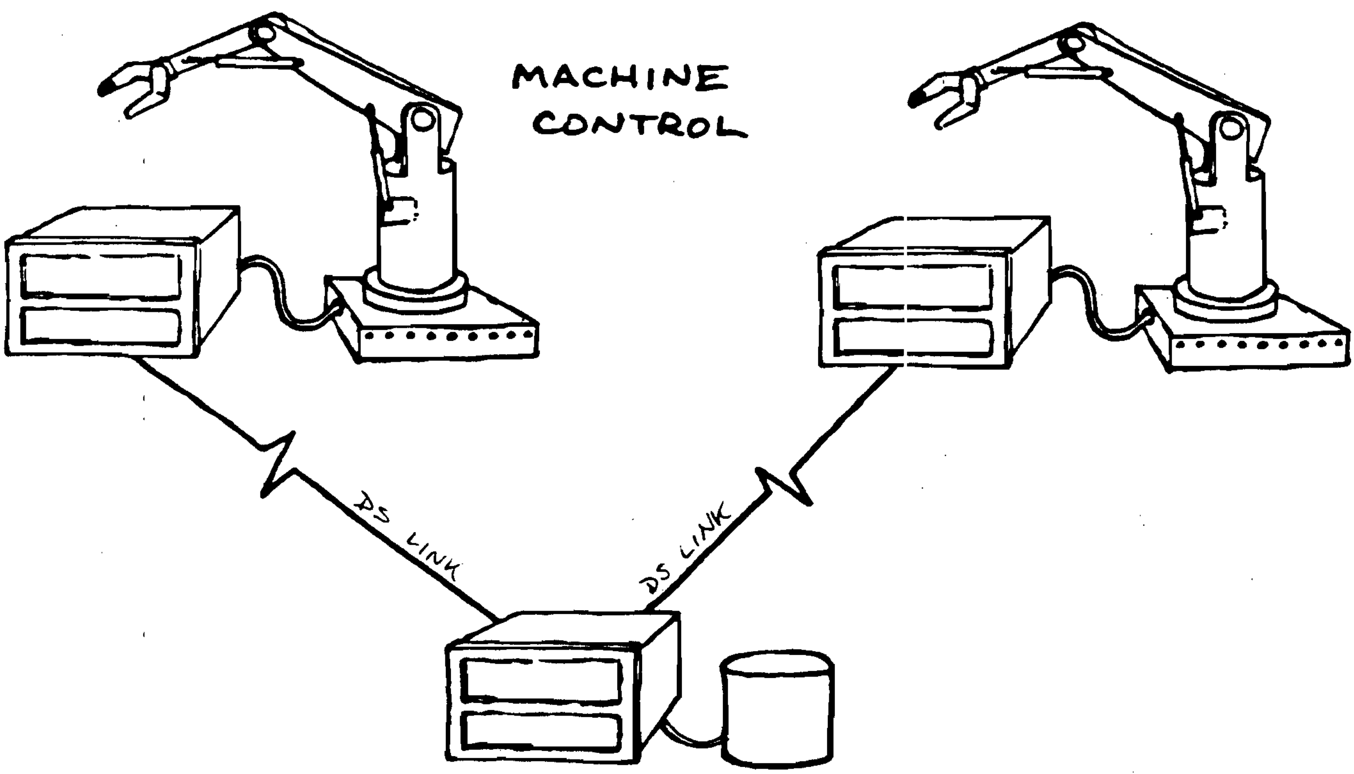
FORMT - Disc format utility

(These utilities are discussed in the next chapter).

Measurement and control applications

MEMORY BASED SYSTEM EXAMPLES





17.4 Memory Based Installation Process

The required items for memory based installation are:

system file
snap file
program files (type 6)

The steps in the installation process are:

* Merge the system file and the program files with the BUILD program.

* Install the merged system file on the bootable medium.

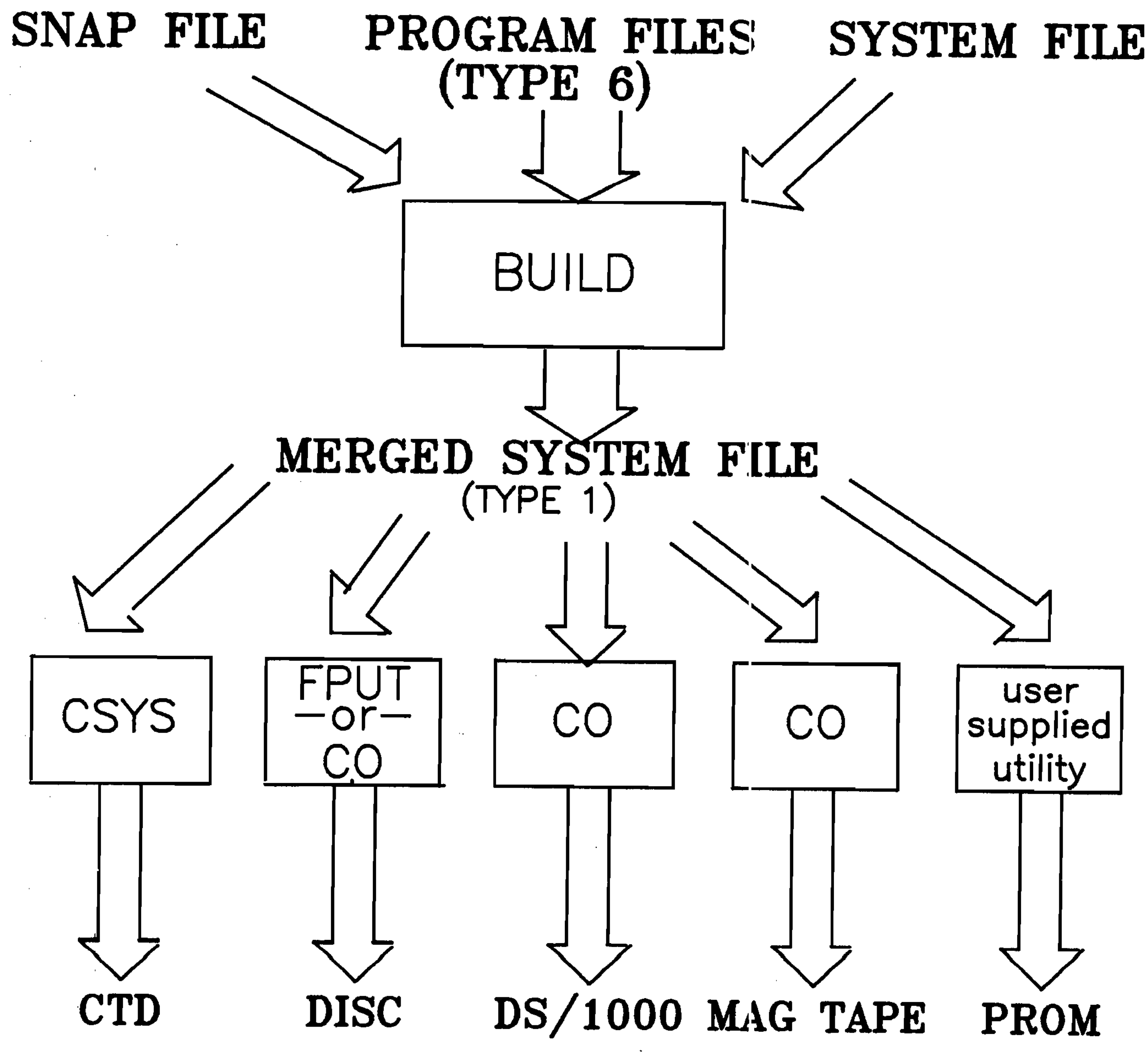
* Bootup the target system.

* Verify operation and backup the system.

NOTE: There is no supported PROM installation procedure. The user must supply a program to translate the type 1 system file to the appropriate format for the PROM burner and PROMs used. Refer to the RTE-L/XL PROM User's Guide (92070-90030) for additional information.

References: System Generation and Installation Manual

MEMORY BASED INSTALLATION PROCESS



17.5 B U I L D - P H A S E 1

The BUILD program combines the system file with all the type 6 program files that will run in the target system. No programs can be loaded from the disc in a memory based system, so all required programs must be loaded in this way. BUILD can be run interactively or with a command file. When run interactively, BUILD will prompt the user for all inputs.

Command - file from which commands are taken. This looks exactly like the inputs you would enter if running interactively.

List - Shows user inputs and any messages output by BUILD.

Output - The type 1 (memory image) merged system file. This is a bootable system image.

Snap - Snapshot file output by generator.

System - System file output by generator or the output file of a previous run of BUILD. BUILD does not modify this file.

OP1 - Specifies what exit path to take on errors when input is from a command file:

/A - abort
/E - exit
/C - continue

The build program has 3 phases.

References: System Generation and Installation Manual

BUILD-PHASE 1

**CI> build <command> <list> <output> <snap>
<system> <opt 1>**

Phase 1: check for runstring information
– prompt for any file not specified.

17.6 BUILD - Phase 2

All programs in a memory based system must reside in reserved partitions.

Automatic partitioning - The memory partitions are automatically built as each program is merged. The size of the partition is the size of the program.

Manual partitioning - The user first defines the size of all the partitions in the system and can then explicitly assign programs to partitions. This is useful if you want to leave bad pages of memory unused.

References: System Generation and Installation Manual

BUILD-PHASE 2

- SPECIFY MEMORY SIZE
- SELECT AUTOMATIC OR MANUAL RESERVED PARTITION DEFINITION
- IF MANUAL PARTITIONING, DEFINE SIZE OF EACH PARTITION.

17.7 BUILD - Phase 3

The RP command is used to build an ID segment for each program and to load it into the merged system file.

Note: You cannot size a VMA or EMA program with BUILD. Such programs must be correctly sized when loaded.

If manual partitioning is used, a partition number can be specified. If none is specified, then BUILD loads the program into the smallest unused partition which will hold it.

VC+ option:

Build will assign appropriate partitions for code and data if the program is a CDS program. If a program is a shareable program, then BUILD will create one code partition and one data partition when the program is RPed. Additional copies of the program will be given a data partition and will share the same code partition.

References: System Generation and Installation Manual

BUILD–PHASE 3

- RESET SYSTEM SECURITY CODE (for FMGR)
- BUILD ID SEGMENT, MERGE PROGRAM INTO SYSTEM, ASSIGN TO PARTITION
  - SET PRIORITY
  - SET SIZE
  - SPECIFY WHICH PARTITION (IF MANUAL PARTITIONING)
  - INDICATE STARTUP PROGRAM
- DISPLAY PARTITION TABLE

17.8 Build - Automatic Partition Example

In this example BUILD was run interactively. The inputs and outputs shown appear in the list file.

Commands:

RP - create ID segment for program
ST - indicates previously RPed program is startup program
PT - display partition table
/E - exit

Partitions are created as each program is RPed.

References: System Generation and Installation Manual

BUILD–AUTOMATIC PARTITIONING

EXAMPLE

```
CI> build 1 ex1.lst ex1.sys snap prmsys
        .
        .
        .
    Do you want automatic partition construction
    (YES/NO) ? YES

    Physical memory size in K words (nnn) ? 128

     76 pages of memory remaining.
    280 memory descriptors remain undefined.

    BUILD: RP,MASTR.RUN

     66 pages of memory remaining.
    BUILD: ST

    BUILD: RP,CTRL1.RUN

     39 pages of memory remaining.
    BUILD: RP,CTRL2,RUN

     12 pages of memory remaining
    BUILD: PT

    prtn num  low page  length occupant
           1        52      10 MASTR (data)
           2        62      27 CTRL1 (data)
           3        89      27 CTRL2 (data)
    BUILD: /E

    BUILD completed.
    Bootable system image in file /LN/EX1.SYS:::1:928:128
```

17.9 Build - Manual Partition Example Using VC+

This example shows the command file for build where the partitions are manually defined. There are 4 partitions which will contain programs and one which has bad memory. This example uses the VC+ option. CTRL.RUN is a shareable CDS program. Thus there is one partition for the shared code and one data partition for each copy of the program.

References: System Generation and Installation Manual

BUILD–MANUAL PARTITIONING

EXAMPLE (using VC+)

```
CI>build ex2.cmd ex2.lst ex2.sys snap prmsys
CI> li ex2.cmd
```

```
NO,                 manual partitioning
128,                memory size
10,                 define partitions
2,                  bad pages of memory
32,
16,
16,
RP,MASTR.RUN,,
ST,
RP,CTRL.RUN,CTRL1
RP,CTRL.RUN,CTRL2
PT
/E
```

The PT command would show:

Partition #	Low Page	Length.	Occupant
1	52	10	MASTR (data)
2	62	2	<none>
3	64	32	CTRL1 CTRL2 (shared code)
4	96	16	CTRL1 (data)
5	112	16	CTRL2 (data)

17.10 INSTALLATION AND BOOTUP

INSTALLATION AND BOOTUP

17.11 Installation on CTD and Mag Tape

CSYS (copy system) copies a type 1 memory image file from a CS/80 disc to a CTD, allowing the system to be directly booted from tape. NOTE: CSYS does not acknowledge the BR command. Once started, it continues to completion or until aborted by an error. The file number determines the starting block for the file on tape. This also corresponds to the file number specified in the boot command string (discussed later). File number 0 starts at block 0, file 1 starts at block 256, etc. (1 block = 128 words).

References: System Gen. & Instl Manual, Utilities Manual

INSTALLATION ON CTD AND MAG TAPE

CTD

CI> CSYS

RU,CSYS, SYSTEM FILE, <TAPE LU>, FILE NO.

CI> CSYS merged.sys 24 0

CSYS COMPLETED 1024 BLOCKS
WRITTEN TO TAPE

MAG TAPE

CI> co merged.sys 8

copying merged.sys to 8. . . [ok]

CI>

17.12 Installation on Disc

To be directly bootable from the disc, the system must be on an LU that starts at physical cylinder 0 and sector 0 of the disc drive. Although the first file on the disc is normally used for bootup, the VCP allows bootup from other files. The bootup file is described to the VCP by its physical location on the disc. Bootable file 0 begins at cylinder 0, sector 0; bootable file 1 begins 256 blocks further in on the disc, etc.

To boot directly from a CI volume, FPUT is used to place the system file in the reserved area at the start of the disc LU. (Remember the IN command ?). The offset parameter is equivalent to the bootable file number. That is, an offset of 0 starts at block 0; an offset of 1 starts at block 256, etc.

To boot from a FMGR cartridge, the system file need only be copied onto the cartridge such that it starts at an integral multiple of 256 blocks and the exact location is known. The easiest way to do this is to use a cartridge which has been cleared of all files except BOOTEX (512 blocks). Then create additional bootable files in multiples of 256 blocks, as required.

Alternately, you could boot your memory based system using an installed BOOTEX by passing the type 1 system file to BOOTEX in the VCP command.

References: System Gen. & Instl Manual, Utilities Manual

INSTALLATION ON DISC

For direct boot from a CI volume

```
CI> fput
 USAGE: RU, fput, filename, lu,[offset]
CI> fput merged.sys 16 0
CI>
```

For direct boot from a FMGR cartridge.

```
CI> co merged.sys boot2::16
 copying MERGED.SYS to BOOT 2::16. . .[ok]
CI>
```

Or using an installed BOOTEX.

No installation required

Merged system file must be on bootable LU

17.13 DS/1000-IV Installation and Bootup Process

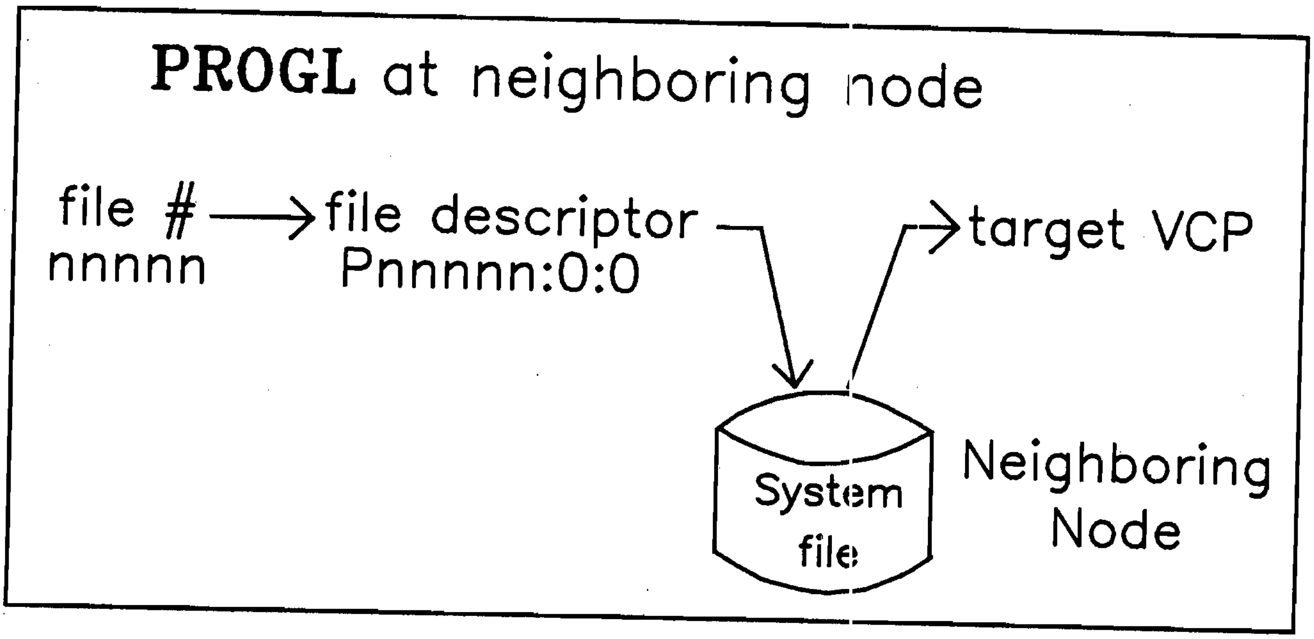
To boot over DS, the DS program PROGL must be available at a node which connects directly to a DS interface card at the target system. This is the card whose select code is given in the boot command string (shown later). PROGL cooperates with the VCP at the target system.

You can boot from a neighboring node without using the routine #DNFL. The file number specified at boot time is converted to the file descriptor by PROGL. On an RTE-6/VM system, this must be on a cartridge under the account MANAGER.SYS. On an RTE-A system, it must be on a FMGR cartridge. No cartridge reference can be specified, so the file should be on the top cartridge.

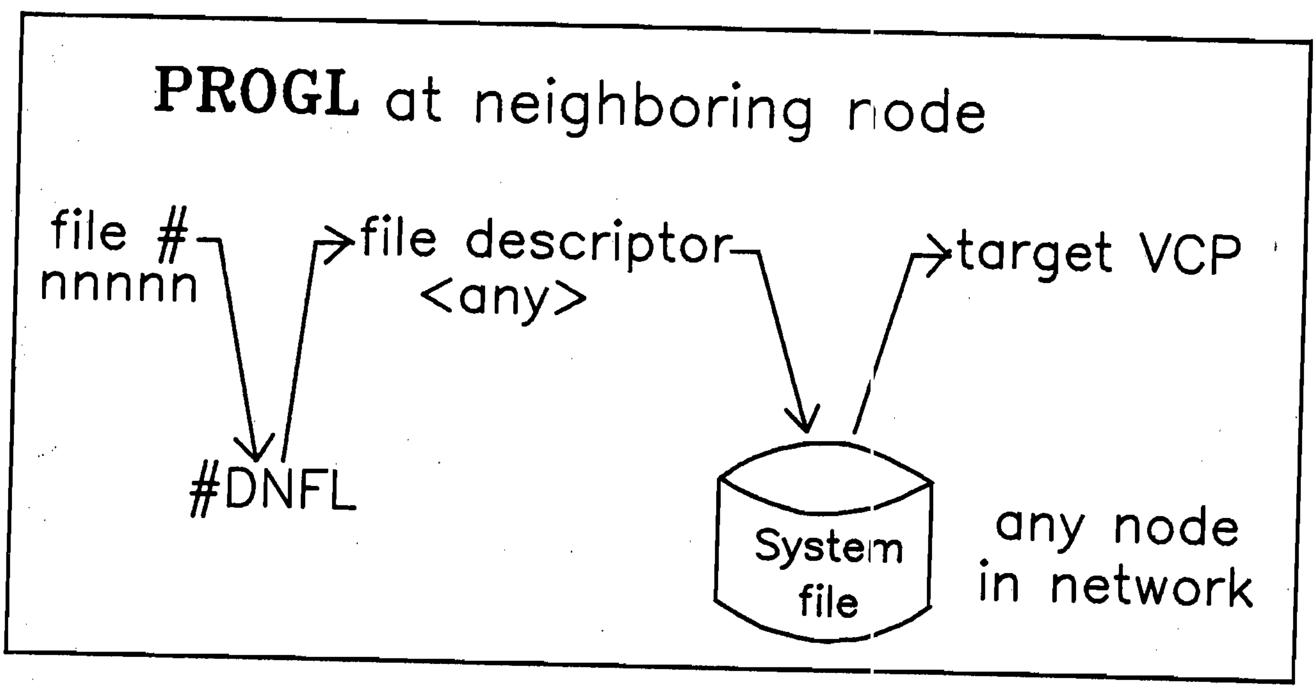
You can write your own subroutine called #DNFL and load it with PROGL. This allows you to translate the file number into any file descriptor and to boot from any node in the network.

References: System Gen. & Instl Manual, DS/1000-IV Manuals

DS/1000-IV INSTALLATION & BOOTUP PROCESS



OR



17.14 Boot Command String

DC = disc or CTD
DI = 248x integrated disc
MT = mag tape
DS = DS/1000-IV
RM = PROM module

File # -

DC or DI - bootable file #, starts at 0, 256 block chunks
DS - corresponds to file name on remote system
otherwise 0

Bus or drive address - HPIB bus address for DC or MT, drive address for DI (0 for fixed disc, 3 for microfloppy), otherwise 0. Defaults are:

DC 2
DI 0
MT 4

Unit # - unit number for discs, 1 for CTD, 0 otherwise. Default is 0.

Select code = select code of interface card. Defaults are:

DC 27
DI 32
DS 24
MT 27
RM 22

File name - for disc only (DC or DI), type 1 bootable system file (or boot command file for disc based systems).

Default file names are:

SYSTEM	if boot LU is a FMGR cartridge
BOOT.CMD::SYSTEM	if boot LU is a CI volume

References: Sys. Gen. & Instl Manual, A600/A700/A900 Comp. Ref Manual

BOOT COMMAND STRING

```
     [DC]
     [DI]
%B   [MT]   ff   b   u   sc   NAME
     [DS]
     [RM]
```

File # — ff

Bus or Drive addr — b

Unit # — u

Select code of interface card. — sc

File Name. — NAME

17.15 Bootup Examples

BOOTUP EXAMPLES

%BDC27 –

%BDS330024 –

%BMT04027 –

%BRM –

– boot from the 2nd file on CTD @ SC = 27 HPIB address = 0

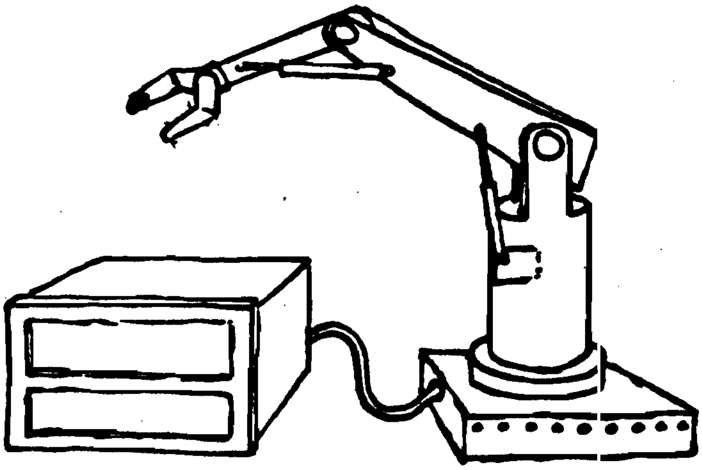
– Using BOOTEX boot the system file MERGED.SYS::SYSTEM on disc @ SC = 27 HPIB address = 0

17.16 System Verification and Backup

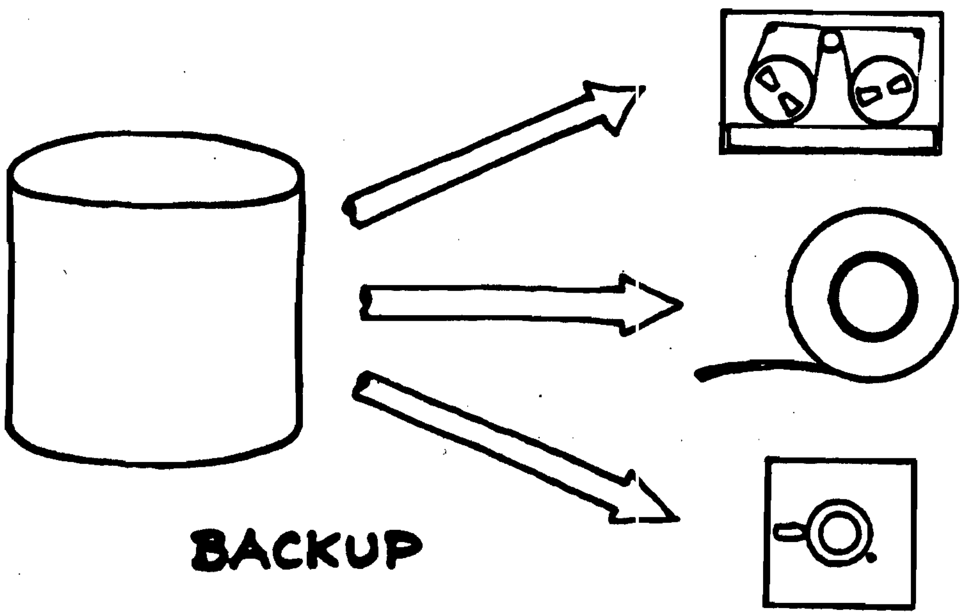
There are no standard tests for verfying a memory based system. You should run your application programs to see that they operate properly.

References: System Generation and Installation Manual

SYSTEM VERIFICATION AND BACKUP



RUN APPLICATION PROGRAMS



BACKUP

SYSTEM BACKUP AND MAINTENANCE

CHAPTER 18

Table of Contents

MODULE OBJECTIVES

1. Be able to determine a backup schedule for a system, making use of both physical and file backup.

2. Be able to perform file backup and incremental backup of system.

3. Be able to perform physical backup of system.

4. Be able to use the format utilities to maintain discs.

5. Be able to update system software.

SELF-EVALUATION QUESTIONS

18-1. Indicate whether the following features are describing file backup or physical backup:

selectively backup individual files
online or offline
faster for very full disc LUs
saves file features such as directories, ownership

18-2. Give 1 advantage and 1 disadvantage of incremental backup.

18-3. What is the largest area of the disc which can be backed-up at one time by ASAVE? What is the smallest area?

18-4. TRUE or FALSE: It is better not to verify backup tapes so that the backup procedure will take less time.

18-5. What kinds of media need to be formatted before use?

18-6. Give 2 differences between TF and FC.

18-7. What document describes update information as well as current revision codes for software?

18.1 D I S C B A C K U P

A backup is a copy of all or part of a disc onto another medium. The following media can be used for backup:

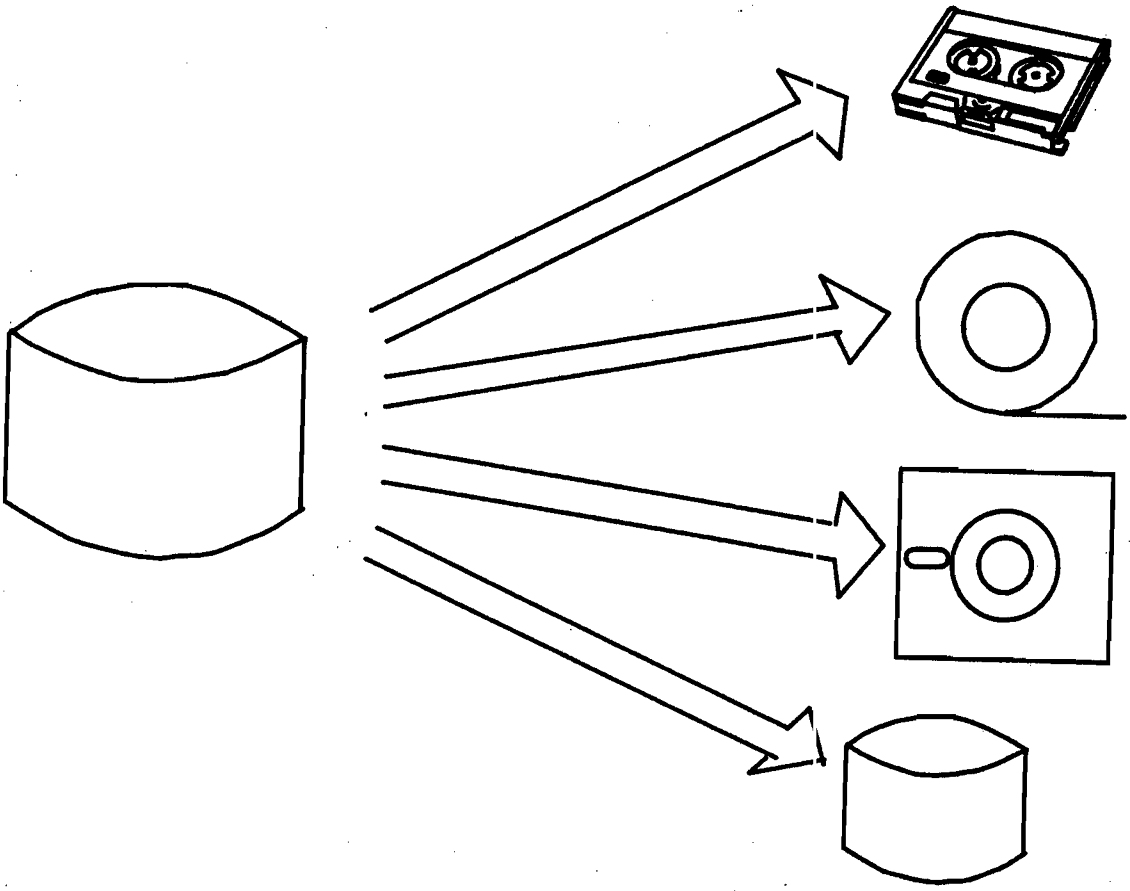
- CTD
- Magnetic Tape
- flexible disc
- hard disc (not really a backup media, but could be used)

A backup is done when there is information on the disc that cannot be recovered in a timely or cost-effective manner.

References: Utilities Manual

DISC BACKUP

What is a backup?



Why backup?

To recover data lost
due to system failure
or operator error.

18.2 File Backup vs. Physical Backup

File backup utilities save data on the disc on a per file basis. File structure and attributes are saved.

Physical backup saves the physical image of data on the disc, independent of the file system.

A typical backup scheme would make use of both file backup and physical backup.

References: Utilities Manual

FILE BACKUP
VS.
PHYSICAL BACKUP

FILE BACKUP	PHYSICAL BACKUP
SELECTIVELY BACKUP & RESTORE INDIVIDUAL FILES OR GROUPS OF FILES	BACKUP & RESTORE ENTIRE DISC LU OR DISC UNIT
IS FASTER FOR A SMALL NUMBER OF FILES	IS FASTER FOR VERY FULL DISC LUs
PERFORMED ONLINE ONLY	PERFORMED ONLINE OR OFFLINE
	REQUIRED TO RESTORE SYSTEM DISC

18.3 Backup Utilities

TF - Tape filer - can back up CI volumes; can backup FMGR cartridges (with some restrictions).

FC - File copy - FMGR cartridges only.

CI COpy command - for disc to disc. Note: no verify option.

Pushbutton - for CS80 discs with integrated cartridge tape drive.

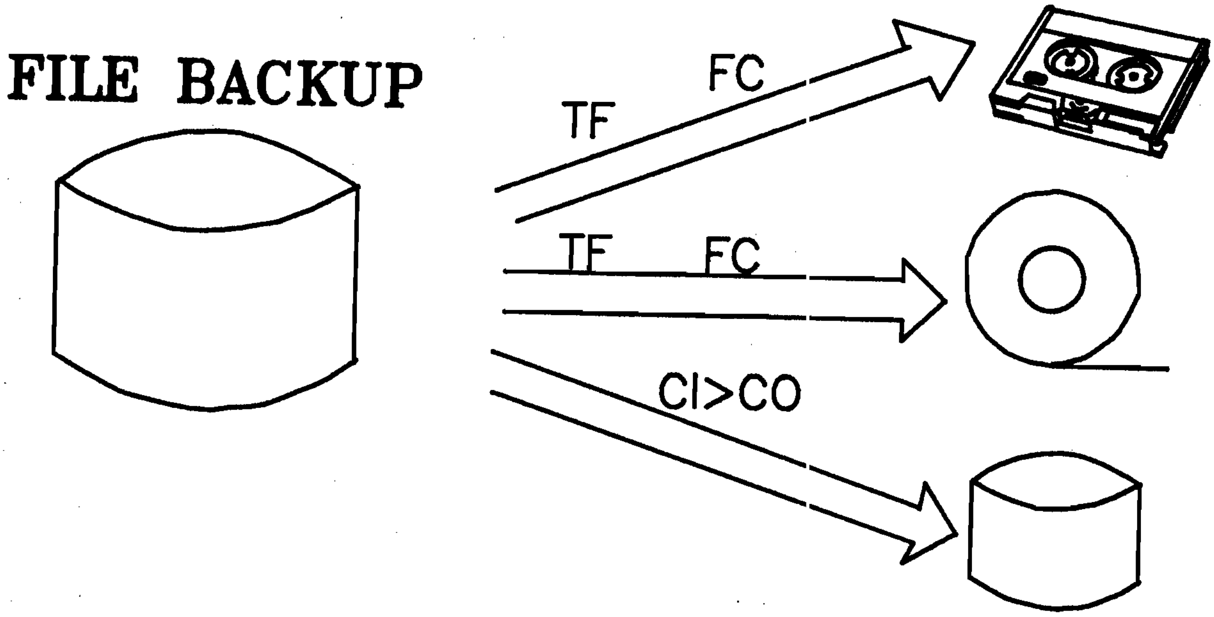
PBV - Pushbutton verify.

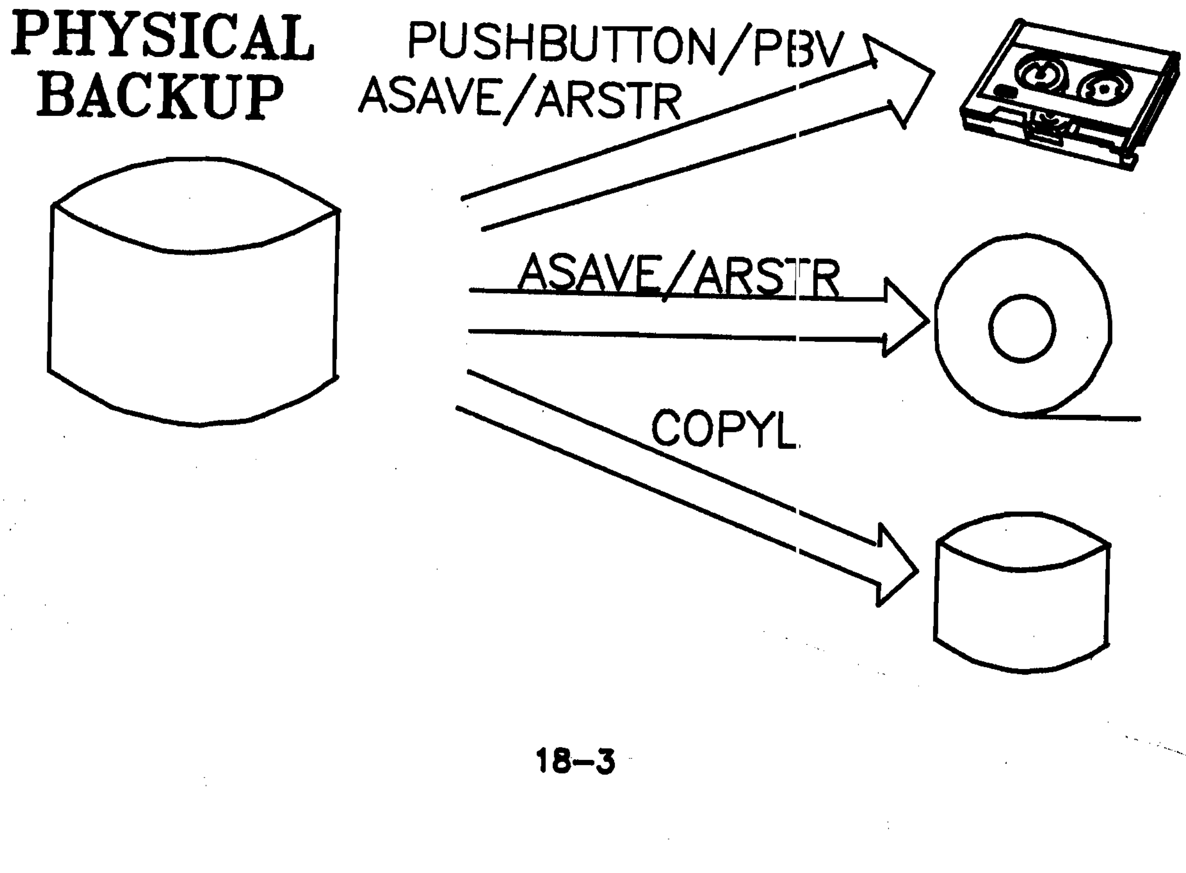
ASAVE /ARSTR - A-series physical save and restore.

COPYL - copies any disc to a like disc. Typically used only by HP Customer Engineer.

References: Utilities Manual

BACKUP UTILITIES





18.4 F I L E   B A C K U P

References: Utilities Manual

FILE BACKUP

18.5 TF - Tape Filer

Masked backup -- full file masking features for selecting source files and renaming destination files.

* Incremental backup -- select only those files that have been changed since the last incremental backup.

* Verify -- direct comparison of tape and disc data after copy has been completed.

* Append to tape -- files may be appended to an existing TF format tape.

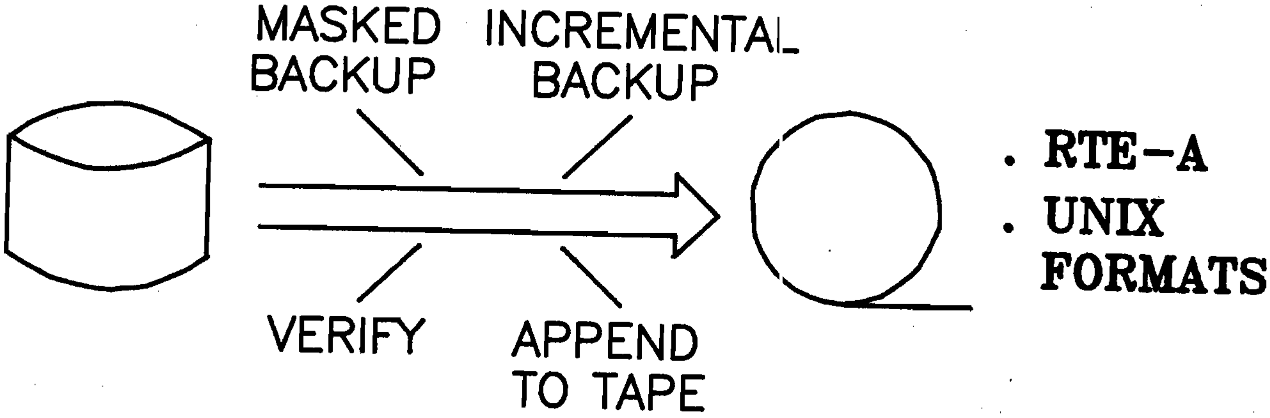
* Masked restore -- full file masking features for selecting tape files and renaming the restored copy.

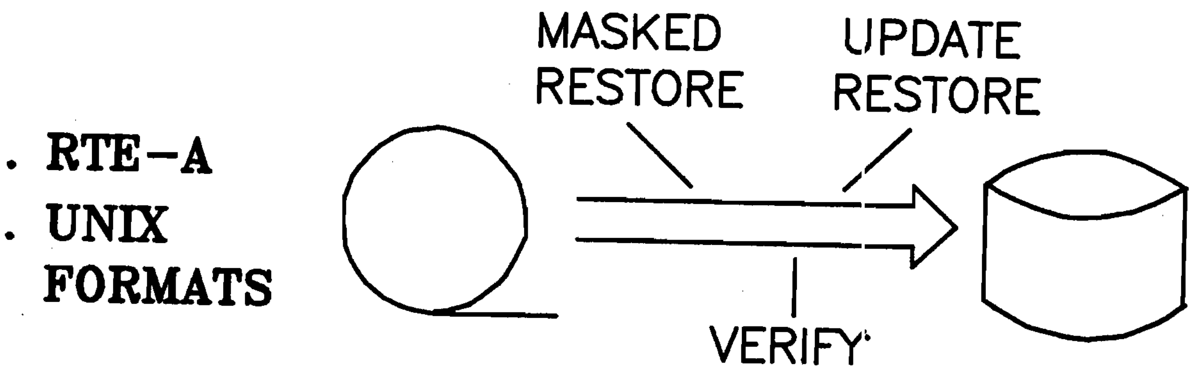
* RTE-A / UNIX formats -- files may be backed up or restored from UNIX TAR (Tape ARchive) format.

* TF runstring -- tf may be run with a single command in the runstring or may be executed interactively (ex command to exit).

References: Utilities Manual

TF – TAPE FILER





CI> tf <command>

CI> tf

tf: <command>

tf: <command>

tf: ex

18.6 TF Copy Command

<source> -- may be a file mask or a tape LU (LU 8 is often the mag tape drive, LU 24 the CTD for a CS80 disc).

<destination> -- may be a destination mask or a tape LU (must be opposite device from <source>).

<options> -- multiple options may be selected if appropriate:

- a -- append files to tape
- b -- suppress listing listing filenames as they are copied
- c -- clear backup bit after verifying copy to tape
- d -- replace duplicate files on disc
- k -- keep tape on line when finished
- u -- update: replace duplicate files, but only with newer version
- v -- verify copy
- x -- UNIX compatibility
- y -- yes: supresses prompting before overwriting or appending tape

References: Utilities Manual

TF COPY COMMAND

TF: CO <source><destination><options>

DISC TO TAPE:

TF: co spot.fun 8 a
TF: co @.ftn 24 v

TAPE TO DISC:

TF: co 8,,v
TF: co 24 {spot.@} /henry/@

MULTIPLE MASKS:

TF: co {/mine/@.@.s /yours/@.@.s} 8 avk
TF : co 24 {@.pas @.ftn} /sources/@ d

REMOTE ACCESS:

TF: co /elmer/@>2[user] 24
TF: co 8 @.@>1[manager] k

18.7 Other TF Commands

Title -- puts a title at the beginning of the tape. Up to 80 characters may be used.

Group -- begins a group of copy commands. No copying is done until after the group has been ended. Any number of copy commands mey be included in the group. If a file has been selected by more than one copy command, duplicate copies will be made.

eg -- end the group of copy commands. Copying is started immediately.

lh -- lists the tape header. The tape LU must be specified here.

dl -- directory list. Works the same as the CI command with the additional feature that tape directories may also be listed.

ll - specify list file

tr - transfer to command file

de - define defaults

ag - abort group copy

ex - exit

References: Utilities Manual

OTHER TF COMMANDS

```
TF: title project backup
TF: group
TF: co {@.pas @.ftn} 24 vy
TF: co {@.rel @.txt} 24 by
TF: eg
Copying AREA.PAS
    .
    .
    .
Copying ZOO.FTN
TF: lh 24

Tape format:  TF
Title:        project backup
Date:         Tue July 19,1983 3:15:21pm
Capacity:     16287 Kilobytes
Used:           114 Kilobytes

TF: dl 24
AREA.PAS
    .
    .
    .
ZOO.TXT
```

18.8 Ownership of Restored Directories

TF will also create the required directories as part of the restore if they do not all ready exist.

References: Utilities Manual

OWNERSHIP OF RESTORED DIRECTORIES

GENERAL USER:

DIRECTORIES ARE OWNED
BY USER DOING RESTORE

SUPER USER:

DIRECTORIES ARE OWNED
BY THEIR ORIGINAL OWNER

DIRECTORIES ARE RESTORED
TO THEIR ORIGINAL LU WHEN
POSSIBLE.

18.9 TF - Incremental Backup

An incremental backup would typically be done by appending delta backups onto the same tape as the full backup. The next full backup would start on a new tape. Alternately, you could do a full backup on one tape and one or more delta backups on additional tapes (depending on the size of your delta backup). These two methods are basically the same but require slightly different procedures to restore.

Advantages of incremental backup:

* Higher system availability since average (delta) backup time is faster.

* Less tape used on the average.

* Fewer tapes used since backups can be appended onto the same tape.

* Multiple versions of files can be accessed more conveniently (archiving).

Disadvantages:

* Takes longer to restore.

* Procedures are more difficult to understand.

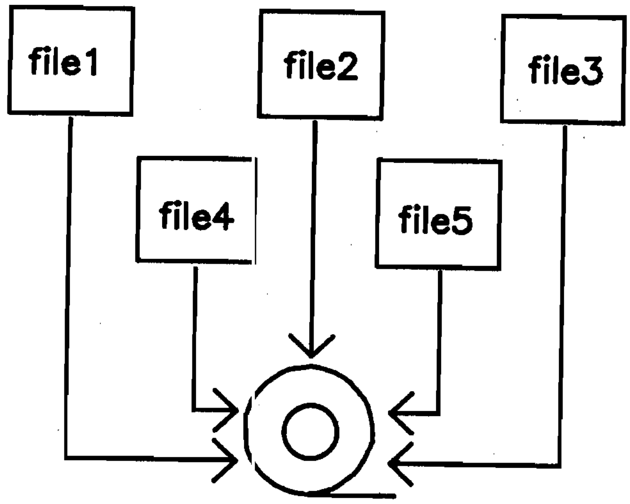
This procedure is not applicable to FMGR files.

References: Utilities Manual

TF INCREMENTAL BACKUP

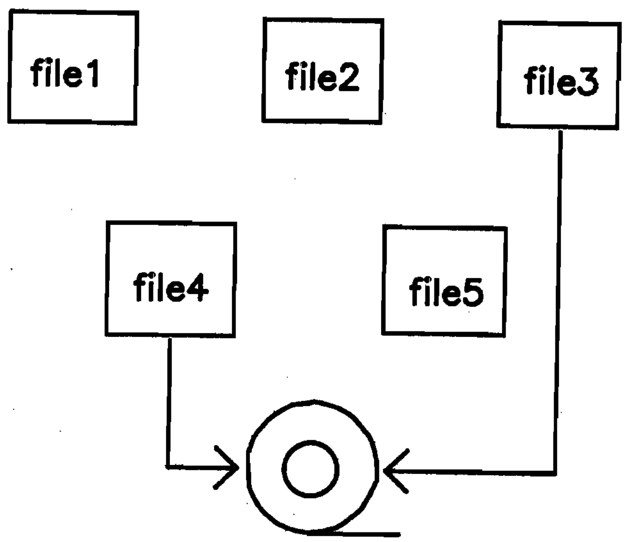
FULL BACKUP

PERIODICALLY
BACKUP ALL FILES



DELTA BACKUP

MORE FREQUENTLY
BACKUP ALL FILES
THAT HAVE CHANGED
SINCE LAST BACKUP



TO RESTORE

1. RESTORE MOST RECENT FULL BACKUP

2. RESTORE ALL SUBSEQUENT DELTA BACKUPS

18.10 Backup Bit

A bit associated with each file on a CI volume which indicates it needs to be backed up. This is useful for file backup only. There is no backup bit for FMGR files.

References: Utilities Manual

BACKUP BIT

ASSOCIATED WITH A FILE OR DIRECTORY

SET WHEN FILE:

- IS CREATED
- IS CHANGED
- IS MOVED TO ANOTHER DIRECTORY
- NAME, TYPE EXTENSION, OWNER OR PROTECTION BITS ARE CHANGED.

CLEARED WHEN:

- FILE COPIED TO TAPE BY TF WITH THE "C" OPTION SPECIFIED.

18.11 Incremental Backup Procedure

This example shows an incremental backup of everything under the global directory /leslie. The C option clears the backup bit and always does a verify. The A option appends the backup onto the tape all ready containing a backup. The same source mask must be used every time the incremental backup is done or the results may be confusing.

In the example shown, TF will automatically replace duplicate files from the delta backup if the files were created since the restore procedure began.

Note: if the incremental backup is not done by appending delta backups on the same tape, then a CO command is needed for each tape. Also the D (replace duplicate files) option must be specified for the delta backups.

References: Utilities Manual

INCREMENTAL BACKUP PROCEDURE

WEEKLY

CI> tf co /leslie.dir 8 c

DAILY

CI> tf co /leslie.dir.b 8 ac

TO RESTORE:

CI> tf co 8

18.12 System-wide Backup & Restore

```
tf co /@.@    <tape>  v   => copy all non-FMGR files
tf co @.@.e   <tape>  v   => copy all files including FMGR
tf co /@.@    <tape>  c   => full backup of all non-FMGR files,
                             clearing backup bit
tf co /@.@.b <tape>   ac  => copy all files whose backup bit is set,
                             clearing backup bit
tf co <tape>              => restores all the above examples
```

Note: Problems may arise if the FMGR file or cartridge names contain certain restricted characters. If so, you can use FC for FMGR file backup.

For successful system backup:

* Keep system time accurate!

* Use transfer files for backup and restore to avoid errors!

* Only one person should clear the backup bit for a given set of files!

* Keep users from accessing files during system backup!

* Always verify backups and restores!

References: Utilities Manual

SYSTEM-WIDE FILE BACKUP PROCEDURE

BACKUP

CI> tf co /@.@ <tape> v

CI> tf co @.@.e <tape> v

INCREMENTAL BACKUP

CI> tf co /@.@ <tape> c

CI> tf co /@.@.b <tape> ac

RESTORE

CI> tf co <tape>

18.13 FC - File Copy

Copies FMGR files to and from FMGR cartidges only. FC is used to copy update software supplied on mag tape or CTD. It could also be used to copy files to and fron RTE-6/VM and RTE-IVB systems.

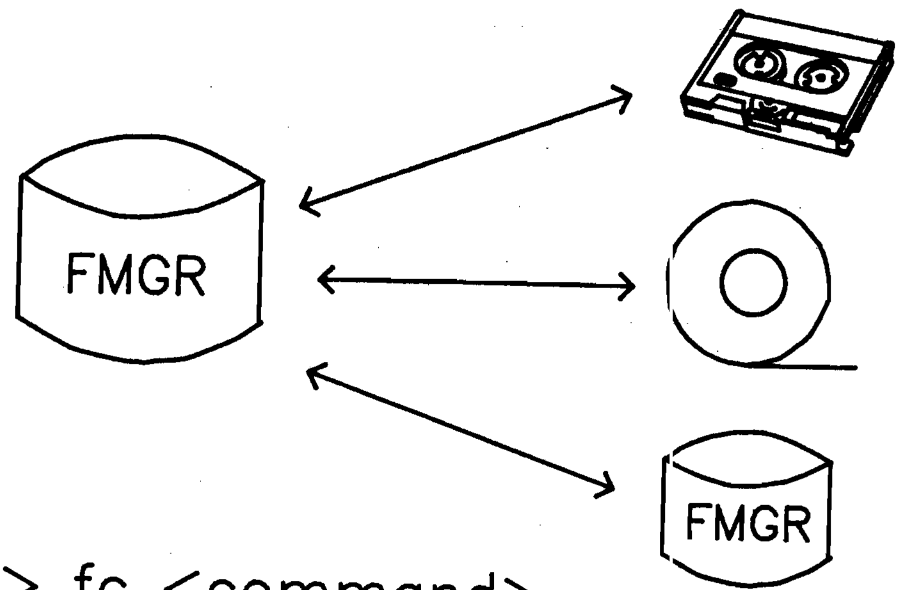
FC has a command set similar to TF, except that it does not allow file masking (except use of dashes for wildcard characters).

Some commonly used FC commands:

CO - copy files
DE - set default parameters
GR - begin group copy
EG - end group copy
AG - abort group copy
LL - specify list file
DL - directory list of files on disc or tape
LH - list header on tape
TI - specify title to be used on tape
TR - transfer to command file
EX - exit
? - help

References: Utilities Manual

FC - FILE COPY



CI> fc <command>
CI> fc
FC: <command>
FC: <command>
FC: ex

SOME FC COMMANDS

CO
DE
GR / EG/ AG
LL
DL
LH
TI
TR
EX
?

18.14 FC Copy Command

<source> - file or files to be copied. Can also be a tape LU or FMGR cartridge. A list of names can be enclosed in braces. Wildcard characters (dashes) can be used.

<destination> - Tape LU, FMGR cartridge or a filename.

<options> - some commonly used options are:

B - suppress listing of files copied
C - clear destination cartridge
D - replace duplicate files
E - Eliminate extents
P - purge source files after copy
V - verify copy

Multiple options may be used.

References: Utilities Manual

FC COPY COMMAND

FC: co <source><dest><options><file1><file2><msc>

EXAMPLES:

FC: CO ::VB –8 v

FC: CO {games,teams}::VB –24

FC: CO {::WS,::SS} –8 v

FC: CO –8 ::LN v

FC: CO –24{copy,me2} ::cr d

FC: CO –8 {::A2,::A3} v

18.15 PHYSICAL BACKUP

You must have some form of physical backup or copy of your boot disc LU. A physical backup on a disc or floppy would be a bootable system. A physical backup on magtape or CTD would have to be downloaded to the disc before it can be booted. This could be done in one of the following ways:

* Download with the pushbutton restore feature of CS80 discs.
* Boot a memory based system which contains ARTSR, the physical restore utility, then restore the tape to disc.

To be useful, this physical backup must be able to restore enough of the system to boot up, run CI and bring other programs and files onto the system.

References: Utilities Manual

PHYSICAL BACKUP

A PHYSICAL BACKUP IS:

A COPY OF YOUR SYSTEM ON
A BOOTABLE MEDIA

OR

A COPY OF YOUR SYSTEM ON
A MEDIA WHICH CAN BE
DOWNLOADED TO THE SYSTEM
DISC WITH AN OFFLINE
UTILITY

THE PHYSICAL BACKUP MUST CONTAIN THE FOLLOWING:

bootable BOOTEX	CI
snap file	DRTR
system file	DL
boot command file	TF
welcome file	LINK
swap file	PROMT (for VC+)
/programs and /system	FORMC or FORMF

18.16 ASAVE and ARSTR

ASAVE is the A-series physical backup utility and ARSTR is the corresponding restore utility. Typically, ASAVE would be used online to backup the disc. For system restoration, ARSTR would be used offline. It could also be used online to restore the rest of the disc.

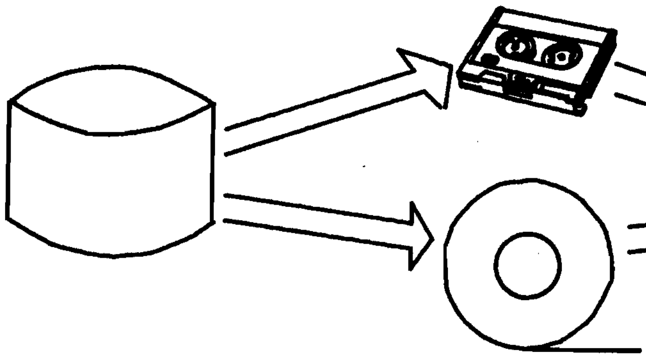
ARTSR can only restore a tape to a disc with the same number of blocks per track that it was saved from.

ASAVE and ARSTR can be run interactively or all commands can be included in the runstring.

To run ARSTR offline, you must prepare a bootable memory based system which contains the ARSTR program. This system must be available on some media other than your system disc. This memory based system is part of the physical backup! RTE-A systems on magnetic tape include a bootable memory-based ARSTR system.

References: Utilities Manual

ASAVE & ARSTR



- ONLINE OR OFFLINE
- SAVE LU, GROUP OF LUs, ENTIRE DISC
- MULTI-TAPE BACKUPS
- VERIFICATION

CI> asave <commands>
CI> asave
ASAVE: <command>
ASAVE: ex
CI> arstr <commands>
CI> arstr
ARSTR: <command>
ARSTR: ex

18.17 ASAVE Commands

Some ASAVE commands:

TA = assigns tape LU
TI = specify title to be included in tape header
SA = save the specified LU(s) to tape; The VE option verifies;
The UN option will back up all LUs on the disc unit.
RW = rewind

In the first part of the example, one disc LU is backed up and verified. A header is included with the file on tape which contains: the format, date, time, file #, tape #, LU, title and other information.

The second part of the example will backup all the LUs on the disc unit which contains LU 16 and verify. Each LU will be saved in a seperate file on the tape with its own header information.

References: Utilities Manual

ASAVE COMMANDS

```
ASAVE: ta  24
Tape LU = 24
ASAVE: ti  SYSTEM BACKUP
Title = SYSTEM BACKUP
ASAVE: sa  12  ve
Disc LU(s) to be saved:
12
   Created using: ASAVE
        .
        .
   Verifying tape
        .
        .

ASAVE: ti  ENTIRE SYSTEM BACKUP
Title = ENTIRE SYSTEM BACKUP
ASAVE: sa  16  un  ve
Disc LU(s) to be saved:
16
17
18
19
20
   Created using ASAVE
        .
        .
   Verifying tape
        .
   ASAVE: rw
   ASAVE: ex
```

18.18 ARSTR Commands

Most of the ARSTR commands are the same as ASAVE. Some ARSTR commands:

* TA = Assign tape LU

* LH = Display tape header for file number specified

* RE = Restore file(s) to LU(s) (file#:LU); The VE option verifies the restores. The UN option restores the entire unit starting with the file # specified.

In the first part of the example, the file header is listed for the first file on the tape. Then the first file is restored to LU 12 and verified.

The second part of the example will restore the entire disc unit (saved in the ASAVE example) starting with file 2. When the UN option is specified, ARSTR will only restore the specified save file and any subsequent files if they were from the same disc unit and were saved in the same save operation.

References: Utilities Manual

ARSTR COMMANDS

```
ARSTR: ta  24
Tape Lu = 24
ARSTR: lh  1
  Created using: ASAVE
        .
        .
  Title: SYSTEM BACKUP
ARSTR:  re 1:12  ve
  Created using: ASAVE
        .
        .
  Title: SYSTEM BACKUP
Restore of file 1 to disc LU 12 complete.
Verifying restores from tape.
ARSTR: re 2 un ve
  Created using: ASAVE
        .
        .
  Title:ENTIRE SYSTEM BACKUP
Restore of file 2 to disc LU 16 complete
Created using: ASAVE
        .
        .
Restore of file 2 to disc LU 16 complete.
Verifying restores from tape.
        .
        .
        .
ARSTR: rw
ARSTR: ex
```

18.19 Pushbutton Backup and Restore

Pushbutton backup and restore can be done only on CS80 discs with integrated Cartridge tape drives. This operation uses only the disc drive, not the system. The disc will not be accessible to the system while either operation is in progress.

To backup or restore:

Remove the front panel over the CTD unit
Insert the CTD and wait for it to load
Push either the appropriate save or restore switch and release
When the busy light begins flashing, push the switch again
(must be while the light is still flashing)

The A-series backup utilities will always put an EOF (end of file) at the start of the tape, so that these cannot accidentally be used for a pushbutton restore.

If you are using the primary disc configuration:

An entire 7908 can be saved on a 150 ft. tape. A 7911 or 7912 can be saved on a 600 ft. tape. A 7914 requires two 600 ft. tapes.

PBV can only be run offline. The example shows how to run PBV using the memory-based PBV system provided with your software. The system is provided as the first file on a CTD tape, so to boot it you would type the following at the VCP:

```
VCP> %bdc0blsc
```

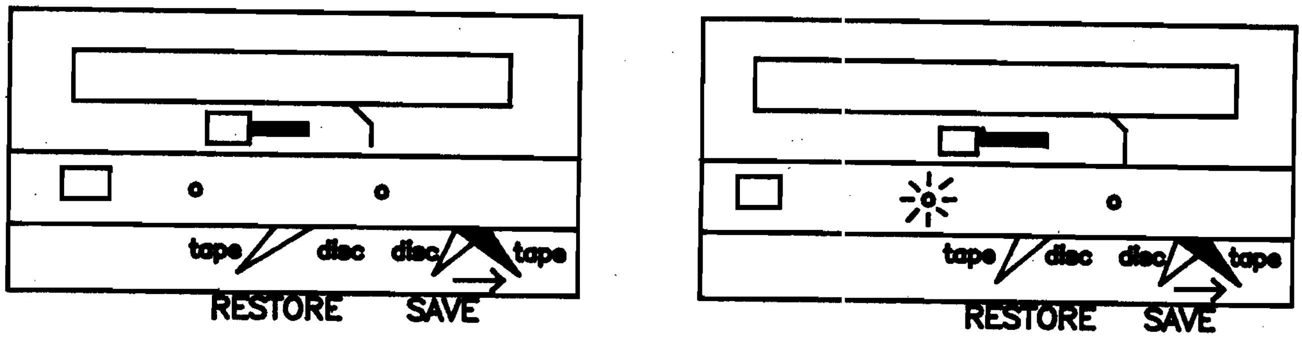
(Remember what busc is ?). COMND is the program included in the system to schedule other programs. It is similar to FMGR and requires commas as delimiters.

The PBV memory based system comes with a number of disc and CTD LUs all ready gen'ed in. If these do not match your system LUs, you will need to build your own PBV memory based system.

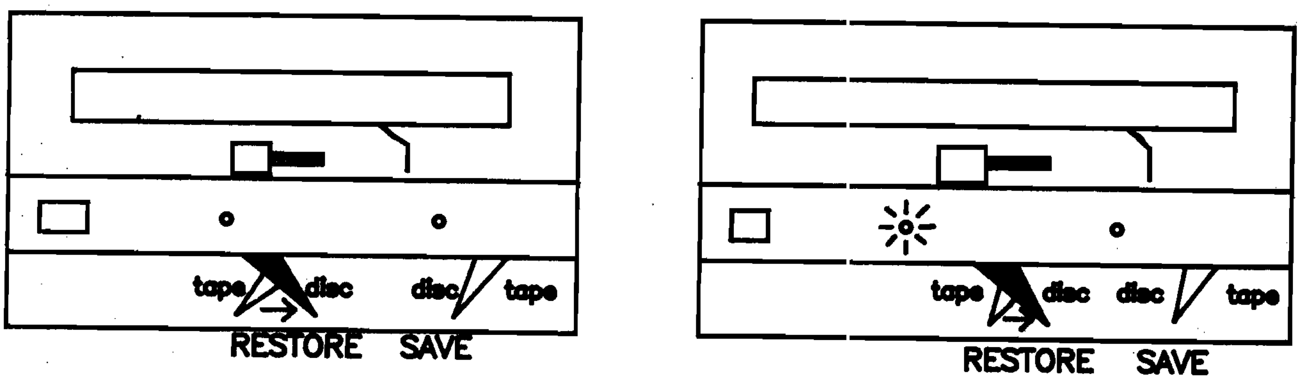
The type of the source LU (disc or tape) is used to determine whether a save or restore is being verified. This is to determine how much of the disc or tape to verify.

References: Utilities Manual, CS80 Disc manuals

PUSHBUTTON BACKUP AND RESTORE



BACKUP



RESTORE

PUSHBUTTON VERIFY (OFFLINE)

comnd: ru,pbv,<source lu>,<destination lu>

18.20 System Backup Strategy

Some keys to successful backup:

- Stick to your backup schedule
- Backup when there are no other users on system
- Label tapes with date, time, contents and type of backup
- Store backups in a safe place
- Keep system time accurate
- Use transfer files
- Always verify your backups !

References: Utilities Manual

SYSTEM BACKUP STRATEGY

* **AT RE-GENERATION**
  RE-BUILD MEMORY BASED ARSTR OR PBV

* **AFTER BOOTUP**
  BACKUP SYSTEM AND THE BOOTEX FILE
  USING ASAVE OR PUSHBUTTON SAVE

* **CHOOSE A BACKUP METHOD BASED ON YOUR NEEDS**

* **CHOOSE A BACKUP SCHEDULE BASED ON SENSITIVITY TO DATA LOSS**

* TO RESTORE FROM BACKUP

  RESTORE SYSTEM FROM PHYSICAL BACKUP (IF NEEDED)

  RESTORE MOST RECENT TOTAL BACKUP

  RESTORE INTERMEDIATE OR INCREMENTAL BACKUPS

18.21 D I S C   F O R M A T T I N G   O P E R A T I O N S .

Verify - Non-destructive operation to determine if the data on the disc is internally consistent and to identify bad blocks or tracks.

Spare - Mark bad blocks or tracks and map to spare blocks or tracks.

Format or Reformat - On a hard disc - allows a user to clear all accumulated bad tracks. On floppy or CTD - initializes media for use and spares bad blocks.

Initialize - Clean up spare track pool and prepare LU for use by sparing any bad tracks.

References: Utilities Manual

DISC FORMATTING OPERATIONS

VERIFY

SPARE

FORMAT

REFORMAT

INITIALIZE

18.22 Disc Format Utilities

FORMT comes as an memory based system. FORMC and FORMF should be available on the physical backup of the system. Generally, FORMT would only be used for 7906/7920/7925.

References: Utilities Manual

DISC FORMAT UTILITIES

FORMC	ONLINE	CS80 DISC & CTD	VERIFY SPARE FORMAT
FORMF	ONLINE	9121 9133 A/B 9134 A/B 9895 2480 MODEL 6+ BUILT-IN FLOPPY	VERIFY FORMAT
FORMT	OFFLINE	9121 9133A 9134A 9895 7906 7920 7925	VERIFY FORMAT VERIFY SPARE REFORMAT INITIALIZE

18.23 FORMC - CS80 Discs

FORMC is for online use only. If offline formatting is required, use the CS80 disc exerciser.

Bad blocks are not very common on a CS80 disc since the media is sealed. Typically there is no reason for the user to format the disc. To spare blocks, you would specify the bad track. FORMC will spare any bad blocks that it finds.

CTDs must be formatted before use unless preformatted CTDs are purchased. Bad blocks on the CTD are spared as part of the format operation.

References: Utilities Manual

FORMC
CS80 DISCS

CI> formc <list lu> ve <media lu> <start> <number>

CI> formc <list lu> fo <media lu> <interleave>

CI> formc <list lu> sp <media lu> <track>

DISC

VERIFY TO DETERMINE BAD TRACKS

SPARE BAD BLOCKS – SPARES ALLOCATED BY DISC

CTD's

FORMAT BEFORE USE

FORMATTING SPARES BAD BLOCKS

18.24 FORMF - Floppy Disc, 2480 and Winchester Disc

Since the floppy discs have exposed surfaces, bad blocks may occur. The formatting process will spare bad blocks as they are found. The amount of space available for spares is the reserved track area on the floppy (remember disc configuration ?). If the amount of spares needed exceeds this area, then the disc must be replaced.

The other discs are sealed, so bad blocks are uncommon.

References: Utilities Manual

FORMF

FLOPPY DISCS, 2480 & WINCHESTER DISC

Cl>formf ve <disc lu>

Cl>formf fo <disc lu> <interleave>

FLOPPIES

FORMAT BEFORE USE

VERIFY TO DETERMINE IF THERE ARE BAD TRACKS

FORMATTING SPARES BAD BLOCKS

2480/9133 & 9134 WINCHESTER

VERIFY TO DETERMINE IF THERE ARE BAD TRACKS

FORMATTING "SPARES" BAD BLOCKS

18.25 FORMT - 7906, 7920, 7925 Discs

Since these discs have removable platters, bad blocks are more likely to occur than for sealed discs. Like the CS80's, there is typically no reason for the user to reformat the disc. If done, the initialize must be done after the reformat.

There is a bootable FORMT system on the primary system, in a file called FORMT::16. To boot:

```
VCP> %bdc27formt
```

References: Utilities Manual

FORMT

7906, 7920, 7925 DISCS

ru,formt, <list lu> ve <disc lu>

ru,formt, <list lu> sp <disc lu> <track>

ru,formt, <list lu> re <disc lu>

ru,formt, <list lu> in <disc lu>

VERIFY TO DETERMINE BAD TRACKS
SPARE BAD TRACKS

18.26 O T H E R   U S E F U L   U T I L I T I E S

OTHER USEFUL UTILITIES

18.27 Volume Free Space -- FREES

Free space % -- gives an indication of how full the volume is.

Largest free space % -- gives an indication of how fragmented the free space is. The lower this percentage is, the more effective will be a volume pack.

FREES cannot be used for FMGR cartridges.

References: System Generation and Installation Manual

VOLUME FREE SPACE FREES

Cl> frees 22

Total blocks: 45072

Free blocks: 37054

Free space is 82% of total space

Largest free space: 36975

Largest free space is 99% of total free space

18.28 Volume Packing -- FPACK

For CI volumes only. For FMGR cartridges, run FMGR and use the PK command.

1. FPACK scans the entire volume to determine the location of all free space and files that might be copied to another location.

2. Files in the highest address spaces are moved into holes in the lowest address spaces on a first fit basis. The file (and all extents) must fit into a hole in a lower address space or else it is not moved.

3. The names of the last 10 files on the disc is printed on the terminal (e.g., X, E, D, F, ...). These are the files that must be removed or purged in order to increase the size of the largest free space on the volume.

File "X" -- not moved for one of the following reasons:

1. Directories are never moved by FPACK.
2. Open files are not moved.
3. Active run files (type 6) are not moved.
4. The system swap file is never moved by FPACK.

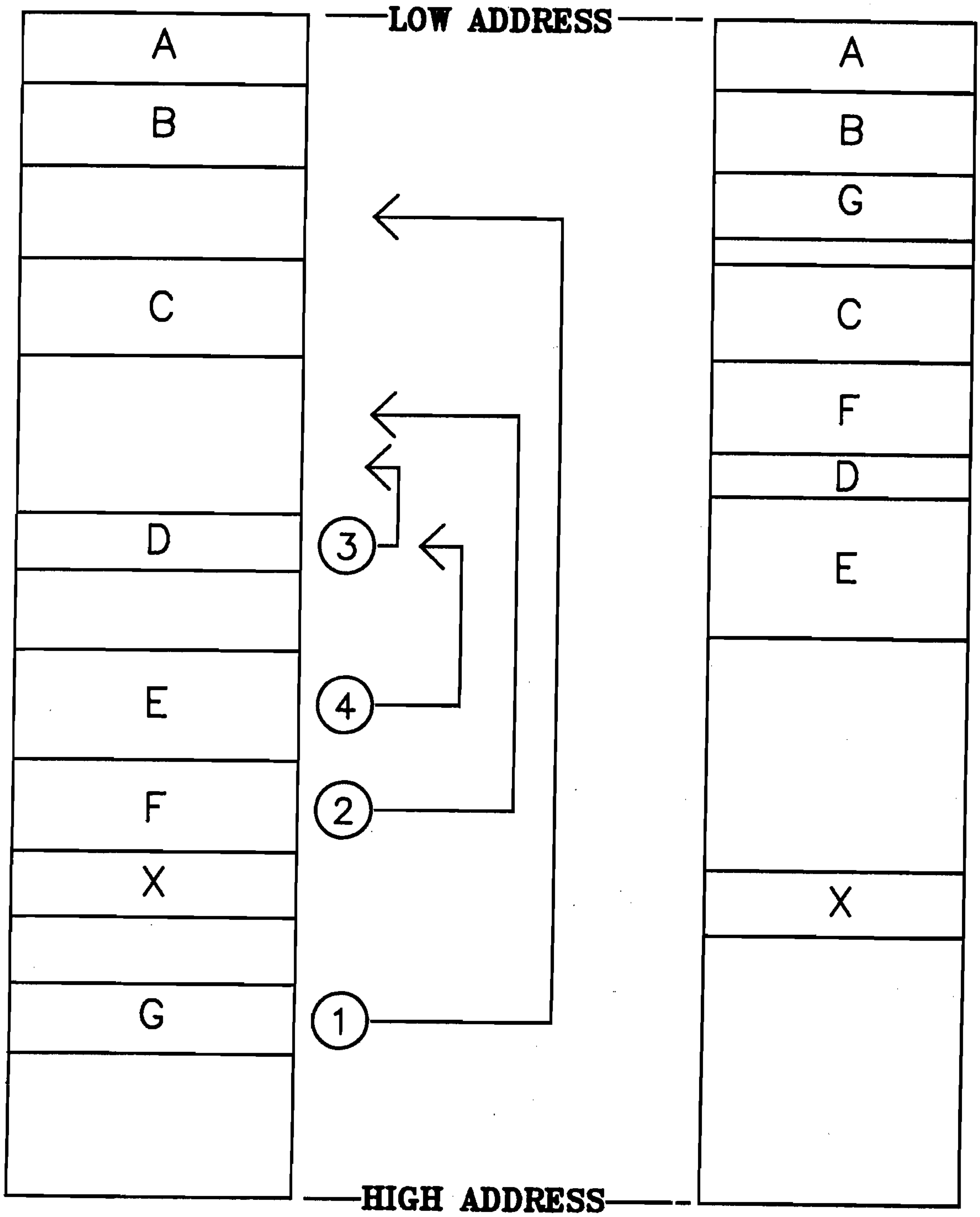
VC+ NOTE:

The user running FPACK must have read/write access to the file and the directory containing the file in order for FPACK to copy it.

References: Utilities Manual

VOLUME PACKING
FPACK

CI>fpack 22



18.29 File System Verification - FVERI

FVERI scans the directories and table structures of a hierarchical file system disc and reports any inconsistencies. If any inconsistencies are found, they are reported with an appropriate message and a number indicating the severity of the problem. For FVERI to give accurate results, no one should access the disc while it is running. FVERI will not verify FMGR cartridges.

FVERI can be used after a system crash to verify the file system is intact. It can also be run from time to time to check the integrity of the file system.

Recovery: Minor errors (low numbers) can be ignored or corrected by copying the affected files to another disc or tape, purging them and restoring them. For severe errors, the entire disc volume should be backed up, initialized and restored.

References: Utilities Manual

FILE SYSTEM VERIFICATION FVERI

```
CI> fveri 22
Verifying LU 22
    .
    .
    .
```

ERROR MESSAGES

less severe	(0) description
↓	(1) description
	.
	.
	.
	(8) description
more severe	(9) description

18.30 File System Conversion - FSCON

Before doing the conversion, FSCON will check for the following requirements:

1. Must be sufficient free space at the end of the old cartridge to create the new directory and free space table. If this is a problem, try packing the FMGR cartridge (using the FMGR PK command).

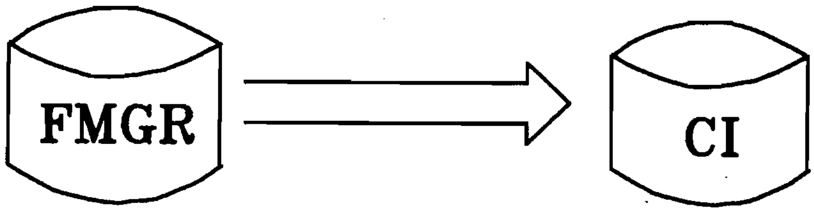
2. Total size of the cartridge cannot exceed 128K blocks.

3. Disc must be dismounted before conversion - no open files, active type 6 or swap files.

Since the period and slash are illegal characters for filenames under the hierarchical file system, filenames with these will be changed. The cartridge reference designation becomes the new directory name.

References: Utilities Manual

FSCON
FILE SYSTEM CONVERSION



CI> dc <lu>
CI> fscon <lu>

FI.LE::CR → FI~LE::CR
/PROG::CR → ||PROG::CR

TIMESTAMPS ← CURRENT TIME
PROTECTION ← RW/RW
BACKUP BIT ← SET
TYPE EXTENSION ← BLANK

18.31 SOFTWARE UPDATES

SOFTWARE UPDATES

18.32 Primary System

The primary is a part of the software product. It allows you to start using your new system right away and also provides a starting point from which to create your customized systems.

References: Primary System Installation Manual

FILE SYSTEM VERIFICATION FVERI

```
CI> fveri 22
Verifying LU 22
    .
    .
    .
```

ERROR MESSAGES

less severe

↓

more severe

(0) description

(1) description

.

.

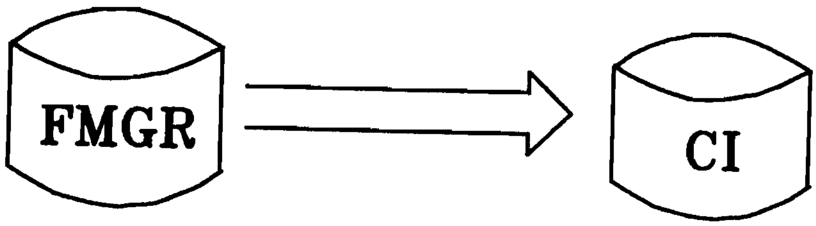
.

(8) description

(9) description

18.31 SOFTWARE UPDATES

FSCON
FILE SYSTEM CONVERSION



CI> dc <lu>
CI> fscon <lu>

FI.LE::CR ⟶ FI~LE::CR
/PROG::CR ⟶ |PROG::CR

TIMESTAMPS ⟵ CURRENT TIME
PROTECTION ⟵ RW/RW
BACKUP BIT ⟵ SET
TYPE EXTENSION ⟵ BLANK

18.32 Primary System

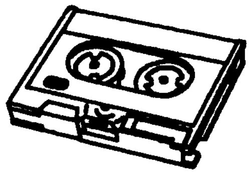
The primary is a part of the software product. It allows you to start using your new system right away and also provides a starting point from which to create your customized systems.

References: Primary System Installation Manual

SOFTWARE UPDATES

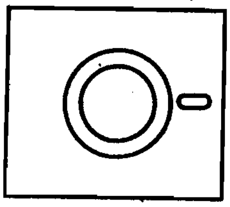
PRIMARY SYSTEM

A PREGENERATED SYSTEM WITH
* System Utilities
* Most supported I/O devices



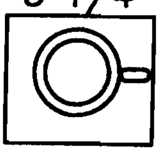
PUSHBUTTON BACKUP FORMAT
BOOTABLE PBV SYSTEM

8"

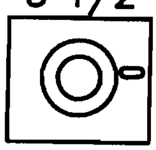


MOUNTABLE FMGR CARTRIDGES
BOOTABLE FMGR CARTRIDGE

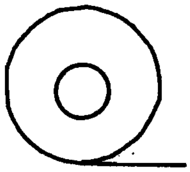
5 1/4



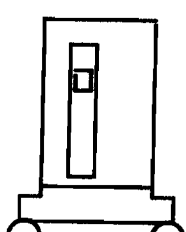
3 1/2



MOUNTABLE FMGR CARTRIDGES
BOOTABLE FCO SYSTEM



ASAVE FORMAT
BOOTABLE ARSTR SYSTEM



BOOTABLE SYSTEM ON HARD DISC

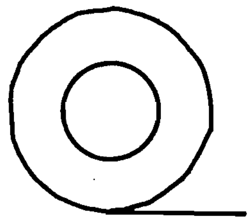
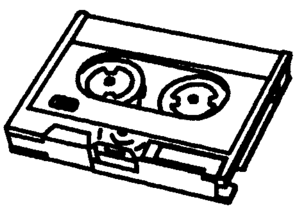
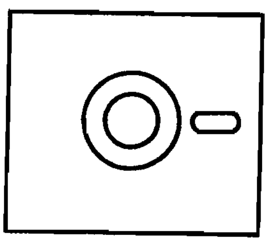
18.33 Update Options

VCP bootable means that these files are loaded directly from tape into memory, then executed by following the instructions in the appropriate diagnostic manual.

Note that updates software does not typically include a primary system.

References: Software Update Notice

UPDATE OPTIONS

MEDIA OPTION	FORMAT
 800 bpi 1600 bpi	FC one product per tape
	FC multiple products per tape VCP bootable (diagnostics)
 8" 5 1/4" 3 1/2"	Mountable FMGR cartridges

CONTAINS ALL FILES OF UPDATED PRODUCTS

IF 1 OR MORE FILES ON A MEDIA PART CHANGE, THEN ALL FILES ON THAT MEDIA PART ARE INCLUDED

18.34 Update Procedure

Regenerate system only if any of the files used in the system generation have changed. To update programs, you must reload them using LINK, not just transport. When updating, check revision codes against those in the Software Update Notice (SUN). The SUN is distributed periodically to customers who have support services (see next chapter).

References: Software Update Notice

UPDATE PROCEDURE

* READ SUN TO BE INFORMED ABOUT FIXES INCLUDED WITH YOUR UPDATE OR CHANGES REQUIRED IN YOUR SYSTEM

* BACKUP CURRENT SYSTEM – VERIFY BACKUP

* COPY UPDATED FILES TO DISC, VERIFY TRANSFER, PURGE OLD FILES

* REGENERATE SYSTEM AND CHECK REVISION CODES

* RELOAD UPDATED PROGRAMS AND CHECK REVISION CODES

* BOOT NEW SYSTEM AND VERIFY SYSTEM OPERATION

* BACKUP UPDATED SYSTEM

18.35 Diagnostics

These diagnostics packages are part of the standard RTE-A product.

DIAGNOSTICS

24612A DIAGNOSTIC PACKAGE

CHECKS OUT:

- CPU
- MEMORY
- MOST STANDARD INTERFACES
- MEASUREMENT &: CONTROL INTERFACES

24398A DIAGNOSTICS PACKAGE

CHECKS OUT:

- CS80 DISCS
- 7906H, 7910H
- MAG TAPE UNIT

SUPPORT SERVICES

CHAPTER 19

Table of Contents

MODULES OBJECTIVES

1. To become aware of the many support services offered.

2. Know the procedure of bug reporting and where to turn for updated software materials and information.

SELF-EVALUATION QUESTIONS

19-1. What is the difference between the basic and standard maintenance services?

19-2. Match the following features with the software support service which provide them:

________	Software Status Bulletin	A. CSS
________	On-site SE assistance	B. SSS
________	Manual updates	C. SNS
________	Software/Firmware updates	D. MUS
________	Software Update Notice	
________	PICS	

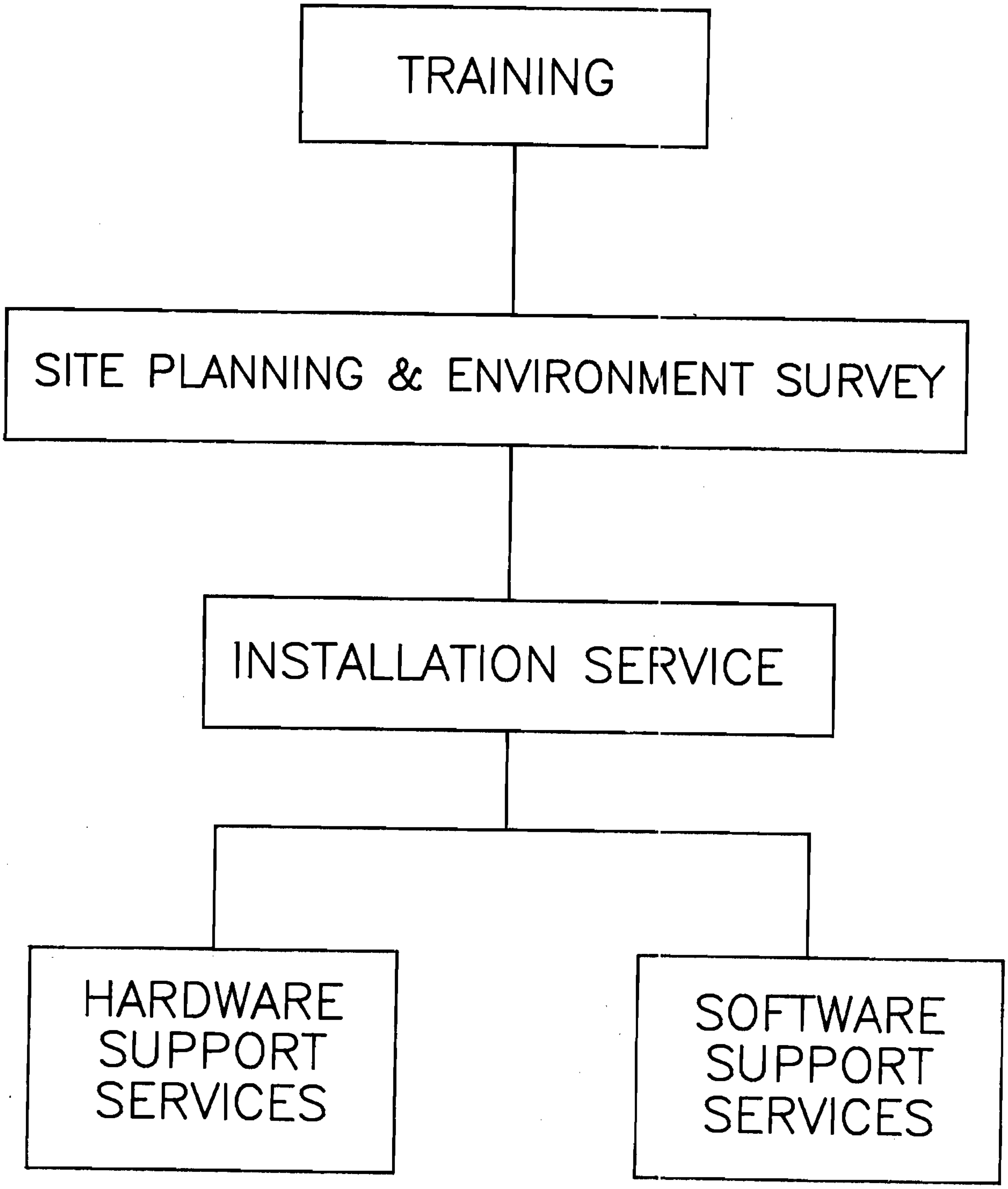
19.1 Field Services

Site planning service - A site planning specialist advises the customer on all technical matters relating to site planning, preparation and installation.

Site environmental survey - Prior to system installation, it serves to verify that the changes recommended during the site planning visit were completed.

Generally, installation is included with system products.

FIELD SERVICES



19.2 Hardware Support Services

There are 2 types of system maintenance services: standard and basic. The standard service provides same-day response and rapid repair of a failed system. The basic service is a lower cost service with next-day response time being its major feature.

Both services provide the following features:

* Account-assigned Customer engineer

* Regularly scheduled preventive maintenance

* Engineering changes (when required to fix problems)

* Work to completion

* Add-on installation for additional system equipment

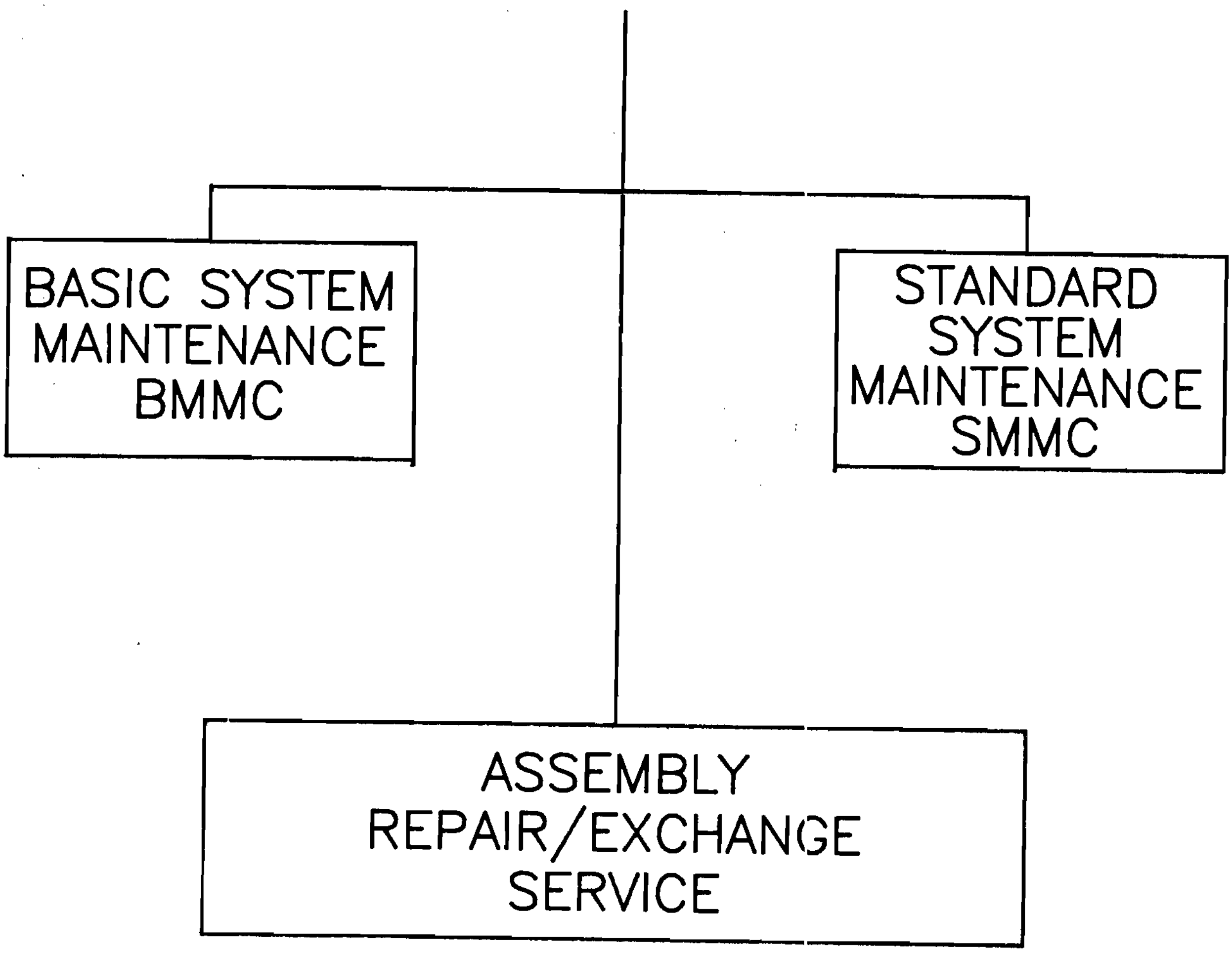
* Site environmental survey

* BMMC - Basic monthly maintenance charge

* SMMC - Standard monthly maintenance charge

Assembly repair/exchange service - For customers not receiving maintenance service, this service replaces the customer's defective assembly with a refurbished one.

HARDWARE SUPPORT SERVICES



19.3 Software Support Services

CSS - Customer Support Service

SE - Systems Engineer

PICS - Phone-in Consulting Service

SSS - Software Subscription Service

SNS - Software Notification Service

MUS - Manual Update Service

The Communicator magazine describes application tips and suggestions for the use of HP software. The software Status Bulletin (SSB) contains information on software bugs and their interim programming solutions.

The Software Update Notice (SUN) includes the following:

* Describes the changes which have occurred in the present update cycle.

* Describes how to incorporate the changes into your system.

Changes include:

- enhancements to software
- bug fixes to software
- software status changes (active - mature - obsolete)
- bug fixes to manuals
- firmware revisions

Current revision code of all software

Consulting services are also available.

SOFTWARE SUPPORT SERVICES

CSS

ACCOUNT ASSIGNED SE
ON-SITE SE ASSISTANCE
PHONE-IN CONSULTING SERVICE

SSS

SOFTWARE/FIRMWARE UPDATES
SOFTWARE PROBLEM REPORTING

SNS

COMMUNICATOR
SOFTWARE STATUS BULLETIN
SOFTWARE UPDATE NOTICE

MUS

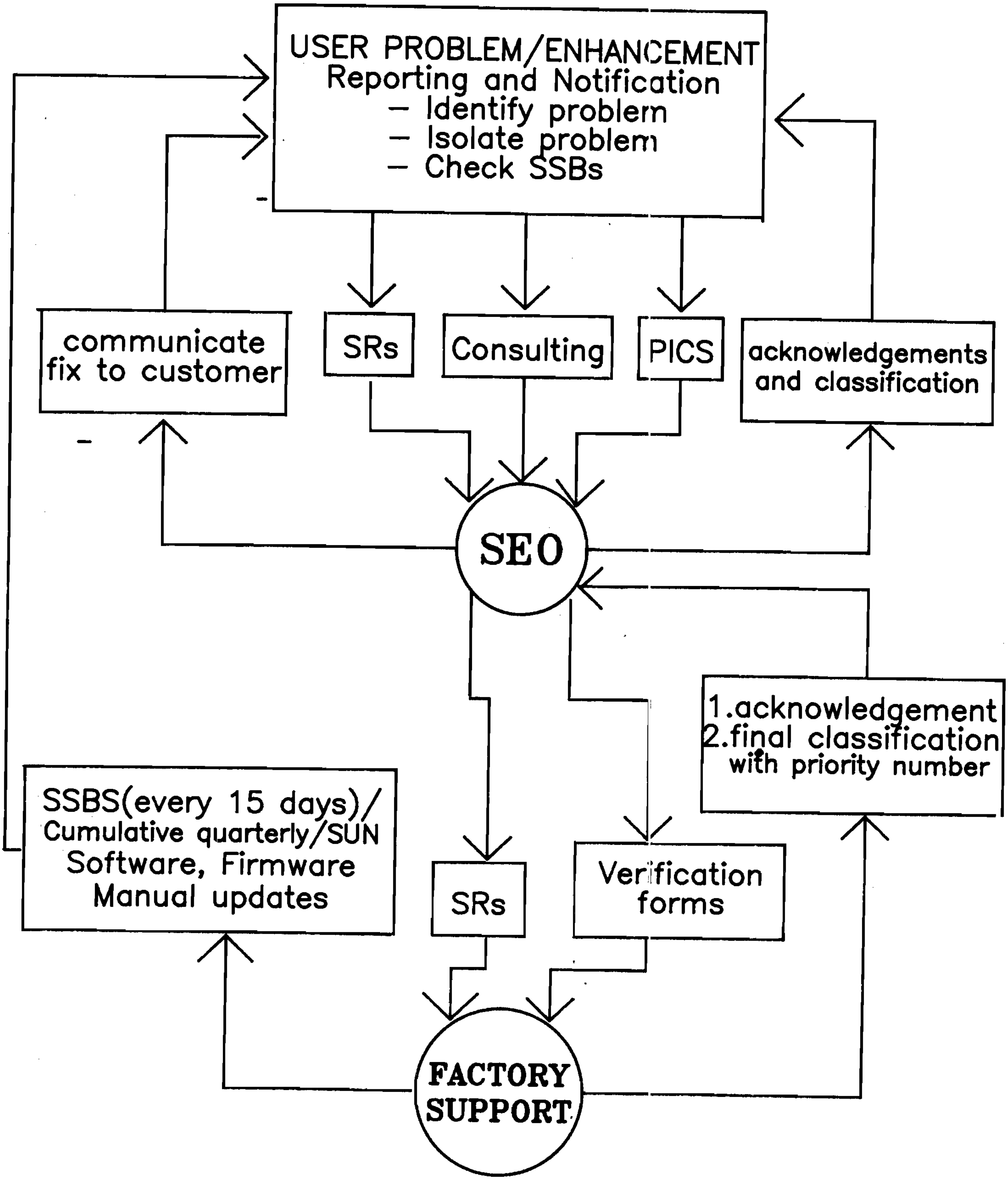
MANUAL UPDATES

19.4 Bug Process

For software not operating according to specifications of the data sheet or manuals:

* Isolate the problem to a repeatable set of circumstances
* Report the bug to your field office:
  - CSS/SSS customers: Fill out a service request form (SR).
  - CSS customers only: call your PICS number.
  - Non-CSS/SSS customers: purchase SE consulting to help resolve discrepancies.

BUG PROCESS


USER PROBLEM/ENHANCEMENT
Reporting and Notification
– Identify problem
– Isolate problem
– Check SSBs
communicate fix to customer
SRs
Consulting
PICS
acknowledgements and classification
SEO
1.acknowledgement
2.final classification with priority number
SSBS(every 15 days)/ Cumulative quarterly/SUN Software, Firmware Manual updates
SRs
Verification forms
FACTORY SUPPORT

Appendix A

Lab Problems

Lab Problems	APPENDIX A

The following pages are copies of the lab problem files exactly as they exist on the lab tape. We have included them here so that you may easily duplicate them and pass them out to the class as you complete each chapter of the course.

REQUIRED
----

1. You will be introduced to the hardware of the A-Series processors including: I/O cards, processor boards, memory cards, power-on and reset switches, and boot-up procedures.

2. Log-on the system using the account specified by the instructor. When you log-on, a session is created for you with a session number equal to your terminal LU number. Type the WH command to determine your terminal LU:

   CI> wh

   The WH command is actually an implied run for the WH program. You could have typed the RU command to run the program as follows. Try it:

   CI> ru wh

   Now type in the RU,WH command using commas as delimiters and adding a parameter to the WH program:

   CI> ru,wh,pa

   The PA parameter tells WH to list the memory partitions in the system, their size, occupant and status. The last partition listed will be at the "top" end of memory. The "Page Range" information for this partition will tell you how much memory is in the system. Add one to the last page number and multiply by 1024 to get the number of words of memory. Multiply this by two to obtain the number of bytes. How much memory is in your system?

3. The CI program contains an on-line help feature which provides a short summary of any CI command and more. Enter the ? command to obtain a list of all the commands for which help is available:

   CI> ?

   Now try the dl command to look at the contents of the directory /help/ :

   CI> dl /help/

   Are the above two commands related? How do you think the ? above command works?

4. Use the LI command to list the help file for the DL command to your terminal:

```
CI> li /help/dl
```

Note that the file is printed to your terminal one page at a time. In order to see one more page of the listing hit any key except the "a" key (which causes the listing to be aborted) or the carriage return key (which causes the remainder of the file to be listed to the terminal).

Now enter the following command:

```
CI> ? dl
```

How does this command relate to the LI command? The System Manager (or any super-user) is allowed to add files to the /help/ directory. Could the System Manager add a help file for local restaurants?

5. Display your command stack using the / command:

```
CI> /
```

Use the terminal cursor keys to edit your command stack at the "li /help/dl" command so that the help file for the LI command is listed to your terminal. Use the "//" command to repeat your last list command. Now look at the command stack again using "/". Note that the repeated command is listed only once. Also note that the "/" commands are not listed at all. What happens when you enter five slashes:

```
CI> /////
```

6. Using your terminal LU determined in exercise 2, use the copy command to copy the LI help file to your terminal:

```
CI> co /help/li <LU>
```

Now try using LU 1 with the copy command:

```
CI> co /help/li 1
```

LU 1 always refers to your terminal when you are using the system interactively or programmatically. This allows you to run the same commands from any terminal without having to determine it's LU.

7. RP the program DL and call it MINE:

```
CI> rp dl mine
```

Now use the WH command to see your RPed program. The WH command shows all of the ID segments attached to your session. Notice that an ID segment for WH is also in the listing. Where did this come from? What happens when you run the program MINE? Every one else is doing this now also. Whose copy are you running? How is it that everyone can have a program RPed by the name MINE?

Use of the OF command to get rid of the ID segment for MINE:

```
CI> of mine id Use WH to verify that it is gone.
```

OPTIONAL
----

8. When you run WH, their will always be an ID segment in the list for the program WH. The ID segment is removed after WH completes because it was RPed implicitely by running the program. How can you verify that there is really no permanent ID segment in your session for WH?

9. Use the OF command to remove the ID segment for your copy of CI:

```
CI> of ci id
```

Bye.

REQUIRED
----

1. Your RTE-A system contains an on-line course called HELLO that will teach you about many of the commands and utilities available to you. The course is self-paced and you may decide to work through the various topics as you learn about them in class.

   Before running the course, you will need to copy some files into your working directory. Use the WD command to make sure you are currently using your own logon working directory, then type in the following command:

   ```
   CI> co /hello/labfiles/ @
   ```

   This need only be done the before the first time you run the HELLO program. The course may now be run by entering:

   ```
   CI> hello
   ```

   HELLO is menu driven and you need only select the appropriate topic to enter the course at any point in it's sequence. You should now run the course as shown above and work through the first topic on the main menu, "Getting Started with HELLO". After completing this topic, go on to problem 2.

2. In order to familiarize yourself with EDIT/1000, run HELLO and select the topic "Developing Programs", and from that menu select "Edit". Work through the first two topics, "Getting Started with EDIT" and "Six Easy Commands to Edit Any File". You will have a chance to work through more topics during lab 4.

3. Using the editor, type in the sample Pascal or FORTRAN program supplied in this chapter. Compile the program, link the relocatable file, and run the program.

4. The RP (Restore program) command creates an ID segment in memory for a program. Use the on-line help feature to learn more about this command:

    CI> ? rp

RP your program. Run WH. What state is the program in? Off the program with the following command:

    CI> of <program>

Run WH. Why is the program still listed? Use the on-line help to learn about the OF command. Now off the program so that it's ID segment goes away.

RP your program twice using two different program names. Use WH to observe the two copies of the program. How would you select one of these copies to run? The two program names you selected are called "clone" names. Can you run the original (program file) with the two clones in the system? Can you off the programs from another terminal?

5. Use help to see what options are available for WH. You should look at the first 2 options. Don't expect to understand all the information here - it will be presented later in the course. Use WH to answer the following questions:

	option:
How many programs are currently active?	AC
How many users are on the system ?	SE
How much memory is in the system ?	PA
What is the value of the time-slice fence?	ST
How many dynamic memory partitions are available?	ST

Run your program. While the program is waiting for input, run WH,AL from another terminal. What programs are in your session? What state are they in? What are their priorities? Are they real-time or background programs?

OPTIONAL
----

6. You can set the priority at which your program runs in a statement in the source program. You may want to consult the language reference manual to learn how to do this. Their are two other ways to change the program's priority. Use the help command for PR (priority) to learn one way. The other way is with link. Type:

```
CI> link
```

to enter LINK interactively. Type "?" to get a list of the link commands. Note that there is no "? <command>" facility as with CI. Now type:

```
link: re area.rel
```

to re-link your relocatable file for program AREA. Change it's priority and then end your LINK session:

```
link: pr 60
link: en
```

Run the program again and verify that you've changed your program's priority.

7. Run HELLO and work through the topics under "Introduction to RTE-A". This will give you a bit of an introduction to the next two chapters in the course.

REQUIRED

1. When you logged on, your working directory was automatically set for you. What is it's name?

2. Use the DL command to list all the files and subdirectories under the directory /LAB/TREE. Make a drawing of it's tree structure. You may find it easier if you change your working directory to /LAB/TREE.

3. Change your working directory back to your logon directory. Create a subdirectory called LAB3. Every one else is creating this subdirectory also -- how are all the LAB3s distinct from each other? Copy the lowest file in the TREE structure from question 2 into your new sub- directory.

4. Create a sub-subdirectory under LAB3 and call it DOWN_UNDER. Change your working directory to LAB3. Move the file in LAB3 to DOWN_UNDER. Verify that the file no longer exists in LAB3. Now change your working directory to DOWN_UNDER. Move the file back to LAB3. Now move the file to directory /OVER_THERE. What problems have you encountered? Can you "move" this file using one command so that it no longer resides in LAB3? Hint: check the CO command options.

5. Who owns your logon working directory? Who owns the two sub-directories you just created? How did this ownership get set? What is the protection on these directories? Who owns directory /LAB/VAULT and what is it's protection? How can this user access the files in this directory? Change the ownership on DOWN_UNDER to your neighbor. What must you do to get that ownership back?

6. Use the DL command to explore the files in your directory. Use the option "!" after the DL command. What additional information does this give you? You may want to read the help file for DL or the User's Manual to find out what all the information refers to.

BREAK

7. On which LU is your working directory? Create a global directory on a different LU. Make it your working directory. Copy all files under directory /LAB/QUALIFIER to your working directory. Purge all files in your working directory whose last update was before October 1, 1982. "Backup" all files whose last update was after March 1, 1983. Simulate a backup by copying to LU 1:

   CI> co <filemask> 1

   You made a mistake by purging one of your FORTRAN source files. I can't remember the exact name but it had an "XX" in it. Restore it using one command (no peeking!) At this point your directory should contain only files with "YES" (or some variation) in their names, and all files that were purged contain "NO" (or some variation) in their names. Verify this. Use the DL command with the option that lists only purged files.

8. Make sure your working directory is still the one created in problem 7. Create a subdirectory. Using a file mask, copy all files in your directory to the subdirectory (Hint: remember the CO command will be recursive unless the appropriate mask qualifier is specified). Did your purged files also get copied? Create a subdirectory another level down. Copy all files again to this subdirectory. Purge your global directory and all of it's files. What is the fewest number of keystrokes with which this can be accomplished?

9. Reset your working directory. Obtain a directory list for /LAB/LOCALUSER from the remote system (use ds transparency). Copy the smallest file to your working directory.

10. Copy all files in directory /LAB/LOCALUSER on your system to your working directory. If you have trouble, list the directory of /LAB/LOCALUSER.

11. Start spooling for the printer:

    CI> sp on 6 *** assuming the printer LU is 6 ***

    Copy /HELP/CL to the printer. Why isn't the file being printed? What must you do to print it? Every on else is doing this now also -- whose copy will get printed first? Does priority affect this order? What if you were a superuser?

12. Start spoling for the printer using a file in your working directory:

```
CI> sp on 6 myfile
```

Copy /HELP/EX to the printer. Stop spooling for the printer. Where is the data now? How can you get it to print?

13. Turn off all LU redirection from previous problems. Redirect printer output to your terminal. Now copy /HELP/CL to the printer. Will your classmate's printer output also show up on your terminal?

OPTIONAL
----

14. Turn off all LU redirection from previous problems. Determine what LU redirection is currently set up on the system. Try to create a redirection loop:

```
6 => 8      8 => 24      24 => 6
```

What happens when you send data to LU 6? Can you redirect data from your terminal to someone else's? Can you redirect data from someone else's terminal to yours?

15. Since class started today, all system errors have been logged to /LOG/ERRORS. Create an error by entering:

```
CI> ci 24
*** wait for the device to time out ***
CM> of ci..a
```

Now list the error log file. Is your error message there? What other types of entries do you find?

REQUIRED
----

1. Run the HELLO program again and this time work through the topic "Line Edits". This can be found by jumping down through the menu topics "Program Development" and then "Edit".

2. Compile your simple program with the compiler option that gives you a mixed listing; create a list file by specifying (or defaulting "-") the second parameter to the compiler. Spool this file to the printer using the spool LI command. Now use the compiler options that give you relocatable addresses, symbol table, and a cross- reference table; once again spooling the file to the printer. Run LINK on your .REL file and create a load map. Compare the relocatable addresses in the load map with those from your previous source listing. The list file is read as follows:

```
(the following are system modules included with your program:)

   PAS.READSEQUENT   PAS.WRITELINE   PAS.PUT   PAS.SETUPFILE
   PAS.CLOSEFILE   PAS.DOUBLE2ASCII  PAS.REAL2ASCII  PAS.RESETFILE
       .
       .
       .
   .DMP    .DDI    .4ZRO   .FLUN

(the load map columns describe...                        )
(  module name  abs address  size  revision number)
   -----------  -----------  ----  ---------------
   AREA                2000  655.  830401.1219
   PAS.READSEQUENT     3217  268.  92833-16005 REV.2326 830404
       .
       .
       .
   .FLUN              32436   14.  24998-1X197 REV.2001 750701

 Main          2000 - 32453   12588. words
              -------------   ----------------
(              abs address range  total size in words)
```

Notice how big your "little" program is. Compare this with the number of words generated by the compiler. The difference is the space that all of the system routines use. You will learn how to save some of this overhead in the next chapter of the course. Save these listings to compare with ones from the next lab.

3. Now compile the program with the Debug compiler option (sorry, not yet available for Pascal). Link the program (don't forget to specify "DE" while running LINK interactively. Run the program from Debug and try out some of the commands you've learned about.

4. Copy your source program to another file and call it CIRCLE.<ext> Change the program into a subroutine that could be called from a main program (that you will write). A Pascal program will look like:

```
$subprogram
program area (input, output);
procedure circle;
    .
    .  {same}
    .
end;
.  {comment brackets must follow the period (bug)}
```

A FORTRAN program will look like:

```
subroutine circle
    .
    .  {same}
    .
end
```

Compile your subroutine using the appropriate compiler. Now rewrite your AREA program so that it calls the circle subroutine after sending a message to the user such as "begin program". Do not attempt to send any parameters to the subroutine. Parameter passing will be addressed at a later time. Compile your program. When you run LINK you will have to relocate your subroutine with the main program using the RE command. When you EN, there should be no unresolved external references. Run the program.

5. Write a subroutine that computes the area of a square similar to the CIRCLE routine above. Also write a subroutine that computes the area of a triangle (1/2 base * height). Have your AREA program prompt the user to select the figure for which to compute the area then jump to the appropriate subroutine. Once again, do not pass any parameters to the subroutine. Make the subroutines prompt for the radius, length, etc. (It will be necessary to have parameter-less subroutines in one of the following labs.) Compile all these. When you link the main program you will have to relocate all of the above subroutine .REL files to resolve the external references. Confirm that it runs.

OPTIONAL

6. Make a library out of the above subroutines as shown in class. Link your program again but this time search the library with the SE command. Confirm that the program runs properly.

7. Make a command file to compile all subroutines and the program, build the library and link everything together. Use the TR command to transfer control to the command file. Now run CI with the command file as a parameter to accomplish the same result.

8. Change the command file to use $ parameters for all the "variable" names so you could use it for some other set of programs. Try it.

REQUIRED
----

1. Rewrite your main program AREA using EXEC calls 1 and 2.

2. Now use REIO to buffer the input.

3. Use the Programmer's Reference Manual to learn how to use LOGLU. In your main program, inquire the LU of the terminal and use XLUEX (or XREIO) to communicate with the actual LU of the terminal (instead of LU 1).

4. Modify your program again to inquire the program's cloned name using PNAME and to get the current time using FTIME. Make a call to LOGIT so that each time the program is run, a message is sent to the spooling log file in the form:

   <prog name> run at <time> from LU <lu number>

   You will also have to use subroutine KCVT which converts the integer LU into ascii representation. You will find a description of KCVT in the Programmer's Reference Manual.

OPTIONAL
----

5. Write a program that prompts for the LU of an output device. Then have the program read input from the terminal and write to the output device using XLUEX. Run the program and specify the printer as the output device. Now redirect the printer LU to your terminal and try the program again. Observe redirection. Now modify your program to set the bit in the CNTWD to override LU redirection and try the above procedure again.

6. Write a program that will continuously display the current time and terminal LU in the upper right corner of the screen display. The program should write the appropriate characters to your terminal, then go into a timing loop (eg. 1000 iterations of a do nothing loop) so it doesn't completely hog the terminal. The time/LU display should not interfere with the normal use of the terminal. In fact, this will not always be the case since cursor positioning/sensing may sometimes interleave with keystrokes. You will need to know the following about programmatic cursor positioning:

* Screen relative addressing - to move the cursor to any position on the visible screen (row 0-23, column 0-79), have your program write the following:

  <esc>&a<column number>c<row number>y

  where <esc> represents the ESC (escape) key. To insert the <esc> into your program, turn on "Display Functions" (one of your soft-keys) and hit the ESC key. Turn off "Display Functions" before hitting any other keys.

You will also want to return the cursor to it's original position after writeing the time and LU number. To do this you will need to sense the cursor's current position before moving it to the upper right corner of the screen.

* Cursor Sensing - to sense the screen relative position of the cursor, have your program write the following:

  <esc>'

  This is the ESC key followed by a single quote. The terminal will send a position back to your program (you will need a read statement) in the following form:

  <esc>&a020c005Y

  This requires an eleven character buffer. Notice that you may rewrite this buffer exactly as received to reposition the cursor.

REQUIRED
----

1. Make a copy of your "AREA" program in your working directory and name it "LINK.RUN". Now try executing LINK. Rename your program to LINK (no type extension). Execute LINK again. Is there any other directory in which you could put your program and have it override LINK in the /PROGRAMS directory?

2. Run a copy of CI by typing:

   CI> ci

   Now execute the WH command and observe the clone name (which CI also uses for your prompt). Run another copy of CI and observe the next clone name. Check with your neighbor to observe the name under which his copy of CI was cloned. Can you explain this? Use the OF command to terminate each cloned copy of CI. Now use the XQ command to run a copy of CI. Hit the return key a few times. Can you explain this? Now use the SS command to suspend CI, then use the EX command to exit the cloned copy. Use WH to see what's happening. You may use the GO command to resume CI. Use the AT command to schedule CI to run in a minute or so. Before it runs, use WH to see it's status, then suspend the original copy of CI and wait for the cloned copy to run. Now schedule WH to run every 15 seconds and observe the behavior.

3. Change your circle, square and triangle subroutines into programs. Now modify your AREA program to schedule (with wait) the circle, square and triangle programs rather than making subroutine calls. The program names you use with the EXEC calls should be upper case. This is where it is important that your subroutines do not expect parameters. You will learn how to pass parameters to your child programs in the next chapter. Remember to RP your child programs before trying to run the main.

4. Use the Programmer's Reference Manual to learn about the system subroutine IFBRK. Use this in your programs to detect a BR command from the terminal and have your program print a message such as "break detected, terminating now", then have your program call EXEC 6.

OPTIONAL
----

5. Use EXEC 11 to record the start time and end time for your program and then print the elapsed time upon termination.

6. Schedule your time/LU display program (from the last lab) using the AT command so that it runs every second. Now modify your program so that it reschedules itself for one second later. Do each of these methods work equally as well?

REQUIRED
----

1. Write two programs which communicate with each other via a word in System Common Area.

   Program 1 - Enter a loop. Request an integer value from the operator and store it in the System Common Area. When 555 is entered, the program should terminate.

   Program 2 - Enter a loop. Print the value it finds in System Common Area on a terminal. On 555 the program terminates.

   - Use two different terminals, be careful of others using System Common Area to do this exercise (i.e., multi-user environment programs may use blank SCA).

2. Write a program that prints a message on a device. The device should be specified in the program runstring. Use the RMPAR method (or language equivalents) to do this lab.

3. Modify (2) above to also accept a message in the runstring, and print it out to the device specified at run-time. Notice how CI inserts commas automatically and packs blanks together.

4. Modify your "area" program to allow parent-child communication between the parent program-area and the three children programs-circle, square, triangle (i.e., schedule with wait). (The parent should prompt which child to schedule and what message the child is to print out.) This prompt can be done with standard language writes. The parent should pick up the parameter string and LU where it will be printed, and pass the the LU and the string to whichever child is scheduled. The child should write out the message to the LU passed from the parent and then send the computed area back to the parent along with a message telling the parent which area has been computed. The parent will then print out the message followed by the area.

   Notice, in Pascal, in this lab, the child cannot have the standard INPUT and OUTPUT in the program heading and must have $run_string 0$ option (i.e., instead of program test (OUTPUT); use rewrite(output, '1');). The schedule request can have "RU,CHILD,1,1" in the passed buffer and the child can then use standard INPUT and OUTPUT in the program header, but this makes it more difficult to also retrieve the message required for this lab. The child program name needs to be in capital letters and any character strings passed via EXEC, GETST etc. needs to be packed array of characters. Remember, Pacsal integers are two-word integers, while FORTRAN integers are one-word integers.

REQUIRED

1. Write two programs (a Parent and a Child). The Parent should:

   - Go into a loop, prompting the user for a string of characters and placing the string in a "mailbox".
   - Terminate the loop when the string "XX" is entered.
   - then, schedule the Child.

   The Child should then retrieve and print the strings, terminating after all the strings have been retrieved and printed. USE CLRQ.

2. Modify the programs you wrote for (1) so that the Child is scheduled after the first string of characters is entered, rather than waiting until an "XX" (the last string) is entered.

3. Modify one of your parent-child programs from "area" program (i.e., Parent = area, Child = circle, triangle, or square). The parent should:

   - Use CLRQ to assign class ownership.
   - Prompt the user to enter a string of characters.
   - Place the string in the "mailbox".
   - Schedule the child to retrieve and print the string to the specified LU.

   Write the parent program to schedule the child "without wait" and assign the child class ownership. Since it is scheduled without wait, the child will have to print out the area itself and will not be able to send back a message to the parent.

OPTIONAL

4. In (2) if the Child "hangs up" for some reason (the printer is down perhaps), the Parent could end up having many buffers in SAM, all waiting to be retrieved by the Child. Arrange your programs so that there will be a maximum of four buffers in SAM at any one time. (Perhaps use a second class number and let the Child tell the Parent when a buffer is consumed. This way the Parent can keep a running count of the number of buffers in SAM.)

5. Write a program which prompts you for a string of characters. Let the program repeatedly print a message on the line printer until you respond with your input. Use Class Get option bits!

REQUIRED
----

1. Write a program that will create a type 2 file with:

   length - 5 blocks
   record length - 1 word.

   The program should store the value 1 into the first record, the value 2 into the second record, and so on. Compile, load and run the program. After the program terminates, use the LI command to verify the operation of your program. Notice, in Pascal, use STRDSC for file descriptor, option string, etc, which are declared as packed array of char.

2. Write a program designed to modify the file created in 9-1. The program will:

   - open the file (update mode)
   - store the value 7777 into the first record
   - terminate

   Use LI to check programs operation.

3. Modify the 'area' program (i.e., question 4 of Lab 7 or question 3 of Lab 8) to RP the child programs. The parent should use FMPRPPROGRAM to programmatically RP the children - square, triangle, and circle. Notice that FMP upshifts characters, and so the children should be RPed in uppercase.

4. Write a program to copy a file to another file using using two different methods (i.e., use FMPCOPY, and sequence of FMP reads and writes).

OPTIONAL
----

5. Write a program to determine the parent directory of your working directory and set your directory to this directory. (Use FMPWORKINGDIR, FMPHIERARCHNAME, and SETWORKINGDIR....).

6. Extend the program from question 5 to accept a parameter which indicates the number of levels up in the tree structure and set that directory to your working directory. A '-1' indicates that your global directory should be set as your working directory.

REQUIRED

1. Write a simple CDS program which pauses. When the program suspends, use WH,PA to see the partition it uses. Identify both the code and data partition.

2. Make the above program shared and invoke the program multiple times. Again, use WH,PA to examine the partitions used. Make sure only one copy of the code partition is present.

OPTIONAL

3. Write a program similar to your 'area' program (Chapter 6 version). Make the three children program segments instead of separate programs. Use NON-CDS segmentation (i.e., SEGLD, SEGRT, or Pas.SegmentLoad). The child should only write out the area, not send any information back to the parent nor receive any from the parent.

4. How would default CDS segmentation create code segments for the following modules?

a.	RE	A.REL	15	b.			15
	RE	B.REL	12				17
	RE	C.REL	5				15
	RE	D.REL	27				7
	RE	E.REL	5				10
					RE	F.REL	12

How could you reorder these modules manually to create fewer, but larger segments?

REQUIRED

Note: All source and relocatable code mentioned in the following labs are contained in /lab/problem/XXXXXX.

1. The file VM11.FTN contains the source code of a program that reads in 16384 real values from a type 1 file called RNDFIL and prints out the average. Make a copy of the program and compile it. Does it compile?

   Correct your copy of the program and run it.

   Modify the program so that the values are stored in VMA.

2. The file VM112.FTN contains the source code of a program which will compute the standard deviation of the EMA array of your program in problem 1.

   Make a copy of the program; compile and load with large enough Shareable EMA partition. Modify your program from problem 1, so that both programs access the data from Shareable EMA.

3. Make a copy, compile, and load the VMA program in VM113.REL, specifying a working set size of 6 pages. Relocate ETIME.REL after VM113.REL. Run the program.

   The program processes a very large array in sequential order and reports system time before and after the processing.

   Increase the size of the Working Set. Run the program and explain why the times are different.

4. Write Parent - Child programs using Shareable EMA. The child should calculate all the prime numbers between 1 and 1000 (use MOD) and store the values in Shareable EMA. The parent should print out the results it retrieves from shareable EMA. The parent should schedule the child WITH wait.

OPTIONAL

5. Write a Pascal program using VMAIO. Notice the variable types needed to make the call.

Chapter 12 - Labs

REQUIRED

1. Write a program which has exclusive access to the line printer by using an LU lock. Have the program pause before unlocking the printer. While the program is suspended, try to list a file to the printer.

2. Write two programs which will compete for use of the line printer.

   - Program 1 should write the message "I'M PROGRAM 1" 25 times on the printer.

   - Program 2 should write the message "I'M PROGRAM 2" 25 times on the printer.

   - Program 1 should schedule program 2 without wait before starting to print its message. This way, the two programs will be competing for the same resource.

   Modify the programs to use a Resource Number so that the output will alternate between 5 lines from program 1 and 5 lines from program 2.

   Some suggestions:

   - Use an unbuffered line printer.

   - After a program unlocks the RN, have the program output a message to the printer. (Perhaps print "PROGRAM x's TURN OVER".)

   - Which program should deallocate the RN?

3. Write 2 programs, a parent and a child. Write the parent so that it schedules the child, passing it an arbitrary parameter. The child should then write a buffer "child active" to LU 1 using EXEC write. Verify that this part is running correctly.

   Now, add the following functions:

   In the parent:
   - Lock LU 1 using LURQ.
   - Pass the KEYNUM parameter to the child.

   In the child:
   - Use the KEYNUM parameter passed by the parent to unlock the LU so it can print it's message.
   - Terminate the child.

   Run the parent and see if indeed the child can write through the LU lock owned by the parent.

4. Modify the Parent - Child program from Chapter 11 (Shareable EMA) to schedule the child WITHOUT wait. The parent should calculate the prime numbers and the child should print them out. Use resource numbers to coordinate access to Shareable EMA partition.

OPTIONAL
----

5. Write a program which prompts the operator for an LU number and then writes whether the LU is up or down. Put the line printer down to verify your program works.

REQUIRED
----

1. Configure a 7912 disc with a CTD for 3 FMGR LUs of about equal size and 3 CI volumes of about equal size. The boot LU should be an FMGR LU. The existing configuration is that of the primary and there is only one disc on the system. What is the disc allocation unit for the CI volumes? The 7912 has 256256 blocks.

2. A 7933 disc should be configured for a FMGR LU to be the boot LU and the rest should be CI volumes. This application will use a large number of very small files, thus the disc allocation unit should be no greater than 2 blocks. (Make as few LUs as possible). The boot LU for the new configuration will be the same as that of the existing configuration (the primary). The 7933 has 1,579,916 blocks.

3. HP has just announced support of a new (mythical) flexible disc, the HP12345, but there is no default file for it yet. Configure it for your system. The specifications are:

   - 120 tracks total, 8 reserved for spares
   - 16 blocks/track

OPTIONAL
----

4. Configure a 7925 disc into 4 CI volumes of about equal size. We are not booting from this disc. Will you use surface or cylinder mode? Why? The 7925 has 9 surfaces, 823 cylinders and 64 blocks/track.

5. Configure a 7920 disc in surface mode for CI volumes. How big is each LU in blocks? Are there any advantages to configuring this disc in surface mode? The 7920 has 5 surfaces, 823 cylinders, and 48 blocks per track.

CS80 DISC WORKSHEET

LU Number

Disc Driver Parameters					
DP1 HPIB addr					
DP2 unit & vol #					
DP3 \ start					
DP4 > block					
DP5 / number					
DP6 tracks					
DP7 sect/track					
DP8 reserved					

LU Number

Disc Driver Parameters					
DP1 HPIB addr					
DP2 unit & vol #					
DP3 \ start					
DP4 > block					
DP5 / number					
DP6 tracks					
DP7 sect/track					
DP8 reserved					

LU Number

CTD Driver Parameters					
DP1 HPIB addr					
DP2 CTD U/V #					
DP3 cache U/V #					
DP4 \ disc cache					
DP5 / block					
DP6 disc cache					
DP7 block					
DP8 reserved					

FLEXIBLE MINI-DISC CONFIGURATION WORKSHEET

Cylinders:

```
       0                                      0
Head 0 +---------------------+ Head 0 +---------------------+
       |                     |        |                     |
       |                     |        |                     |
Head 1 +---------------------+ Head 1 +---------------------+
            unit #                         unit #

                                           Total Tracks:

    disc LU

-----------------+----+----+----+----+----+----+----+----+----+
DP1 HPIB addr    |    |    |    |    |    |    |    |    |    |
DP2 unit number  |    |    |    |    |    |    |    |    |    |
DP3 start head   |    |    |    |    |    |    |    |    |    |
DP4 start cyl    |    |    |    |    |    |    |    |    |    |
DP5 spares       |    |    |    |    |    |    |    |    |    |
DP6 tracks       |    |    |    |    |    |    |    |    |    |
DP7 blks/track   |    |    |    |    |    |    |    |    |    |
DP8 surfaces     |    |    |    |    |    |    |    |    |    |
                 +----+----+----+----+----+----+----+----+----+
```

NON-CS80 DISC WORKSHEET

Driver Parms:

start #					
1 HPIB addr					
2 unit number					
3 start head					
4 start cyl					
5 spares					
start #					
6 tracks					
7 sect/track					
8 surfaces					

Driver Parms:

start #					
1 HPIB addr					
2 unit number					
3 start head					
4 start cyl					
5 spares					
start #					
6 tracks					
7 sect/track					
8 surfaces					

REQUIRED
----

1. In the driver relocatable files, the GEN records start at record 2. List the first few lines of the relocatable file %DDC12. What are the default DVT parameters specified in this file? Do the same for %ID.00. What are the default IFT parameters specified? List the first few lines of %DD.00. The parameters following a specific model number are used only if that model number is specified in the generation answer file. What are the default DVT parameters for a 2621 terminal?

2. Copy the file /LAB/GENLAB/ANS1 to your directory. This is a generation answer file for non-VC+ which is similar to the primary system and to the system you are running on. Make the following changes to the answer file:

   - remove the 2635 console/printer at LU 66 and 67.

   - add a 262x terminal at LU 66 using an async card at select code 33B.

   - add an HPIB card with both a 2608S lineprinter (LU 85, HPIB address 2) and a 2631 printer (LU 6, HPIB address 6), use select code 30B; add any drivers you need for these printers.

   - add the pascal library $PLIB as a default library

   Run the generator until there are no errors. Don't attempt to install your system - this is the next lab. (Hint: make sure the cartridges containing the system relocatables are mounted).

3. Using this answer file, create an answer file for a VC+ system. Run the generator until there are no errors in your answer file. Don't attempt to install your system - this is the next lab.

OPTIONAL
----

4. Copy the file /LAB/GENLAB/ANS2 to your directory. This is a generation answer file which is similar to the primary answer file for a non-VC+ system. Unfortunately, one of your former colleagues (who has since left to join a startup company) has already attempted to modify the answer file and has left it with several errors. Run RTAGN to find the errors and fix them. (Hint: Are the cartridge reference designations correct for your system? You may wish to remove DS from the answer file).

   DO NOT RECONFIGURE THE SYSTEM DISC IN ANY OF YOUR SYSTEMS !

REQUIRED
----

1. Logon as MANAGER and run USERS to make yourself a superuser and give yourself a hello file. Create an additional account for yourself using your last name. Use the same working directory and give this account a password. Logon under these accounts to verify. Remove the account which you just created.

2. Your instructor will give you the boot command string for your system. What are the HPIB address and select code for the system disc? What is the name of the boot command file? The welcome file? The system and snap files? What volumes will be mounted? What startup program will be run? How could you tell this is a VC+ system?

3. Create a boot command file and welcome file for the non-VC+ system which you created in the previous lab. (Your system will have 2 versions of CI available. The VC+ version will be called CI.RUN::PROGRAMS. The non-VC+ version is probably called CINCD.RUN::PROGRAMS - check with your instructor). Put the necessary files on the boot LU. Are the required programs and directories available to the new system? You don't need to run INSTL, since you haven't changed the disc configuration. Bootup your new system and verify operation.

4. Repeat the above procedure for the VC+ system that you created in the previous lab. After booting the system, initialize the spool system. Verify your new system.

REQUIRED
----

1. Use your non-VC+ system to build a 128 K memory based system. Include CI (non-CDS), D.RTR, WH and a small program that you have written (AREA from the first day would be fine). Put your merged system file on LU 16 and boot the system using the existing BOOTEX - don't modify BOOTEX. Verify operation of the system. What commands can be used from CI?

2. Using your VC+ system, build a memory based system. Include the CI (CDS), D.RTR, WH and one of your programs. Make CI a shared program and RP two copies. Boot and verify your system.

OPTIONAL
----

3. Install one of your memory based system on either magnetic tape or CTD. Boot and verify the system.

REQUIRED

1. Create a subdirectory called TFLAB under your global directory. Copy all the files under /LAB/TFLAB to this subdirectory. Use TF to backup all the files in this subdirectory to tape. (Hint: use the K option to keep the tape online throughout this lab). Now restore the files to the same place. Restore the files again into a new subdirectory called MORETF without creating it first, and without creating an additional subdirectory TFLAB.

2. Create a subdirectory under your global directory called DELTA. Copy all the files under /LAB/DELTA to this subdirectory. Are the backup bits set on these files? Do a full backup of your subdirectory DELTA using TF and clearing the backup bit. (Hint: use the K option to keep the tape online throughout this lab).

   Now, do the following:

   Edit FILE1 and do an immediate ER
   List FILE2
   Copy FILE3 into FILE4
   Edit FILE5 and change its contents
   Rename FILE6 to NEWFILE

   Which files have the backup bit set? Do an incremental backup of this subdirectory on the same tape as your full backup.

   Purge your subdirectory DELTA and all the files in it. Restore the files from your backup. Are the backup bits set for these files?

3. Use FC to copy to tape all the files on LU 16 which begin with the character "%".

4. (If LU 23 is empty, do this exercise with LU 22). What is the size of LU 23 in blocks? How much of this is free space? Which CI volume has the largest block of free space? Pack LU 23. (The disc should only be packed by one student at a time). What is the largest free space on LU 23 now? Has the total free space changed? Why? Which files should be moved to increase the largest free space?

5. Verify the LU which contains your gloabl directory. If you do this while others are accessing the disc, you will probably see errors. Did any errors occur? How would you fix them?

6. FREES and FPACK are not used for FMGR LUs. To do these functions on a FMGR LU you will need to use FMGR commands. Read in the Utilities Manual about the FMGR commands: DL, CL and PK. Run FMGR. Do a CL command. What information does it give you? Do a DL command on lu 16. How large is LU 16 in blocks? How much of that is available? To make more space available on the LU, the PK command can be used, however, DO NOT try this command now - it shouldn't be used while others are accessing the disc!

OPTIONAL
----

7. Use ASAVE to backup LU 20. Do not lock the disc (so that others can still access it).

Appendix B

Lab Solutions

Chapter 1 - Lab Solutions

1. Hardware introduction

2. Check with your instructor to verify the amount of memory in your system.

3. The "?" command is identical to typing "dl /help/". When the "?" command is issued, CI performs a sequence of subroutine calls that are similar to that the DL program uses to access the /HELP directory contents.

4. The "? dl" command is identical to typing "li /help/dl". A super-user need only add a file to the /HELP directory to provide on-line help for any subject. To add a help file for restaurants, the system manager would add a file called /help/restaurants. Then any user could type "? restaurants" to get help at lunch time.

5. One slash displays a list of the previous 12 commands. Two slashes display only the last command. Three slashes, the last two commands; four, the last three commands; five, the last four, etc.

6. If the WH command returned a line such as:

   ```
   Session  71 User GEORGE
   ```

   then your terminal LU would be 71. The following two commands would then be identical:

   ```
   CI> co /help/li 71
   CI> co /help/li 1
   ```

7. When you run the WH program, it is implicitely RPed and will show up as having an ID segment attached to your session. The program MINE is attached to your session and can have the same name as a program attached to anyone else's session. CI differentiates them by session number so that you run only your copy of MINE.

Computer Museum

8. You can explicitely RP the program WH under a different name. For example:

   CI> rp wh check

   Now when you run CHECK, you will get a WH listing but WH will not be present. You could also use the command:

   CI> wh al

   from another terminal to verify that no ID segment for WH is attached to your session.

9. When you OF your own copy of CI, and there are no more programs in your session and you will be automatically logged off. Notice that this is different than using the EX command since you are avoiding the termination processing that EX does.

1. Run the HELLO program

2. Run the HELLO program

3. See /LAB/SOLUTION/LAB2/S3.PAS or S3.FTN for the source file. Compile the sourcé file with one of the commands:

```
CI> pascal area.pas 1 -
CI> ftn7x area.ftn 1 -
```

   Link the relocatable file with the command:

```
CI> link area.rel
```

   Run the program with the command:

```
CI> area
```

4. RP your program with the command:

```
CI> rp area
```

   The OF command aborts a program's execution but does not remove an ID segment that has been explicitely RPed. To remove the ID segment, you must use the command:

```
CI> of area id
```

   You can create two separate ID segments for the same program by using two distinct clone names when RPing. For example:

```
CI> rp area name1
CI> rp area name2
```

   Now you can run either copy simply by specifying the name of the program you want to run:

```
CI> name1
CI> name2
CI> area
```

   The last command will run your original copy of the area program; it knows nothing of the cloned copies in the system. Only super-users can off programs in your session. Even if you log on from another terminal using the same account name, you will not be able to off the programs in your first session.

5. CI> wh ac

   Most sessions will have one or two active programs. The System Session will have many active programs, most of which are associated with the DS/1000 network which links your computer with others.

   CI> wh se

   All of the sessions except the System Session will be associated with a user on the system.

   CI> wh pa

   The amount of memory should be the same as in Lab1, question 2.

   CI> wh st

   The value of the time-slice fence and the number of dynamic memory partitions cna be read directly form the output of this command.

   CI> wh al

   There are probably two programs in your session: CI and AREA. AREA will be class suspended (waiting for your input) and CI will be waiting for AREA to complete. CI is usually priority 51 and AREA (unless otherwise specified) will be 99. These are both likely to be background programs as long as the "background fence priority" (use wh,st) is lower than either of the program priorities.

6. Set program priority using LINK.

7. Run the HELLO program.

```
{  /LAB/SOLUTION/LAB2/S3.PAS
}
program area (input, output);

var radius, area : real;

begin
   writeln ('area of circle program');
   repeat
      writeln ('radius: _');
      read (radius);
      area := 3.14159 * radius * radius;
      if radius > 0
         then writeln ('area =', area:4:2)
         else writeln ('finished')
   until radius <= 0
end.
```

```
C  /LAB/SOLUTION/LAB2/S3.FTN
C
      program area

      real radius, area

      write (1,'("area of circle program")')
      radius = 1
      do while (radius .GT. 0)
         write (1,'("radius _")')
         read (1,*) radius
         area = 3.14159 * radius * radius
         if (radius .GT. 0) then
            write (1,'("area = " F4.2)') area
         else
            write (1,'("finished")')
         end if
      end do
      end
```

1. Your working directory name is likely to be the same as your logon name. The name is specified by the system manager when he creates your account.

2.

```
                        TREE
      -------------------------------------
ISN'T                   THIS                  XCITING
 ---                 ----------                 --
LOOKING            AT          MY              TREE
                 -----
              STRUCTURE
             -----------
           EVEN        WAY
                        --
                       DOWN
                        --
                       HERE
```

3. All of the LAB3 directories are distinct because they each have a different parent directory, namely your logon working directory. Use the following copy command:

```
CI> co /lab/tree/this/at/structure/way/down/here lab3/here
```

4. You should have used the following sequence of commands:

```
CI> crdir  lab3/downunder
CI> wd  lab3
CI> mo  here  downunder/here
CI> dl  lab3
CI> wd  downunder
CI> mo  here  /george/lab3/here
CI> mo  /george/lab3/here  overthere/here
CI> co  /george/lab3/here  overthere/here  p
```

You will have trouble with the last command if someone else has already copied the file into directory /OVERTHERE since the copy command will not overwrite an existing file.

5. You can use the OWNER command to determine who owns a directory:

```
CI> owner /george
Owner of /GEORGE is GEORGE
```

You will find that the owner is your logon name. Sub-directories are owned by the creator (note lower-case "c"), so you will find that you are also the owner of the sub-directories you've created. Protection can be found with the PROT command:

```
CI> prot /george
```

and you will find that your logon directory has protection of "rw/r" meaning that you can read or write your own files but other users can only read them. Files and sub-directories take on the protection attributes of their parent directory when they are created.

The owner of /LAB/VAULT is the owner of the directory /LAB which is MANAGER. You will find that the file /LAB/VAULT is read/write protected for both the owner and other users. If the owner is a general user, he must change this protection in order to read or write to the file. If the owner is a super-user, then protection has no consequence.

If you change the ownership of a directory to your neighbor, you will not be able to access the directory or get the ownership back again. Your neighbor must use the OWNER command to give it back to you.

6. The "!" option to the DL command gives you the following information:

```
name ex ba tmp sc prot type msize blks words recs rlen addr
```

The headers indicate:

ex	Extent; * means the file has extents
ba	Backup; * means the file hasn't been backed up since its last change
tmp	Temporary; * means the file is a temporary file
sc	Security code; always 0 except for FMGR files
msize	Size of main file in blocks
blks	Total number of blocks in main and extents
words	Number of words, up to the end-of-file mark
recs	Number of records in the file
addr	Block address of beginning of file

Also listed are the create time, access time and update time.

1. CI> hello

2. Pascal compiler options:

```
$ MIX ON $           {mixed listing}
$ TABLES ON $        {relocatable addresses and symbol table}
$ XREF $             {cross-reference listing}
```

   Then use the runstring:

```
CI> pascal area.pas - -
```

   FORTRAN compiler options:

```
FTN7X,M              !mixed listing
FTN7X,MQTC           !mixed listing, relocatable addresses,
                     !symbol table, and cross-reference listing
```

   Then use the runstring:

```
CI> ftn7x area.ftn - -
```

3. FORTRAN compiler option:

```
FTN7X,S
```

   When running link:

```
link: re,area.rel
link: de
link: en
```

4. See /LAB/SOLUTION/LAB4/S4.PAS or S4.FTN

5. See /LAB/SOLUTION/LAB4/S5.PAS or S5.FTN

6. Make a library in the following manner:

```
CI> merge 1 temp.rel
Enter filename circle.rel
Enter filename square.rel
Enter filename triangle.rel
Enter filename
Merge Stop

CI> lindx temp.rel area.lib
Sorting entries
CIRCLE
SQUARE
TRIANGLE

CI> link

link: re area.rel
link: se area.lib
link: en

CI>
```

7. The merge command will require an input file containing the names of the relocatable subroutine files (the same ones you had to type in manually in the last problem). The file is called AREA.MRG and looks like:

```
circle.rel
square.rel
triangle.rel
```

The command file is located in /LAB/SOLUTION/LAB4/S7.CMD and you can create your .RUN file by executing:

```
CI> ci s7.cmd
```

8. The command file is located in /LAB/SOLUTION/LAB4/S8.CMD and you can create your .RUN file by executing:

```
CI> ci  s8.cmd  area  circle  square  triangle
```

```
{  /LAB/SOLUTION/LAB4/S4.PAS
}
File AREA.PAS:

program area (input, output);

procedure circle;
   external;

begin
   writeln ('area program');
   circle
end.


File CIRCLE.PAS:

$subprogram
program area (input, output);
procedure circle;

var radius, area : real;

begin
   repeat
      writeln ('radius: _');
      read (radius);
      area := 3.14159 * radius * radius;
      if radius > 0
         then writeln ('area =', area:4:2)
         else writeln ('finished')
   until radius <= 0
end;
. {}
```

```
C  /LAB/SOLUTION/LAB4/S4.FTN
C
File AREA.FTN:

      program area

      write (1,'("area program")')
      call circle
      end


File CIRCLE.FTN:

      subroutine circle

      real radius, area

      radius = 1
      do while (radius .GT. 0)
         write (1,'("radius _")')
         read (1,*) radius
         area = 3.14159 * radius * radius
         if (radius .GT. 0) then
            write (1,'("area = " F4.2)') area
         else
            write (1,'("finished")')
         end if
      end do
      end
```

```
{  /LAB/SOLUTION/LAB4/S5.PAS
}
File AREA.PAS:

program area (input, output);

var selection : integer;

procedure circle;
   external;
procedure square;
   external;
procedure triangle;
   external;

begin
   writeln ('area program');
   repeat
      writeln ('select one:');
      writeln ('0 = finished, 1 = circle, 2 = square, 3 = triangle');
      read (selection);
      case selection of
         0 : writeln ('finished');
         1 : circle;
         2 : square;
         3 : triangle
         end;
   until selection = 0
end.


File CIRCLE.PAS:

$subprogram
program area (input, output);
procedure circle;

var radius, area : real;

begin
   writeln ('radius: _');
   read (radius);
   area := 3.14159 * radius * radius;
   if radius > 0
      then writeln ('area =', area:4:2)
      else writeln ('invalid data')
end;
. {}
```

File SQUARE.PAS:

```
$subprogram
program area (input, output);
procedure square;

var side, area : real;

begin
   writeln ('side: _');
   read (side);
   area := side * side;
   if side > 0
      then writeln ('area =', area:4:2)
      else writeln ('invalid data')
end;
. {}
```

File TRIANGLE.PAS:

```
$subprogram
program area (input, output);
procedure triangle;

var base, height, area : real;

begin
   writeln ('base: _');
   read (base);
   writeln ('height: _');
   read (height);
   area := 0.5 * base * height;
   if (base > 0) and (height > 0)
      then writeln ('area =', area:4:2)
      else writeln ('invalid data')
end;
. {}
```

```
C   /LAB/SOLUTION/LAB4/S5.FTN
C
File AREA.FTN:

      program area

      integer selection

      write (1,'("area program")')
      selection = -1
      do while (selection .NE. 0)
         write(1,'("select one:")')
         write(1,'(
     +         "0 = finished, 1 = circle, 2 = square, 3 = triangle")')
         read (1,*) selection
         if (selection .EQ. 0) write (1,'("finished")')
         if (selection .EQ. 1) call circle
         if (selection .EQ. 2) call square
         if (selection .EQ. 3) call triangle
      end do
      end


File CIRCLE.FTN:

      subroutine circle

      real radius, area

      write (1,'("radius: _")')
      read (1,*) radius
      area = 3.14159 * radius * radius
      if (radius .GT. 0) then
         write (1,'("area =", f4.2)') area
      else
         write (1,'("invalid data")')
      end if
      end
```

File SQUARE.FTN:

```
      subroutine square

      real side, area

      write (1,'("side: _")')
      read (1,*) side
      area = side * side
      if (side .GT. 0) then
         write (1,'("area =", f4.2)') area
      else
         write (1,'("invalid data")')
      end if
      end
```

File TRIANGLE.FTN:

```
      subroutine triangle

      real base, height, area

      write (1,'("base: _")')
      read (1,*) base
      write (1,'("height: _")')
      read (1,*) height
      area = 0.5 * base * height
      if ((base .GT. 0) .AND. (height .GT. 0)) then
         write (1,'("area =", f4.2)') area
      else
         write (1,'("invalid data")')
      end if
      end
```

```
*   /LAB/SOLUTION/LAB4/S7.CMD
*
* compile all programs
*       note: these could also be Pascal compiler runstrings
*
ftn7x area.ftn - -
ftn7x circle.ftn - -
ftn7x square.ftn - -
ftn7x triangle.ftn - -
*
* merge the relocatable subroutine files
*       note: this requires file AREA.MRG to contain the names
*             of the relocatable subroutine files
*
merge area.mrg temp.rel
*
* index the references and create a library
*
lindx temp.rel area.lib
pu temp.rel
*
* link the main program with the library files
*
link area.rel area.lib
*  /LAB/SOLUTION/LAB4/S8.CMD
*
* compile all programs
*       note: these could also be Pascal compiler runstrings
*
ftn7x $1.ftn - -
ftn7x $2.ftn - -
ftn7x $3.ftn - -
ftn7x $4.ftn - -
*
* merge the relocatable subroutine files
*       note: this requires file <$1>.MRG to contain the names
*             of the relocatable subroutine files
*
merge $1.mrg temp.rel
*
* index the references and create a library
*
lindx temp.rel $1.lib
pu temp.rel
*
* link the main program with the library files
*
link $1.rel $1.lib
```

1. See /LAB/SOLUTION/LAB5/S1.PAS AND S1.FTN
2. See /LAB/SOLUTION/LAB5/S2.PAS AND S2.FTN
3. See /LAB/SOLUTION/LAB5/S3.PAS AND S3.FTN
4. See /LAB/SOLUTION/LAB5/S4.PAS AND S4.FTN
5. See /LAB/SOLUTION/LAB5/S5.PAS AND S5.FTN

```
c     /LAB/SOLUTION/LAB5/S1.FTN

c     File AREA.FTN:

      program area

      integer cntwd, selection, header(6), prompt(6), list(25),
     +        finished(4)
      data header/'area program'/, prompt/'select one:'/,
     +     list/'0 = finished, 1 = circle, 2 = square, 3 = triangle'/,
     +     finished/'finished'/

      cntwd = 1
      call exec (2, cntwd, header, 6)
      selection = -1
      do while (selection .NE. 2h0 )
         call exec (2, cntwd, prompt, 6)
         call exec (2, cntwd, list, 25)
         call exec (1, cntwd, selection, 1)
         if (selection .EQ. 2h0 ) call exec (2, cntwd, finished, 4)
         if (selection .EQ. 2h1 ) call circle
         if (selection .EQ. 2h2 ) call square
         if (selection .EQ. 2h3 ) call triangle
      end do
      end
```

```
{ /LAB/SOLUTION/LAB5/S1.PAS }

{ File AREA.PAS: }

program area (input, output);

const ec = 256;     {echo bit}

type int = -32768..32767;
     word = packed array [1..2] of char;
     buffer = packed array [1..80] of char;

var cntwd : int;
    selection : word;
    bufr : buffer;

procedure circle;
   external;
procedure square;
   external;
procedure triangle;
   external;
procedure execwrite (ecode, cntwd: int; bufr: buffer; bufln: int);
   $alias 'exec'$ external;
procedure execread (ecode, cntwd: int; selection: word; bufln: int);
   $alias 'exec'$ external;

begin
   cntwd := 1 + ec;                {set to lu of terminal}
   bufr := 'area program';
   execwrite (2, cntwd, bufr, 6);
   repeat
      bufr := 'select one:';
      execwrite (2, cntwd, bufr, 6);
      bufr := '0 = finished, 1 = circle, 2 = square, 3 = triangle';
      execwrite (2, cntwd, bufr, 25);
      execread (1, cntwd, selection, 1);
      bufr := 'finished';
      {note following changes because exec reads char, not integer}
      if selection = '0' then execwrite (2, cntwd, bufr, 4);
      if selection = '1' then circle;
      if selection = '2' then square;
      if selection = '3' then triangle
   until selection = '0'
end.
```

```
c     /LAB/SOLUTION/LAB5/S2.FTN

c     File AREA.FTN:

      program area

      integer cntwd, selection, header(6), prompt(6), list(25),
     +        finished(4)
      data header/'area program'/, prompt/'select one:'/,
     +     list/'0 = finished, 1 = circle, 2 = square, 3 = triangle'/,
     +     finished/'finished'/

      cntwd = 1
      call exec (2, cntwd, header, 6)
      selection = -1
      do while (selection .NE. 2h0 )
         call exec (2, cntwd, prompt, 6)
         call exec (2, cntwd, list, 25)
         call reio (1, cntwd, selection, 1)
         if (selection .EQ. 2h0 ) call exec (2, cntwd, finished, 4)
         if (selection .EQ. 2h1 ) call circle
         if (selection .EQ. 2h2 ) call square
         if (selection .EQ. 2h3 ) call triangle
      end do
      end
```

```
{ /LAB/SOLUTION/LAB5/S2.PAS }

{ File AREA.PAS: }

program area (input, output);

const ec = 256;     {echo bit}

type int = -32768..32767;
     word = packed array [1..2] of char;
     buffer = packed array [1..80] of char;

var cntwd : int;
    selection : word;
    bufr : buffer;

procedure circle;
   external;
procedure square;
   external;
procedure triangle;
   external;
procedure execwrite (ecode, cntwd: int; bufr: buffer; bufln: int);
   $alias 'exec'$ external;
procedure execread (ecode, cntwd: int; selection: word; bufln: int);
   $alias 'reio'$ external;

begin
   cntwd := 1 + ec;                {set to lu of terminal}
   bufr := 'area program';
   execwrite (2, cntwd, bufr, 6);
   repeat
      bufr := 'select one:';
      execwrite (2, cntwd, bufr, 6);
      bufr := '0 = finished, 1 = circle, 2 = square, 3 = triangle';
      execwrite (2, cntwd, bufr, 25);
      execread (1, cntwd, selection, 1);
      bufr := 'finished';
      {note following changes because exec reads char, not integer}
      if selection = '0' then execwrite (2, cntwd, bufr, 4);
      if selection = '1' then circle;
      if selection = '2' then square;
      if selection = '3' then triangle
   until selection = '0'
end.
```

```
c     /LAB/SOLUTION/LAB5/S3.FTN

c     File AREA.FTN:

      program area

      integer cntwd(2), selection, header(6), prompt(6), list(25),
     +        finished(4), dummy
      data header/'area program'/, prompt/'select one:'/,
     +     list/'0 = finished, 1 = circle, 2 = square, 3 = triangle'/,
     +     finished/'finished'/

      dummy = loglu (cntwd(1))
      cntwd(2) = 0
      call xluex (2, cntwd, header, 6)
      selection = -1
      do while (selection .NE. 2h0 )
         call xluex (2, cntwd, prompt, 6)
         call xluex (2, cntwd, list, 25)
         call xreio (1, cntwd, selection, 1)
         if (selection .EQ. 2h0 ) call xluex (2, cntwd, finished, 4)
         if (selection .EQ. 2h1 ) call circle
         if (selection .EQ. 2h2 ) call square
         if (selection .EQ. 2h3 ) call triangle
      end do
      end
```

```
{ /LAB/SOLUTION/LAB5/S3.PAS }

{ File AREA.PAS: }

program area (input, output);

const ec = 256;      {echo bit}

type int = -32768..32767;
     int2 = array [1..2] of int;
     word = packed array [1..2] of char;
     buffer = packed array [1..80] of char;

var lu, dummy : int;
    cntwd : int2;
    selection : word;
    bufr : buffer;

procedure circle;
   external;
procedure square;
   external;
procedure triangle;
   external;
procedure execwrite (ecode: int; cntwd: int2; bufr: buffer; bufln: int);
   $alias 'xluex'$ external;
procedure execread (ecode: int; cntwd: int2; selection: word; bufln: int);
   $alias 'xreio'$ external;
function loglu (lu: int) : int;
   external;
```

```
begin
   dummy := loglu (lu);          {get real lu of terminal}
   cntwd[1] := lu;               {set to lu of terminal}
   cntwd[2] := ec;               {set echo bit}
   bufr := 'area program';
   execwrite (2, cntwd, bufr, 6);
   repeat
      bufr := 'select one:';
      execwrite (2, cntwd, bufr, 6);
      bufr := '0 = finished, 1 = circle, 2 = square, 3 = triangle';
      execwrite (2, cntwd, bufr, 25);
      execread (1, cntwd, selection, 1);
      bufr := 'finished';
      {note following changes because exec reads char, not integer}
      if selection = '0' then execwrite (2, cntwd, bufr, 4);
      if selection = '1' then circle;
      if selection = '2' then square;
      if selection = '3' then triangle
   until selection = '0'
end.
```

```
c     /LAB/SOLUTION/LAB5/S4.FTN

c     File AREA.FTN:

      program area

      integer cntwd(2), selection, header(6), prompt(6), list(25),
     +        finished(4), dummy, prog(3), time(15), lu, buffer(28)
      data header/'area program'/, prompt/'select one:'/,
     +     list/'0 = finished, 1 = circle, 2 = square, 3 = triangle'/,
     +     finished/'finished'/

      call pname (prog)   !get program name
      call ftime (time)   !get current time
      buffer(1) = prog(1) !put program name in buffer
      buffer(2) = prog(2)
      buffer(3) = prog(3)
      buffer(4) = 2hru    !hollerith notation must be used here
      buffer(5) = 2hn     !assign one word at a time...
      buffer(6) = 2hat    !ho, hum
      buffer(7) = 2h
      do i = 1, 15        !put time in buffer
         buffer(i+7) = time(i)
      end do
      buffer(23) = 2h f
      buffer(24) = 2hro
      buffer(25) = 2hm
      buffer(26) = 2hLU
      buffer(27) = 2h
      dummy = loglu (lu)      !get lu number
      buffer(28) = kcvt (lu)  !convert number to ascii
      call logit (buffer, 28)

      dummy = loglu (cntwd(1))
      cntwd(2) = 0
      call xluex (2, cntwd, header, 6)
      selection = -1
      do while (selection .NE. 2h0 )
         call xluex (2, cntwd, prompt, 6)
         call xluex (2, cntwd, list, 25)
         call xreio (1, cntwd, selection, 1)
         if (selection .EQ. 2h0 ) call xluex (2, cntwd, finished, 4)
         if (selection .EQ. 2h1 ) call circle
         if (selection .EQ. 2h2 ) call square
         if (selection .EQ. 2h3 ) call triangle
      end do
      end
```

```
{ /LAB/SOLUTION/LAB5/S4.PAS }

{ File AREA.PAS: }

program area (input, output);

const ec = 256;     {echo bit}

type int = -32768..32767;
     int2 = array [1..2] of int;
     word = packed array [1..2] of char;
     name = packed array [1..6] of char;
     tbuff = packed array [1..30] of char;
     buffer = packed array [1..80] of char;

var i, lu, dummy : int;
    cntwd : int2;
    asciilu, selection : word;
    prog : name;
    time : tbuff;
    bufr : buffer;

procedure circle;
   external;
procedure square;
   external;
procedure triangle;
   external;
procedure execwrite (ecode: int; cntwd: int2; bufr: buffer; bufln: int);
   $alias 'xluex'$ external;
procedure execread (ecode: int; cntwd: int2; selection: word; bufln: int);
   $alias 'xreio'$ external;
function loglu (lu: int) : int;
   external;
procedure pname (prog: name);
   external;
procedure ftime (time: tbuff);
   external;
function kcvt (lu: int) : word;
   external;
procedure logit (bufr: buffer; len: int);
   external;
```

```
begin
   pname (prog);            {get program name}
   ftime (time);            {get current time}
   bufr := prog;            {put program name in buffer, with blank fill}
   bufr[7] := 'r';          {enter the rest of the characters}
   bufr[8] := 'u';          {... one character at a time!!!}
   bufr[9] := 'n';          { ugh }
   bufr[11] := 'a';
   bufr[12] := 't';
   for i := 1 to 30 do          {put time in buffer}
      bufr[i+13] := time[i];
   bufr[47] := 'f';
   bufr[48] := 'r';
   bufr[49] := 'o';
   bufr[50] := 'm';
   bufr[52] := 'L';
   bufr[53] := 'U';
   dummy := loglu (lu);         {get lu of terminal}
   asciilu := kcvt (lu);        {convert it to ascii}
   bufr[55] := asciilu[1];
   bufr[56] := asciilu[2];
   logit (bufr, 40);

   dummy := loglu (lu);         {get real lu of terminal}
   cntwd[1] := lu;              {set to lu of terminal}
   cntwd[2] := ec;              {set echo bit}
   bufr := 'area program';
   execwrite (2, cntwd, bufr, 6);
   repeat
      bufr := 'select one:';
      execwrite (2, cntwd, bufr, 6);
      bufr := '0 = finished, 1 = circle, 2 = square, 3 = triangle';
      execwrite (2, cntwd, bufr, 25);
      execread (1, cntwd, selection, 1);
      bufr := 'finished';
      {note following changes because exec reads char, not integer}
      if selection = '0' then execwrite (2, cntwd, bufr, 4);
      if selection = '1' then circle;
      if selection = '2' then square;
      if selection = '3' then triangle
   until selection = '0'
end.
```

```
c       /LAB/SOLUTION/LAB5/S5.FTN

        program out

        integer  cntwd(2), buffer(40), redirection
        data redirection /0/  !set to 100000b to override redirection

        write (1, '("output LU = _")')
        read (1,*) cntwd(1)
        cntwd(1) = cntwd(1) + redirection
        write (1, '("write to LU " i2 " ...")') cntwd(1)
        cntwd(2) = 0
        read (1, '(80a)') buffer
        call xluex (2, cntwd, buffer, 40)
        end
```

```
{ /LAB/SOLUTION/LAB5/S5.PAS }

program out (input, output);

const redirection = 0;   {set to -32768 to override redirection}

type int = -32768..32767;
     int2 = array [1..2] of int;
     buffer = packed array [1..80] of char;

var cntwd : int2;
    bufr : buffer;

procedure xluex (ecode: int; cntwd: int2; bufr: buffer; bufln: int);
   external;

begin
   writeln ('output LU = _');
   readln (cntwd[1]);
   cntwd[1] := cntwd[1] + redirection;
   cntwd[2] := 0;
   writeln ('write to LU ', cntwd[1]:2, ' ...');
   cntwd[2] := 0;
   readln (bufr);
   xluex (2, cntwd, bufr, 40)
end.
```

1. When the command to execute LINK is given, the working directory is searched first. Since you now have a program named LINK.RUN in your working directory, this is found first and executed. You could also put the file in a FMGR directory and set your working directory to 0. Now when you issue the RU command, the FMGR cartridges will be searched first and your surrogate LINK will be found.

2. Each program name is associated with it's respective session, transparently to the user. When you and your neighbor both run the same named program, each of you create an ID segment by the same name but attached to different sessions. You can see this by running WH,AL to observe everyone's session.

   When you XQ CI, both CI and CI..A are running at the same time and contend for your terminal. Each time you hit the return key, the other copy of CI takes over and issues it's prompt. This way the two copies share your terminal. This, of course, is not very practical and you will generally want to XQ programs that do not interact with your terminal.

3. See /LAB/SOLUTION/LAB6/S3.PAS and S3.FTN

4. See /LAB/SOLUTION/LAB6/S4.PAS and S4.FTN

```
C     /LAB/SOLUTION/LAB6/S3.FTN
C
C     File AREA.FTN:

      program area

      integer selection, circle(3), square(3), triangle(3)
      data circle/'CIRCL'/, square/'SQUAR'/, triangle/'TRIAN'/

      write (1,'("area program")')
      selection = -1
      do while (selection .NE. 0)
         write(1,'("select one:")')
         write(1,'(
     +        "0 = finished, 1 = circle, 2 = square, 3 = triangle")')
         read (1,*) selection
         if (selection .EQ. 0) write (1,'("finished")')
         if (selection .EQ. 1)
     +      call exec (9, circle)
         if (selection .EQ. 2)
     +      call exec (9, square)
         if (selection .EQ. 3)
     +      call exec (9, triangle)
      end do
      end

C     File CIRCLE.FTN:

      program circle

      real radius, area

      write (1,'("radius: _")')
      read (1,*) radius
      area = 3.14159 * radius * radius
      if (radius .GT. 0) then
         write (1,'("area =", f4.2)') area
      else
         write (1,'("invalid data")')
      end if
      end
```

```
C     File SQUARE.FTN:

      program square

      real side, area

      write (1,'("side: _")')
      read (1,*) side
      area = side * side
      if (side .GT. 0) then
         write (1,'("area =", f4.2)') area
      else
         write (1,'("invalid data")')
      end if
      end

C     File TRIANGLE.FTN:

      program triangle

      real base, height, area

      write (1,'("base: _")')
      read (1,*) base
      write (1,'("height: _")')
      read (1,*) height
      area = 0.5 * base * height
      if ((base .GT. 0) .AND. (height .GT. 0)) then
         write (1,'("area =", f4.2)') area
      else
         write (1,'("invalid data")')
      end if
      end
```

```
{  /LAB/SOLUTION/LAB6/S3.PAS  }
{  File AREA.PAS:  }

program area (input, output);

type int = -32768..32767;
     progname = packed array [1..6] of char;

var selection : integer;
    circle,
    square,
    triangle : progname;

procedure exec (ecode: int; prog: progname);
   external;

begin
   circle := 'CIRCL';
   square := 'SQUAR';
   triangle := 'TRIAN';

   writeln ('area program');
   repeat
      writeln ('select one:');
      writeln ('0 = finished, 1 = circle, 2 = square, 3 = triangle');
      read (selection);
      case selection of
         0 : writeln ('finished');
         1 : exec (9, circle);
         2 : exec (9, square);
         3 : exec (9, triangle);
         end;
   until selection = 0
end.

{  File CIRCLE.PAS:  }

program circle (input, output);

var radius, area : real;

begin
   writeln ('radius: _');
   read (radius);
   area := 3.14159 * radius * radius;
   if radius > 0
      then writeln ('area =', area:4:2)
      else writeln ('invalid data')
end.
```

```
{  File SQUARE.PAS:  }

program square (input, output);

var side, area : real;

begin
   writeln ('side: _');
   read (side);
   area := side * side;
   if side > 0
      then writeln ('area =', area:4:2)
      else writeln ('invalid data')
end.


{  File TRIANGLE.PAS:  }

program traingle (input, output);

var base, height, area : real;

begin
   writeln ('base: _');
   read (base);
   writeln ('height: _');
   read (height);
   area := 0.5 * base * height;
   if (base > 0) and (height > 0)
      then writeln ('area =', area:4:2)
      else writeln ('invalid data')
end.
```

```
C     /LAB/SOLUTION/LAB6/S4.FTN
C
C     File AREA.FTN:

      program area

      integer selection, circle(3), square(3), triangle(3)
      data circle/'CIRCL'/, square/'SQUAR'/, triangle/'TRIAN'/

      write (1,'("area program")')
      selection = -1
      do while (selection .NE. 0)
         if (ifbrk) call exec (6)
         write(1,'("select one:")')
         write(1,'(
     +        "0 = finished, 1 = circle, 2 = square, 3 = triangle")')
         read (1,*) selection
         if (selection .EQ. 0) write (1,'("finished")')
         if (selection .EQ. 1)
     +      call exec (9, circle)
         if (selection .EQ. 2)
     +      call exec (9, square)
         if (selection .EQ. 3)
     +      call exec (9, triangle)
      end do
      end

C     File CIRCLE.FTN:

      program circle

      real radius, area

      if (ifbrk) call exec (6)
      write (1,'("radius: _")')
      read (1,*) radius
      area = 3.14159 * radius * radius
      if (radius .GT. 0) then
         write (1,'("area =", f4.2)') area
      else
         write (1,'("invalid data")')
      end if
      end
```

```
C       File SQUARE.FTN:

        program square

        real side, area

        if (ifbrk) call exec (6)
        write (1,'("side: _")')
        read (1,*) side
        area = side * side
        if (side .GT. 0) then
           write (1,'("area =", f4.2)') area
        else
           write (1,'("invalid data")')
        end if
        end


C       File TRIANGLE.FTN:

        program triangle

        real base, height, area

        if (ifbrk) call exec (6)
        write (1,'("base: _")')
        read (1,*) base
        write (1,'("height: _")')
        read (1,*) height
        area = 0.5 * base * height
        if ((base .GT. 0) .AND. (height .GT. 0)) then
           write (1,'("area =", f4.2)') area
        else
           write (1,'("invalid data")')
        end if
        end
```

```
{  /LAB/SOLUTION/LAB6/S4.PAS  }

{  File AREA.PAS:  }

program area (input, output);

type int = -32768..32767;
     progname = packed array [1..6] of char;

var selection : integer;
    circle,
    square,
    triangle : progname;

procedure exec (ecode: int; prog: progname);
   external;
procedure halt (ecode: int);
   $alias 'exec'$ external;
function ifbrk : boolean;
   external;

begin
   circle := 'CIRCL';
   square := 'SQUAR';
   triangle := 'TRIAN';

   writeln ('area program');
   repeat
      if ifbrk then halt (6);
      writeln ('select one:');
      writeln ('0 = finished, 1 = circle, 2 = square, 3 = triangle');
      read (selection);
      case selection of
         0 : writeln ('finished');
         1 : exec (9, circle);
         2 : exec (9, square);
         3 : exec (9, triangle);
         end;
   until selection = 0
end.
```

```
{  File CIRCLE.PAS:  }

program circle;

var radius, area : real;
    inp, out : text;

begin
   reset (inp, '1');
   rewrite (out, '1');
   if ifbrk then halt (6);
   writeln (out, 'radius: _');
   read (inp, radius);
   area := 3.14159 * radius * radius;
   if radius > 0
      then writeln (out, 'area =', area:4:2)
      else writeln (out, 'invalid data')
end.


{  File SQUARE.PAS:  }

program square;

var side, area : real;
    inp, out : text;

begin
   reset (inp, '1');
   rewrite (out, '1');
   if ifbrk then halt (6);
   writeln (out, 'side: _');
   read (inp, side);
   area := side * side;
   if side > 0
      then writeln (out, 'area =', area:4:2)
      else writeln (out, 'invalid data')
end.


{  File TRIANGLE.PAS:  }

program traingle;

var base, height, area : real;
    inp, out : text;

begin
   reset (inp, '1');
   rewrite (out, '1');
   if ifbrk then halt (6);
```

```
   writeln (out, 'base: _');
   read (inp, base);
   writeln (out, 'height: _');
   read (inp, height);
   area := 0.5 * base * height;
   if (base > 0) and (height > 0)
      then writeln (out, 'area =', area:4:2)
      else writeln (out, 'invalid data')
end.
```

```
FTN7X,L
C
C    /lab/solution/lab7/sla.ftn
C
C    This is a program which stores values in system
C    Common area. Link program with SC command.
C
C

      PROGRAM COMON1
C
      COMMON //NUMBER
C
C  INITIALIZE THE ARRAY
C
      NUMBER = 0
C
      DO WHILE (NUMBER .NE. 555)
         WRITE(1,*)'Please enter a number ?'
         READ(1,*)NUMBER
      END DO
C
C
50    END
```

```
FTN7X,L
C
C     /LAB/SOLUTION/LAB7/S1B.FTN
C
C     This is a program which receives the values
C     from the System Common area.  Link program with SC command.
C
      PROGRAM COMON2
      COMMON //NUMBER
C
C  PRINT OUT THE ARRAY
C
      DO WHILE (NUMBER .NE. 555)
          WRITE(1,*)'THE NUMBER IS : ', (NUMBER)
      END DO
C
C
50    END
```

```
  {
     /lab/solution/lab7/sla.pas
    This program stores values in System Common Area.
    Link program with SC command.
  }

PROGRAM coml (input,output);
TYPE
    int = -32768..32767;
    comptr = ^int;

VAR
    system_ptr : comptr;
    sizeblank : int;

FUNCTION common_blank $ ALIAS 'Pas.BlankComl' $  : comptr;  EXTERNAL;

FUNCTION blank_size $ ALIAS 'Pas.BlankSize' $  : int;  EXTERNAL;

BEGIN
     sizeblank := blank_size;
     IF sizeblank <> 0 THEN
        BEGIN
            system_ptr := common_blank;
            WHILE (system_ptr^  <> 555) DO
                BEGIN
                    writeln('Please enter a number : _');
                    read(system_ptr^);
                END;
        END ELSE writeln('This program has no access to System Common!!');
END.
```

```
{    /lab/solution/lab7/slb.pas

     This program receives values from  the  System Common area.
     Link with the SC command.
                                                                   }
PROGRAM COM2;
TYPE
    int = -32768..32767;
    comptr = ^int;

VAR
    system_ptr : comptr;
    sizeblank : int;
    out : text;

FUNCTION common_blank $ ALIAS 'Pas.BlankComl' $ : comptr; EXTERNAL;

FUNCTION blank_size $ ALIAS 'Pas.BlankSize' $ : int; EXTERNAL;

BEGIN
     rewrite(out,'l');
     sizeblank := blank_size;
     IF sizeblank <> 0 THEN
        BEGIN
             system_ptr := common_blank;
             WHILE (system_ptr^  <> 555) DO
                  BEGIN
                       writeln(out,'The number is : ',system_ptr^);
                  END;
        END ELSE writeln(out,'This program has no access to System Common!!');
END.
```

```
FTN7X,L
C     /lab/solution/lab7/s2.ftn
C
C     This program prints a message to a device which is
C     specified at run-time.
C
C

C
      PROGRAM msgs
C
      IMPLICIT INTEGER(A-Z)
      DIMENSION PARM(5)
C
C
      CALL RMPAR(PARM)
      ILU = PARM(1)
C
      WRITE(ILU,*)'THIS IS THE MESSAGE TO BE PRINTED OUT'
C
      END
```

```
{       /lab/solution/lab7/s2.pas

        This program prints a message to a device which is specified
        at run-time in the program runstring.                          }

PROGRAM MSGS;
TYPE
        int = -32768..32767;
        ptype = packed array [1..5] of int;
        conv_type = packed array [1..6] of char;

VAR
        out : conv_type;
        parm : ptype;
        lu : text;

PROCEDURE PARAMS $ ALIAS 'PAS.NUMERICPARMS' $
        (VAR parm : ptype);  EXTERNAL;

PROCEDURE CNUMD (num : int; buffer : conv_type);  EXTERNAL;


BEGIN
      params(parm);
      cnumd(parm[1],out);
      rewrite(lu,out);

      write(lu,'The message is: YOU''VE FINALLY GOT THE PROGRAM TO WORK!!')

END.
```

```
FTN7X,L
C        /lab/solution/lab7/s3.ftn
C
C       This program accepts a message and also the print LU
C       from the run-string.
C
C
        PROGRAM meslu
C
        IMPLICIT INTEGER(A-Z)
        DIMENSION PARM(5), BUFR(40)
C
C
        CALL RMPAR(PARM)
        ILU = PARM(1)
        CALL GETST(BUFR,40,TLOG)
C
        WRITE(ILU,'(40A2)')(BUFR(I),I=2,TLOG)
C
        END
```

```
ftn7x,l
C
C        /lab/solution/lab8/s3s.ftn
C
C     This child program recieves a message from the parent via
C     class I/O.
C
      program SQUARE

      integer parm(5), message(25)
      real side, area

      call rmpar(parm)
      lu = parm(1)
      iclas = parm(2)

      call exec(21,iclas,message,-50)

      write (1,'("side: _")')
      read (1,*) side
      area = side * side
      if (side .GT. 0) then
         write (lu,'("The message sent from above is : ",50a)') message
         write (lu,'("area = ",f4.2)') area
      else
         write (lu,'("invalid data")')
      end if

      end
```

```
ftn7x,l
C
C     /lab/solution/lab7/s4c.ftn
C
C     This is the circle program.  It calculates the area and passes
C     it back to the parent program.
C
      program S4C

      integer parm(5), buffer(25), message(15)
      real radius, area(3)
      data message/'The area of the circle is :   '/

      call rmpar(parm)
      lu = parm(1)
      call exec(14,1,buffer,-50)
      write(lu,5)buffer
5     format('The message sent from above is : ', 50a)

      write (1,'("radius: _")')
      read (1,*) radius
      area = 3.14159 * radius * radius
      if (radius .LT. 0) then
         write (1,'("invalid data")')
      end if

      call exec(14,2,message,-30)
      call prtn(area)

      end
```

```
ftn7x,l
C
C     /lab/solution/lab7/s4m.ftn
C
C     This is the parent program which schedules the programs
C     circle, triangle, and square. This parent writes out the area.
C
      program parent

      implicit integer(a-z)
      real area
      integer message(25),buffer(15),circle(3),square(3),triangle(3),parm(5)
      data circle/'S4C'/, square/'S4S'/, triangle/'S4T'/

      write (1,'("area program")')
      selection = -1
      do while (selection .NE. 0)
         write(1,'("select one:")')
         write(1,'(
     +        "0 = finished, 1 = circle, 2 = square, 3 = triangle")')
         read (1,*) selection

         if (selection .EQ. 0) then
            write(1,'("finished")')
         else
            write(1,'("Enter a message : ")')
            read (1,5) message
5           format(50a)
            write(1,'("Enter LU to be printed at : ")')
            read(1,*) lu
         endif

         if (selection .EQ. 1)
     +      call exec (9, circle, lu, 0, 0, 0, 0, message, -50)
         if (selection .EQ. 2)
     +      call exec (9, square, lu, 0, 0, 0, 0, message, -50)
         if (selection .EQ. 3)
     +      call exec (9, triangle, lu, 0, 0, 0, 0, message, -50)

         if (selection .NE. 0) then
            call rmpar(area)
            call exec(14,1,buffer,-30)
            write(1,10) buffer
10          format(30a)
            write(1,20) area
20          format(f7.4)
         endif

      end do
      end
```

```
ftn7x,l
C
C      /lab/solution/lab7/s4s.ftn
C
C     This is the square program which calculates the area of
C     a square and passes it back to the parent.
C
      program s4s

      integer parm(5), buffer(25), message(15)
      real side, area
      data message/'The area of the square is :   '/

      call rmpar(parm)
      lu = parm(1)
      call exec(14,1,buffer,-50)
      write(lu,5) buffer
5     format('The message sent from above is : ',50a)

      write (1,'("side: _")')
      read (1,*) side
      area = side * side
      if (side .LT. 0) then
         write (1,'("invalid data")')
      end if

      call exec(14,2,message,-30)
      call prtn(area)

      end
```

```
ftn7x,l
C
C     /lab/solution/lab7/s4t.ftn
C
C     This is the triangle program.  It calculates the area of a
C     triangle and passes it back to the parent.
C
      program S4T

      integer parm(5), buffer(25), message(15)
      real base, height, area
      data message/'The area of the triangle is : '/

      call rmpar(parm)
      lu = parm(1)
      call exec(14,1,buffer,-50)
      write(lu,5)buffer
5     format('The message sent from above is : ',50a)

      write (1,'("base: _")')
      read (1,*) base
      write (1,'("height: _")')
      read (1,*) height
      area = 0.5 * base * height
      if ((base .LT. 0) .AND. (height .LT. 0)) then
         write (1,'("invalid data")')
      end if

      call exec(14,2,message,-30)
      call prtn(area)

      end
```

```
{                    /lab/solution/lab7/s4c.pas

  This child program first prints out a message sent from the parent
  and then calculates the area of a circle.  The area is passed back
  to the parent, so that the parent program can print it out.       }

$run_string 0$
program S4C ;

type
     int    = -32768..32767;
     ptype = packed array [1..5] of int;
     buftype = packed array [1..50] of char;
     outtype = packed array [1..6] of char;

var
     message,
     buffer     : buftype;
     where      : outtype;
     pram       : ptype;
     outer,
     inp, out  : text;
     radius, area : real;
     lu, bufln : int;

procedure rmpar ( VAR pram : ptype);  external;

procedure prtn ( VAR area : real);  external;

procedure cnumd (num : int; buffer : outtype);  external;

procedure execl4 $alias 'exec'$ (ecode,rcode : int; var bufr : buftype;
          var len : int);  external;

begin
     rmpar(pram);
     lu := pram[1];

     bufln := -50;
     execl4(14,1,buffer,bufln);

     cnumd(lu,where);
     rewrite(out,where);
     writeln(out,'The buffer passed from the parent is : ',buffer);

     rewrite(outer,'1');
     reset(inp,'1');

     writeln (outer,'radius: _');
     read (inp,radius);
     area := 3.14159 * radius * radius;
```

```
    if radius > 0
       then writeln (outer,'area =', area:4:2)
       else writeln (outer,'invalid data');

       message := 'The area of the circle is : ';
       bufln := -30;
       execl4(14,2,message,bufln);

       prtn(area);
end.
```

```
{                /lab/solution/lab7/s4m.pas

  This parent program schedules the child programs triangle, square,
  and circle. This program sends a message and a specified LU to the
  child. The child sends back the area and the parent prints it out. }

program s4m (input, output);

type
     int = -32768..32767;
     progname = packed array [1..6] of char;
     buftype = packed array [1..30] of char;
     msgtype = packed array [1..50] of char;

var
    square,
    circle,
    triangle  : progname;
    area      : real;
    selection : integer;
    bufr      : buftype;
    message   : msgtype;
    lu, bufln : int;

procedure exec9 $ alias 'exec' $ (icode : int; prog : progname; lu,dum2,
          dum3,dum4,dum5 : int; var message : msgtype; len : int); external;

procedure rmpar (area : real);  external;

procedure exec14 $ alias 'exec' $ (icode, rcode : int; var bufr : buftype;
          var len : int);  external;

begin
    circle := 'S4C';
    square := 'S4S';
    triangle := 'S4T';

    writeln ('area program');
    repeat
         writeln ('select one:');
         writeln ('0 = finished, 1 = circle, 2 = square, 3 = triangle');
         read (selection);

         if (selection <> 0) then
             begin
                 writeln('Enter a message to be printed by the child : ');
                 read(message);
                 writeln('Enter lu number to print the message at : ');
                 read(lu);
             end;
```

```
        case selection of
           0 : writeln ('finished');
           1 : exec9 (9, circle, lu, 0, 0, 0, 0, message, -50);
           2 : exec9 (9, square, lu, 0, 0, 0, 0, message, -50);
           3 : exec9 (9, triangle, lu, 0, 0, 0, 0, message, -50);
        end;

        if (selection <> 0) then
            begin
                rmpar(area);
                bufln := -30;
                exec14(14,1,bufr,bufln);
                writeln(bufr,area);
            end;

     until selection = 0;

end.
```

```
{                    /lab/solution/lab7/s4s.pas

  This child program first prints out a message sent from the parent
  and then calculates the area of a square.  The area is passed back
  to the parent, so that the parent program can print it out.         }

$run_string 0$
program S4S ;

type
     int    = -32768..32767;
     ptype = packed array [1..5] of int;
     buftype = packed array [1..50] of char;
     outtype = packed array [1..6] of char;

var
     message,
     bufr         : buftype;
     where        : outtype;
     pram         : ptype;
     outer,
     inp, out     : text;
     side, area : real;
     lu, bufln    : int;

 procedure rmpar ( VAR pram : ptype);  external;

 procedure prtn ( area : real);  external;

 procedure cnumd (num : int; buffer : outtype);  external;

 procedure execl4 $alias 'exec'$ (ecode,rcode : int; var bufr : buftype;
           var len : int);  external;

 begin

     rmpar(pram);
     lu := pram[1];

     bufln := -50;

     execl4(14,1,bufr,bufln);

     cnumd(lu,where);
     rewrite(out,where);
     writeln(out,'The buffer passed from parent is : ',bufr);

     rewrite(outer,'1');
     reset(inp,'1');

     writeln (outer,'side: _');
```

```
    read (inp,side);
    area := side * side;

    if side > 0
       then writeln (outer,'area =', area:4:2)
       else writeln (outer,'invalid data');

    message := 'The area of the square is : ';
    bufln := -30;
    execl4(14,2,message,bufln);

    prtn(area);
end.
```

```
{                    /lab/solution/lab7/s4t.pas

  This child program first prints out a message sent from the parent
  and then calculates the area of a triangle. The area is passed back
  to the parent, so that the parent program can print it out.        }

$run_string 0$
program S4T ;

type
     int   = -32768..32767;
     ptype = packed array [1..5] of int;
     buftype = packed array [1..50] of char;
     outtype = packed array [1..6] of char;

var
     message,
     buffer    : buftype;
     where     : outtype;
     pram      : ptype;
     outer,
     inp, out  : text;
     base, height, area : real;
     lu, bufln : int;

procedure rmpar ( VAR pram : ptype);  external;

procedure prtn ( VAR area : real);  external;

procedure cnumd (num : int; buffer : outtype);  external;

procedure execl4 $alias 'exec'$ (ecode,rcode : int; var bufr : buftype;
          var len : int);  external;

begin
     rmpar(pram);
     lu := pram[1];

     bufln := -50;
     execl4(14,1,buffer,bufln);

     cnumd(lu,where);
     rewrite(out,where);
     writeln(out,'The buffer passed from the parent is : ',buffer);

     rewrite(outer,'1');
     reset(inp,'1');

     writeln (outer,'base: _');
     read (inp,base);
     writeln (outer,'height: _');
```

```
    read (inp,height);
    area := 0.5 * base * height;

    if (base > 0) and (height > 0)
       then writeln (outer,'area =', area:4:2)
       else writeln (outer,'invalid data');

    message := 'The area of the triangle is : ';
    bufln := -30;
    execl4(14,2,message,bufln);

    prtn(area);

end.
```

```
FTN7X,L
C     /LAB/SOLUTION/LAB8/S1A.FTN
C
C     This parent program places values into SAM via class I/O
C
      PROGRAM PARENT
      DIMENSION IBUF(20), ISON(3)
      DATA ISON/6HCHILD /

C**************************************************
C*    Initial some variables                      *
C**************************************************

      J = 1
      ISIZE = 20
      ICLAS = 0

C**************************************************
C*    Give user prompt to user                    *
C**************************************************

      WRITE (1,100)
100   FORMAT(/"ENTER MESSAGE OF 40 CHAR OR LESS.  ENTER XX ",
     +"TO TERMINATE.")

C**************************************************
C*    Start loop to receive and send messages     *
C**************************************************

200   WRITE (1,'(/"MESSAGE ",I3," :> _")') J

C**************************************************
C*    Clear user array to ready it for next read  *
C**************************************************

      DO 400, I = 1,20
        IBUF(I) = 2H
400   CONTINUE

C**************************************************
C*    Read message into user's array              *
C**************************************************

      ICODE = 1
      CALL EXEC(ICODE+100000B,1+400B,IBUF,ISIZE,*1000)
C
C  Call ABREG, so that IB will contain the actual size of the buffer
C
      CALL ABREG( IA, IB)

C**************************************************
```

```
C*     Allocate class ownership                           *
C**************************************************

      IFUNC = 1
      CALL CLRQ (IFUNC,ICLAS)

500   IF (IBUF(1) .NE. 2HXX) THEN
        J = J+1
        ICODE = 20
        CALL EXEC(ICODE+100000B,0,IBUF,IB,N,M,ICLAS,*1000)
600     GOTO 200

C**************************************************
C*      Schedule son assigning him class ownership *
C**************************************************

      ELSE
        IFUNC = 1
        CALL CLRQ (IFUNC,ICLAS,ISON)
        ICODE=10
        CALL EXEC(ICODE+100000B,ISON,ICLAS,J,ISIZE,*1000)
      END IF
      STOP

C**************************************************
C*    Error messages                                  *
C**************************************************

1000  CALL ABREG (IA,IB)
      WRITE (1,1100) ICODE,IA,IB
1100  FORMAT(/"ERROR IN EXEC CALL",I3,"  ERROR ",A2," NUMBER ",A3)
      END
```

```
FTN7X,L
C     /LAB/SOLUTION/LAB8/S1B.FTN
C
C     This child program retrieves values from SAM.
C
      PROGRAM CHILD
      DIMENSION IBUF(20), IPM(5)

C***********************************************
C*    Get parameters passed by PARENT          *
C***********************************************


      CALL RMPAR(IPM)
      ICLAS=IPM(1)
      INUM=IPM(2)
      ISIZE=IPM(3)

C***********************************************
C*    Get buffer and output to user            *
C***********************************************

      DO 1000 I = 1,INUM-1
        CALL EXEC(21+100000B,ICLAS,IBUF,ISIZE,*9000)
        CALL ABREG ( IA, IB)
        WRITE(1,'(20a2)') (IBUF(J),J=1,IB)
C
1000  CONTINUE
      STOP

C***********************************************
C*    Error message                            *
C***********************************************

9000  CALL ABREG(IA,IB)
      WRITE(1,9100) IA,IB
9100  FORMAT(/"ERROR IN EXEC, ERROR ",A2," NUMBER ",A3)
      END
  {
     /lab/solution/lab8/sla.pas

     This parent program stores values in SAM via Class I/O
  }

PROGRAM parent (INPUT,OUTPUT);
TYPE
        int = -32768..32767;
        stype = packed array [1..6] of char;
        btype = packed array [1..50] of char;
        rtype = packed array [1..12] of char;
```

```
VAR
        ibuf : btype;
        ison : stype;
        runstr : rtype;
        j, isize, iclas, icode, ifunc   : int;

PROCEDURE EXEC_20 $ALIAS 'EXEC'$
        (icode,icnwd : int; ibuf : btype; isize,iop1,iop2,iclas : int);
        EXTERNAL;

PROCEDURE CLRQ $ALIAS 'CLRQ'$
        (ifunc, iclas : int);  EXTERNAL;

PROCEDURE EXEC_9 $ALIAS 'EXEC'$
        (icode : int; ison : stype; iclas,dum1,dum2,dum3,dum4 : int;
         runst : rtype; runlen : int);  EXTERNAL;

BEGIN
        runstr := 'ru,CHILD,1,1';
        ison := 'CHILD';

        isize := -50;
        iclas := 0;
        ifunc := 1;
        j := 1;

        clrq(ifunc,iclas);

        writeln;
        writeln('Enter message of 50 char or less. Enter XX to terminate.');
        writeln;

        REPEAT
              writeln('Message #',j:3,' :> _');
              read(ibuf);
              j := j+1;
              icode := 20;
              exec_20(icode,0,ibuf,isize,0,0,iclas);
        UNTIL (ibuf = 'XX');

        icode := 9;
        exec_9(icode,ison,iclas,0,0,0,0,runstr,-12);
END.
```

```
{   /lab/solution/lab8/slb.pas

    This child program retrieves the values from the SAM buffer. }

PROGRAM child (input,output);
TYPE
        int = -32768..32767;
        btype = packed array [1..50] of char;
        ptype = array [1..5] of int;

VAR
        ibufr : btype;
        pram  : ptype;
        iclas, isize, icode : int;

PROCEDURE EXEC_21 $ALIAS 'EXEC'$
        (ICODE,ICLAS : int; IBUFR : btype; ISIZE : int);  EXTERNAL;
{ Exec 21 call - class get }

PROCEDURE PARAMS $ALIAS 'Pas.NumericParms'$
        (VAR PRAM : ptype);  EXTERNAL;
{ Pick up the parameters sent from the 'parent' program }

BEGIN
        params(pram);
        iclas := pram[1];          { Pram[1] contains the class number }

        isize := -50;
        icode := 21;

        { GET THE DATA FROM SAM AND PUT IT IN 'IBUFR' }

        exec_21(icode,iclas,ibufr,isize);

        IF (ibufr = 'XX')
            THEN  write('There were no messages sent from the parent !!');

        WHILE (ibufr <> 'XX') DO
            BEGIN

                { PRINT THE DATA }

                write('The buffer passed from the parent is : ');
                writeln(ibufr);

                exec_21(icode,iclas,ibufr,isize);
            END;
        {}
END.
```

```
FTN7X,L
C
C     /LAB/SOLUTION/LAB8/S2A.FTN
C
C     This parent program places values in SAM via class I/O.  The son
C     is scheduled after the first buffer in placed in SAM, instead of
C     waiting for the parent to finish.
C
      PROGRAM PARENT
      DIMENSION IBUF(20), ISON(3)
      DATA ISON/6HCHILD /

C***********************************************
C*    Initial some variables                   *
C***********************************************

      J = 1
      ISIZE = 20
      IFUNC = 1
      ICLAS = 0
      CALL CLRQ(IFUNC,ICLAS,ISON)

C***********************************************
C*    Give user prompt to user                 *
C***********************************************

      WRITE(1,100)
100   FORMAT(/"ENTER MESSAGE OF 40 CHAR OR LESS.  ENTER XX ",
     +"TO TERMINATE.")

C***********************************************
C*    Start loop to receive and send messages  *
C***********************************************

200   WRITE(1,'(/"MESSAGE",I2," :> _")') J

C***********************************************
C*    Clear buffer to ready it for new message *
C***********************************************

      DO 400, I=1,20
        IBUF(I)=2H
400   CONTINUE

C***********************************************
C*    Read the user's message into a user array *
C***********************************************

      ICODE=1
      CALL EXEC(ICODE+100000B,1+400B,IBUF,ISIZE,*1000)
C
```

```
C Call ABREG, so that IB can contain the actual size of the buffer
C
      CALL ABREG ( IA, IB)

C***********************************************
C*    Pass buffer to SAM                       *
C***********************************************

500   ICODE=20
      CALL EXEC(ICODE+100000B,0,IBUF,IB,N,M,ICLAS,*1000)

C***********************************************
C*    If first message then schedule son       *
C***********************************************

600   IF (J .NE. 1) GOTO 700
        ICODE=10
        CALL EXEC(ICODE+100000B,ISON,ICLAS,ISIZE,*1000)

C***************************************************
C*    Terminate program if XX is received          *
C***************************************************

700   IF (IBUF(1) .EQ. 2HXX) STOP
      J = J+1
800   GOTO 200

C***************************************************
C*    Error messages                               *
C***************************************************

1000  CALL ABREG(IA,IB)
      WRITE(1,1100) ICODE,IA,IB
1100  FORMAT(/"ERROR IN EXEC CALL",I3,"  ERROR ",A2," NUMBER ",A3)
      END
```

```
FTN7X,L
C
C     /LAB/SOLUTION/LAB8/S2B.FTN
C
C     This child program retrieves buffers from SAM.  The child is
C     scheduled after the first buffer is inputted.
C
      PROGRAM CHILD
      DIMENSION IBUF(20), IPM(5)

C***********************************************
C*    Get parameters passed by PARENT          *
C***********************************************

      CALL RMPAR(IPM)
      ICLAS=IPM(1)
      ISIZE=IPM(2)

C***********************************************
C*    Get buffer from SAM                      *
C***********************************************

      CALL EXEC(21+100000B,ICLAS+20000B,IBUF,ISIZE,*9000)

      DO WHILE (IBUF(1) .NE. 2HXX)

C***********************************************
C*    Output buffer                            *
C***********************************************

        WRITE (1,'("THE MESSAGE PASSED FROM THE PARENT IS : ")')
        WRITE (1,'(/20A2)') (IBUF(J), J=1,ISIZE)

        DO 400, I = 1,20
           IBUF(I) = 2H
400     CONTINUE

        CALL EXEC(21+100000B,ICLAS+20000B,IBUF,ISIZE,*9000)
      END DO

C***********************************************
C*    Do an extra GET to clean up the class number*
C***********************************************

      CALL EXEC(21+100000B,ICLAS,IBUF,ISIZE,*9000)
      STOP

C***********************************************
C*    Error message                            *
C***********************************************
```

```
9000  CALL ABREG(IA,IB)
      WRITE(1,9100) IA,IB
9100  FORMAT(/"ERROR IN EXEC, ERROR ",A2," NUMBER ",A3)
      END
```

```
FTN7X,L
C
C     /LAB/SOLUTION/LAB8/S2B.FTN
C
C     This child program retrieves buffers from SAM.  The child is
C     scheduled after the first buffer is inputted.
C
      PROGRAM CHILD
      DIMENSION IBUF(20), IPM(5)

C*************************************************
C*    Get parameters passed by PARENT           *
C*************************************************

      CALL RMPAR(IPM)
      ICLAS=IPM(1)
      ISIZE=IPM(2)

C*************************************************
C*    Get buffer from SAM                       *
C*************************************************

      CALL EXEC(21+100000B,ICLAS+20000B,IBUF,ISIZE,*9000)

      DO WHILE (IBUF(1) .NE. 2HXX)

C*************************************************
C*    Output buffer                             *
C*************************************************

        WRITE (1,'("THE MESSAGE PASSED FROM THE PARENT IS : ")')
        WRITE (1,'(/20A2)') (IBUF(J), J=1,ISIZE)

        DO 400, I = 1,20
           IBUF(I) = 2H
 400    CONTINUE

        CALL EXEC(21+100000B,ICLAS+20000B,IBUF,ISIZE,*9000)
      END DO

C*************************************************
C*    Do an extra GET to clean up the class number*
C*************************************************

      CALL EXEC(21+100000B,ICLAS,IBUF,ISIZE,*9000)
      STOP

C*************************************************
C*    Error message                             *
C*************************************************
```

```
9000  CALL ABREG(IA,IB)
      WRITE(1,9100) IA,IB
9100  FORMAT(/"ERROR IN EXEC, ERROR ",A2," NUMBER ",A3)
      END
```

```
ftn7x,l
C
C    /lab/solution/lab8/s3a.ftn
C
C    This program uses class numbers to pass a message to the square
C    program.
C
      program parent

      implicit integer (a-z)
      integer message(25),square(3)
      data square/'SQUARE'/

      iclas = 0
      ifunc = 1
      call clrq (ifunc,iclas)

      write(1,'("Enter a message : ")')
      read (1,5) message
5     format(50a)
      write(1,'("Enter lu to be printed at : ")')
      read (1,*) lu

      call exec(20 + 100000B, 0, message, -50, n, n, iclas, *1000)

      ifunc = 1
      call clrq(ifunc,iclas,square)
      call exec (9, square, lu, iclas)

      stop

1000  call abreg(ia,ib)
      write(1,1100) icode,ia,ib
1100  format('Error in SQUARE, Call ',I3,' Error ',2a2)

      end
```

```
ftn7x,l
C
C       /lab/solution/lab8/s3s.ftn
C
C     This child program recieves a message from the parent via
C     class I/O.
C
      program SQUARE

      integer parm(5), message(25)
      real side, area

      call rmpar(parm)
      lu = parm(1)
      iclas = parm(2)

      call exec(21,iclas,message,-50)

      write (1,'("side: _")')
      read (1,*) side
      area = side * side
      if (side .GT. 0) then
         write (lu,'("The message sent from above is : ",50a)') message
         write (lu,'("area = ",f4.2)') area
      else
         write (lu,'("invalid data")')
      end if

      end
```

```
{
    /lab/solution/lab8/s2a.pas

    This parent program stored values in SAM via Class I/O.
}

PROGRAM parent (INPUT,OUTPUT);
TYPE
        int = -32768..32767;
        stype = packed array [1..6] of char;
        btype = packed array [1..50] of char;
        rtype = packed array [1..12] of char;

VAR
        ibuf : btype;
        ison : stype;
        runstr : rtype;
        j, isize, iclas, icode, ifunc   : int;

PROCEDURE EXEC_20 $ALIAS 'EXEC'$
        (icode,icnwd : int; ibuf : btype; isize,iop1,iop2,iclas : int);
        EXTERNAL;

PROCEDURE SETOWN $ALIAS 'CLRQ'$
        (ifunc, iclas : int; ison : stype);  EXTERNAL;

PROCEDURE EXEC_10 $ALIAS 'EXEC'$
        (icode : int; ison : stype; iclas,dum1,dum2,dum3,dum4 : int;
         runst : rtype; runlen : int);  EXTERNAL;

BEGIN
        runstr := 'ru,CHILD,1,1';
        ison := 'CHILD ';

        isize := -50;
        iclas := 0;
        ifunc := 1;
        j := 1;

        setown(ifunc,iclas,ison);

        writeln;
        writeln('Enter message of 50 char or less.  Enter XX to terminate.');
        writeln;

        REPEAT
              writeln('Message #',j:3,' :> ');
              readln(ibuf);
              icode := 20 ;
              exec_20(icode,0,ibuf,isize,0,0,iclas);
              IF  j = 1  THEN
```

```
                BEGIN
                     icode := 10;
                     exec_10(icode,ison,iclas,0,0,0,0,runstr,-12);
                END;
            j := j + 1;
        UNTIL (ibuf = 'XX');

END.
PROGRAM child (input,output);
CONST
        sc_bit = 8192;

TYPE
        int = -32768..32767;
        btype = packed array [1..50] of char;
        ptype = array [1..5] of int;

VAR
        ibufr : btype;
        pram  : ptype;
        iclas, isize, icode : int;

PROCEDURE EXEC_21 $ALIAS 'EXEC'$
        (ICODE,ICLAS : int; IBUFR : btype; ISIZE : int);  EXTERNAL;
{ Exec 21 call - class get }

PROCEDURE PARAMS $ALIAS 'Pas.NumericParms'$
        (VAR PRAM : ptype);  EXTERNAL;
{ Pick up the parameters sent from the 'parent' program }

PROCEDURE CLRQ (ifunc,iclas : int);  EXTERNAL;

BEGIN
        params(pram);              { Pram[1] contains the class number }
        iclas := pram[1] + sc_bit;
        isize := -50;
        icode := 21;

        { GET THE DATA FROM SAM AND PUT IT IN 'IBUFR' }

        exec_21(icode,iclas,ibufr,isize);

        IF (ibufr = 'XX')
            THEN  writeln('There were no messages sent from the parent !!');

        WHILE (ibufr <> 'XX') DO
            BEGIN
                { PRINT THE DATA }
                write('The buffer passed from the parent is : ');
                writeln(ibufr);
```

```
            exec_21(icode,iclas,ibufr,isize);
        END;
    clrq(2,iclas);
END.
```

```
{                     /lab/solution/lab8/s3m.pas
  This parent program uses Class I/O to send messages to the child
  program - the 'square' program.  The child then computes the area
  of the square and prints it out.                                  }

program s3m (input, output);

type
     int = -32768..32767;
     progname = packed array [1..6] of char;
     msgtype = packed array [1..50] of char;

var
    selection : integer;
    square    : progname;
    message   : msgtype;
    func,
    lu,class  : int;

procedure exec9 $alias 'exec'$ (icode: int; prog: progname; lu,class : int);
          external;

procedure exec20 $ alias 'exec' $ (icode, dum : int; message : msgtype; len,
          dum1, dum2, class : int);  external;

procedure clrq (func, class : int; name : progname);  external;

begin
   square := 'S3S';

   class := 0;
   func  := 1;
   clrq (func,class,'S3S');

   writeln ('area program');
   writeln('Enter a message to be printed by the child : ');
   read(message);
   writeln('Enter lu number to print the message at : ');
   read(lu);

   exec20(20,0,message,-50,0,0,class);

   clrq(func,class,square);
   exec9 (9, square, lu, class);
end.
```

```
{                 /lab/solution/lab8/s3s.pas
  This child program is sent a message from the parent using Class I/O.
  The child then calculates the area of a square and prints it out.     }

program S3S ;

type
     int    = -32768..32767;
     ptype = packed array [1..5] of int;
     msgtype = packed array [1..50] of char;
     outtype = packed array [1..6] of char;
var
     message     : msgtype;
     where       : outtype;
     pram        : ptype;
     outer,
     inp, out    : text;
     side, area  : real;
     lu, class   : int;

procedure rmpar ( VAR pram : ptype);  external;

procedure cnumd (num : int; buffer : outtype);  external;

procedure exec21 $alias 'exec'$ (icode, class : int; message : msgtype;
          len : int);  external;
begin
    rmpar(pram);
    lu := pram[1];
    class := pram[2];

    exec21(21,class,message,-50);

    cnumd(lu,where);
    rewrite(out,where);

    rewrite(outer,'1');
    reset(inp,'1');

    writeln (outer,'side: _');
    read (inp,side);
    area := side * side;
    if side > 0 then
       begin
            writeln (out,'The message sent from above is : ',message);
            writeln (outer,'area =', area:4:2);
       end;
    if side < 0 then writeln (outer,'invalid data');

end.
```

```
FTN7X,L
C
C     /LAB/SOLUTION/LAB8/S4A.FTN
C
      PROGRAM PARENT
      DIMENSION IBUF(20),ISON(3)
      DATA  ISON /6HCHILD /,J/0/

C     Allocate two class numbers assign ownership to the father
C     then schedule the son.
C
      IFUNC = 1
      CALL CLRQ(IFUNC,ICLAS1)
      CALL CLRQ(IFUNC,ICLAS2)
      ICODE=24
      CALL EXEC(ICODE+100000B,ISON,ICLAS1,ICLAS2,*1000)
C
C     Prompt for the A buffer
C
5     WRITE (1,100)
100   FORMAT('ENTER A MESSAGE OF 40 CHARACTERS OR LESS. XX TO END')

C     Read the user buffer, pick up the transmission log

      ICODE=1
      CALL EXEC(ICODE+100000B,1+400B,IBUF,20,*1000)
      CALL ABREG(IA,IB)

C     Pass the buffer to SAM

      ICODE=20
      CALL EXEC(ICODE+100000B,0,IBUF,IB,N,N,ICLAS1,*1000)
      J=J+1

C     Check for XX to end

      IF (IBUF(1) .EQ. 2HXX) GOTO 10

C     Check for four buffers in SAM

      IF( J .LT.4) GOTO 5

C     Four in SAM already, suspend until son says he's consumed one

      ICODE = 21
      CALL EXEC(ICODE+100000B,ICLAS2+20000B,I,1,*1000)
      J = J - 1
      GO TO 5

C     Ready to end, be sure all messages received
```

```
10      DO WHILE (J.NE.0)
          ICODE = 21
          CALL EXEC(ICODE+100000B,ICLAS2+20000B,I,1,*1000)
          J = J - 1
        END DO
        STOP

C       Handle EXEC errors

1000    CALL ABREG(IA,IB)
        WRITE(1,200) ICODE,IA,IB
200     FORMAT('ERROR IN L174A, CALL',I3, ' ERROR ',2A2)
        END
```

```
FTN7X,L
C
C     /LAB/SOLUTION/LAB8/S4B.FTN
C
      PROGRAM CHILD
      DIMENSION IBUF(20),IPRAM(5)

C     Pick up class numbers with RMPAR

      CALL RMPAR(IPRAM)
      ICL1 = IPRAM(1)
      ICL2 = IPRAM(2)

C     Pick up a buffer from the father, let father know he got it
C
10    ICODE = 21
      CALL EXEC(ICODE+100000B,ICL1+20000B,IBUF,20,*1000)
      CALL ABREG(IA,IB)
      ICODE=20
      CALL EXEC(ICODE+100000B,0,I,1,N,N,ICL2,*1000)

C     Check for the last buffer, print the buffer

      IF (IBUF(1) .EQ. 2HXX) GO TO 20
      WRITE(1,100) (IBUF(I),I=1,IB)
100   FORMAT(' MESSAGE FROM PARENT IS ',20A2)

C     Go get the next buffer

      GO TO 10

C     Got the last buffer, so end.

20    STOP

C     EXEC error reporting section

1000  CALL ABREG(IA,IB)
      WRITE(1,200) ICODE,IA,IB
200   FORMAT('L174B EXEC ERROR CALL',I3,' ERROR  ',2A2)
      END
```

```
{
      /lab/solution/lab8/s4a.pas

PROGRAM parent(input,output);
CONST
      sc_bit = 8192;

TYPE
      int = -32768..32767;
      stype = packed array [1..6] of char;
      btype = packed array [1..50] of char;
      rtype = packed array [1..12] of char;

VAR
      ibuf : btype;
      ison : stype;
      runstr : rtype;
      i, j, isize, clas1, clas2, icode, ifunc : int;

PROCEDURE EXEC_20 $ALIAS 'EXEC'$
      (icode,icnwd : int; ibuf : btype; isize,iop1,iop2,clas1 : int);
      EXTERNAL;

PROCEDURE EXEC_21 $ALIAS 'EXEC'$
      (icode, clas2 : int; i : int; len : int);  EXTERNAL;

PROCEDURE CLRQ $ALIAS 'CLRQ'$
      (ifunc, clas1 : int);  EXTERNAL;

PROCEDURE EXEC_24 $ALIAS 'EXEC'$
      (icode : int; ison : stype; clas1,clas2 : int); EXTERNAL;

BEGIN
      runstr := 'ru CHILD 1 1';
      ison := 'CHILD';

      isize := -50;
      clas1 := 0;
      clas2 := 0;
      ifunc := 1;
      j := 0;

      clrq(ifunc,clas1);
      clrq(ifunc,clas2);

      icode := 24;
      exec_24(icode,ison,clas1,clas2);

      writeln;
      writeln('Enter message of 50 char or less.  Enter XX to end !');
      writeln;
```

```
      REPEAT
            REPEAT
                  writeln('Message is :> ');
                  read(ibuf);
                  icode := 20 ;
                  exec_20(icode,0,ibuf,isize,0,0,clas1);
                  j := j+1;
            UNTIL ((j = 4) OR (ibuf = 'XX'));

            IF (j = 4) THEN
                 BEGIN
                      icode := 21 ;
                      exec_21(icode,clas2+sc_bit,i,1);
                      j := j-1;
                 END;
      UNTIL (ibuf = 'XX');

END.
```

```
  {
      /lab/solution/lab8/s4b.pas

  }
$run_string 0$
PROGRAM CHILD ;
CONST
        sc_bit = 8192;

TYPE
        int = -32768..32767;
        btype = packed array [1..50] of char;
        ptype = array [1..5] of int;

VAR
        ibufr : btype;
        pram  : ptype;
        i, clas1, clas2, isize, icode : int;
        out : text;

PROCEDURE EXEC_21 $ALIAS 'EXEC'$
          (ICODE,clas1 : int; IBUFR : btype; ISIZE : int);  EXTERNAL;
{ Exec 21 call - class get }

PROCEDURE EXEC_20 $ALIAS 'EXEC'$
          (icode,icnwd :int; i : int; len,iop1,iop2,clas2 : int);
          EXTERNAL;

PROCEDURE PARAMS $ALIAS 'Pas.NumericParms'$
          (VAR PRAM : ptype);  EXTERNAL;
{ Pick up the parameters sent from the 'parent' program }

PROCEDURE CLRQ (ifunc,clas1 : int);  EXTERNAL;

BEGIN
        params(pram);                 { Pram[1] contains the class number }
        clas1 := pram[1] + sc_bit;
        clas2 := pram[2];
        isize := -50;

        rewrite(out,'1');

        { PICK UP BUFFER FROM THE PARENT, LET PARENT KNOW HE GOT IT }

        icode := 21;
        exec_21(icode,clas1,ibufr,isize);
        icode := 20;
        exec_20(icode,0,i,1,0,0,clas2);

        IF (ibufr = 'XX')
          THEN writeln(out,'There were no messages sent from the parent !!');
```

```
    WHILE (ibufr <> 'XX') DO
        BEGIN
            { PRINT THE DATA }
            write(out,'The buffer passed from the parent is : ');
            writeln(out,ibufr);

            icode := 21;
            exec_21(icode,clas1,ibufr,isize);
            icode := 20;
            exec_20(icode,0,i,1,0,0,clas2);
        END;

END.
```

```
FTN7X,L

C         /lab/solution/lab8/s5.ftn
C
      PROGRAM S5
      DIMENSION IBUF(20)

C     ****************************************************
C     * This program will prompt the user for a string of *
C     * characters and print a message on the line printer*
C     * until the user responds.                           *
C     ****************************************************

C     Prompt user
C
      WRITE (1,*) 'ENTER SOME RESPONSE TO STOP PRINTER!'

C     Allocate class ownership
C
      ICLAS = 0
      IFUNC = 1
      CALL CLRQ (IFUNC,ICLAS)

C     Read reply into SAM buffer
C
      ICODE=17
      CALL EXEC(ICODE+100000B,1+400B,IBUF,20,N,M,ICLAS,*1000)

C     Try to get buffer read in with no wait
C
150   ICODE=21
200   CALL EXEC (ICODE+100000B,ICLAS+100000B,IBUF,20,*1000)
C
C     If buffer was present for Class Get jump out of loop
C
250   CALL ABREG (IA,IB)
      DO WHILE (IA .LT. 0)

C       Give message through LINE PRINTER

300     WRITE(6,'(/" PLEASE KILL ME QUICKLY!")')
        CALL EXEC (ICODE+100000B,ICLAS+100000B,IBUF,20,*1000)
        CALL ABREG (IA,IB)
      END DO
      STOP
C
C     Error messages
C
1000  CALL ABREG(IA,IB)
      WRITE(1,1100) ICODE,IA,IB
1100  FORMAT(/"ERROR IN EXXEC CALL ",I2," ERROR ",A2," NUMBER ",A2)
```

```
      END
PROGRAM linep(input,output);
CONST
        echo_bit = 256;
        nw_bit = -32768;

TYPE
        int = -32768..32767;
        btype = packed array [1..20] of char;

VAR
        bufr : btype;
        a, b, ifunc, iclas, bufln, icode, cntwd : int;
        lu  : text;

PROCEDURE EXEC_21 $ALIAS 'EXEC'$
          (ICODE,ICLAS : int; IBUFR : btype; ISIZE : int);  EXTERNAL;

PROCEDURE EXEC_17 $ALIAS 'EXEC'$
          (icode,cntwd : int; bufr : btype; bufln,dum1,dum2,iclas : int);
          EXTERNAL;

PROCEDURE ABREG (VAR  a, b : int);  EXTERNAL;

PROCEDURE CLRQ (ifunc,iclas : int); EXTERNAL;

BEGIN
      writeln('ENTER SOME RESPONSE TO STOP PRINTER!!');

      rewrite(lu,'6');

      bufln := -20;

      iclas := 0;
      ifunc := 1;
      clrq(ifunc,iclas);

      icode := 17;
      cntwd := 1 + echo_bit;
      exec_17(icode,cntwd,bufr,bufln,0,0,iclas);
      icode := 21 ;
      iclas := iclas + nw_bit;
      exec_21(icode,iclas,bufr,bufln);
      abreg(a,b);
      WHILE  (a < 0) DO
          BEGIN
               writeln(lu,'PLEASE KILL ME QUICKLY');
               exec_21(icode,iclas,bufr,bufln);
               abreg(a,b);
          END;
END.
```

```
FTN7x,L
C
C     /LAB/SOLUTION/LAB9/S1.FTN
C
      PROGRAM CREATE
      DIMENSION IDCB(144), ISIZE(2), IBUF(1)
C
C     This program is designed to create a type 2 file with
C     record length = 1 word, and blocks = 5
C     ***************************************************
C     *                                                 *
C     *    Create the file using an FMP call            *
C     *                                                 *
C     ***************************************************
      ISIZE (1) = 5
      ISIZE (2) = 1
      ITYPE = FMPOPEN(IDCB,IERR,'XXTEMP:::2:5:1','wc',1)
      IF (ITYPE .LT. 0) GOTO 1200
C
C     Write each value to each record
C
 200  J=ISIZE(1)*128/ISIZE(2)
 300  DO 1000 I=1,J
        IBUF(1)=I
C     ***************************************************
C     *                                                 *
C     *  Write to the file using an FMP Call            *
C     *                                                 *
C     ***************************************************
        LENGTH = FMPWRITE(IDCB,IERR,IBUF,2)
        IF (LENGTH .LT. 0) GOTO 1200
 1000 CONTINUE
C
C     Give user message that the process is complete
C
      WRITE(1,*) 'THE FILE HAS BEEN CREATED AND WRITTEN TO !!'
C     ***************************************************
C     *                                                 *
C     *    Close the file using an FMP call             *
C     *                                                 *
C     ***************************************************
      IERROR = FMPCLOSE(IDCB,IERR)
      STOP
C
C     Error in creating the file
C
 1200 CALL FMPREPORTERROR(IERR,'XXTEMP')
      IERROR = FMPCLOSE(IDCB,IERR)
      END
```

```
  {
      /lab/solution/lab9/sl.pas
   }
PROGRAM create(INPUT,OUTPUT);

TYPE
     int = -32768..32767;
     filetype = packed array [1..64] of char;
     optiontype = packed array [1..2] of char;
     dcbtype = array [1..144] of int;
     sizetype = array [1..2] of int;

VAR
     dcb : dcbtype;
     size : sizetype;
     num, len, error, err, recnum, count : int;
     filename : filetype;
     option : optiontype;
     opts, filedesc : integer;

FUNCTION strdsc (filenamr : filetype; startchar, nchars : int): integer;
          EXTERNAL;

FUNCTION optdsc $ ALIAS 'strdsc' $
          (option : optiontype; startchar, nchars : int): integer;
          EXTERNAL;

FUNCTION fmpopen (VAR dcb : dcbtype; VAR error: int; name,options : integer;
                   buffers : int) : int;  EXTERNAL;

FUNCTION fmpwrite (VAR dcb : dcbtype; VAR error, count, length : int) : int;
          EXTERNAL;

FUNCTION fmpclose (dcb : dcbtype; error : int) : int;  EXTERNAL;

PROCEDURE fmpreporterror (VAR error : int; VAR filename : integer);
           EXTERNAL;

BEGIN
      size[1] := 5;
      size[2] := 1;

{  Convert 'xxprog' to a FORTRAN compatible character string.  }

      filename := 'xxprog:::2:5:1';
      filedesc := strdsc(filename,1,64);

{  Convert the option to a FORTRAN compatible character string.  }

      option := 'wc';
      opts := optdsc(option,1,2);
```

```
{  Open the 'xxprog' file.  }

      IF (fmpopen(dcb,err,filedesc,opts,1) < 0)
          THEN fmpreporterror(err,filedesc);
      recnum := size[1] * 128 DIV size[2];
      FOR count := 1 to recnum  DO
          BEGIN
               num := count;
               len := 2;
               IF (fmpwrite(dcb,err,num,len) < 0)
                   THEN  fmpreporterror(err,filedesc);
          END;

     Writeln('The file ''xxprog'' has been created and written to !!');

     error := fmpclose(dcb,err);

END.
```

```
FTN7X,L
C
C     /LAB/SOLUTION/LAB9/S2.FTN
C
      PROGRAM UPDATE
C
      INTEGER IDCB(144), IERR
      CHARACTER*64 FNAME
C
      WRITE(1,*) ' WHAT IS THE NAME OF YOUR FILE? '
      READ(1,102) FNAME
102   FORMAT(A64)

C     ******************************************
C     *          Open the file                 *
C     ******************************************

      ITYPE = FMPOPEN(IDCB,IERR,FNAME,'wou',1)
      IF (ITYPE .LT. 0) GOTO 90

C     ******************************************
C     *     Store 7777B into the first record  *
C     ******************************************

      NEWVAL = 7777B

      LENGTH = FMPWRITE(IDCB,IERR,NEWVAL,1)
      IF (LENGTH .LT. 0) GOTO 90

C     ******************************************
C     *     Close the file and terminate       *
C     ******************************************

      IERROR = FMPCLOSE(IDCB,IERR)
      STOP

C     ******************************************
C     *        Error processing                *
C     ******************************************

90    CALL FMPREPORTERROR(IERR,FNAME)
      IERROR = FMPCLOSE(IDCB,IERR)
C
      END
```

```
{
    /lab/solution/lab9/s2.pas
}
PROGRAM update(INPUT,OUTPUT);

TYPE
    int = -32768..32767;
    filetype = packed array [1..64] of char;
    optiontype = packed array [1..3] of char;
    dcbtype = array [1..144] of int;

VAR
    dcb : dcbtype;
    newval, len, error, err : int;
    filename : filetype;
    option : optiontype;
    opts, filedesc : integer;

FUNCTION strdsc (filenamr : filetype; startchar, nchars : int): integer;
        EXTERNAL;

FUNCTION optdsc $ ALIAS 'strdsc' $
        (option : optiontype; startchar, nchars : int): integer;
        EXTERNAL;

FUNCTION fmpopen (VAR dcb : dcbtype; VAR error: int; name,options : integer;
                  buffers : int) : int;  EXTERNAL;

FUNCTION fmpwrite (VAR dcb : dcbtype; VAR error, count, length : int) : int;
        EXTERNAL;

FUNCTION fmpclose (dcb : dcbtype; error : int) : int;  EXTERNAL;

PROCEDURE fmpreporterror (VAR error : int; VAR filename : integer);
        EXTERNAL;

BEGIN

{  Input the file name.  }

     Writeln('What is the name of your file? ');
     Read(filename);

{  Convert the file to a FORTRAN compatible character string.  }

     filedesc := strdsc(filename,1,64);

{  Convert the option to a FORTRAN compatible character string.  }

     option := 'woc';
     opts := optdsc(option,1,3);
```

```
{  Open the and update the file.  }

      IF (fmpopen(dcb,err,filedesc,opts,1) < 0)
          THEN fmpreporterror(err,filedesc);
      newval := 4095;
      len := 1;
      IF (fmpwrite(dcb,err,newval,len) < 0)
          THEN fmpreporterror(err,filedesc);
      Writeln('Your file has been updated !!');

      error := fmpclose(dcb,err);

END.
```

```
ftn7x,l
C
C     /LAB/SOLUTION/LAB9/S3.FTN
C
C     This program uses FMPRPPROGRAM to programmatically schedule the
C     children - square, triangle, circle.
C
      program FMPAREA

      implicit integer(a-z)
      real area
      integer message(25),buffer(15),circle(3),square(3),triangle(3),parm(5)
      integer return(3)
      data circle/'SL74C'/, square/'SL74S'/, triangle/'SL74T'/

      write (1,'("area program")')
      selection = -1
      do while (selection .NE. 0)
         write(1,'("select one:")')
         write(1,'(
     +        "0 = finished, 1 = circle, 2 = square, 3 = triangle")')
         read (1,*) selection

         if (selection .EQ. 0) then
             write(1,'("finished")')
         else
             write(1,'("Enter a message : ")')
             read (1,5) message
5            format(50a)
             write(1,'("Enter LU to be printed at : ")')
             read(1,*) lu
         endif

         if (selection .EQ. 1)  then
            call fmprpprogram('SL74C.RUN',return,'c',error)
            call exec (9, circle, lu, 0, 0, 0, 0, message, -50)
         endif
         if (selection .EQ. 2)  then
            call fmprpprogram('SL74S.RUN',return,'c',error)
            call exec (9, square, lu, 0, 0, 0, 0, message, -50)
         endif
         if (selection .EQ. 3)  then
            call fmprpprogram('SL74T.RUN',return,'c',error)
            call exec (9, triangle, lu, 0, 0, 0, 0, message, -50)
         endif
         if (selection .NE. 0) then
            call rmpar(area)
            call exec(14,1,buffer,-30)
            write(1,10) buffer
10          format(30a)
            write(1,20) area
```

```
20          format(f7.4)
         endif

      end do
      end
```

```
{                   /lab/solution/lab9/s3m.pas
    This parent program uses FMPRPPROGRAM to schedule the child
    programs - square, circle, triangle - in lab 7 exercise 4. The
    parent sends a message and LU to the child.  The parent later
    receives a message and the area of the specified child program. }

program s3m (input, output);

type
     int = -32768..32767;
     progname = packed array [1..5] of char;
     buftype = packed array [1..30] of char;
     msgtype = packed array [1..50] of char;
     ptype = packed array [1..5] of int;
     rpname = packed array [1..7] of char;

var
    area       : real;
    bufr       : buftype;
    message    : msgtype;
    pram       : ptype;
    retname, circle, square, triangle : progname;
    option, filename, selection : integer;
    bufln, lu, error : int;

procedure exec9 $ alias 'exec' $ (icode : int; prog : progname; lu,dum2,
          dum3,dum4,dum5 : int; VAR message : msgtype; len : int); external;

procedure rmpar ( VAR area : real );  external;

procedure exec14 $ alias 'exec' $ (icode, rcode : int; VAR bufr : buftype;
          var len : int);  external;

function  fmprpprogram (var name : integer;  retname : progname;
          opt : integer; error : int) : int;   external;

function  strdsc $alias 'strdsc'$ (str : rpname; first,last : int)
          : integer;  external;

function  optdsc $alias 'strdsc'$ (opt : char; first,last : int) : integer;
          external;

begin
   circle := 'S4C';
   square := 'S4S';
   triangle := 'S4T';

   option := optdsc('c',1,1);

   writeln ('area program');
   repeat
```

```
        writeln ('select one:');
        writeln ('0 = finished, 1 = circle, 2 = square, 3 = triangle');
        read (selection);
        if (selection <> 0) then
            begin
                writeln('Enter a message to be printed by the child : ');
                read(message);
                writeln('Enter lu number to print the message at : ');
                read(lu);
            end;
        case selection of
           0 : writeln ('finished');
           1 : begin
                   filename := strdsc('S4C.RUN',1,9);
                   error := fmprpprogram(filename,retname,option,error);
                   exec9 (9, circle, lu, 0, 0, 0, 0, message, -50);
               end;
           2 : begin
                   filename := strdsc('S4S.RUN',1,9);
                   error := fmprpprogram(filename,retname,option,error);
                   exec9 (9, square, lu, 0, 0, 0, 0, message, -50);
               end;
           3 : begin
                   filename := strdsc('S4T.RUN',1,9);
                   error := fmprpprogram(filename,retname,option,error);
                   exec9 (9, triangle, lu, 0, 0, 0, 0, message, -50);
               end;
        end;

        if (selection <> 0) then
            begin
                rmpar(area);
                bufln := -30;
                exec14(14,1,bufr,bufln);
                writeln(bufr,area);
            end;

     until selection = 0;

end.
```

```
FTN7X,L
C
C       /LAB/SOLUTION/LAB9/S4A.FTN
C
        PROGRAM COPY1
C
        IMPLICIT INTEGER(A-Z)
C
C PROGRAM TO COPY A FILE TO ANOTHER FILE  USING 'FMPCOPY'
C
        INTEGER BUFFER(528)
        CHARACTER FILE1*64, FILE2*64
C
C GET THE SOURCE AND DESTINATION FILE NAMES
C
        CALL FPARM(FILE1,FILE2)
C
C MAKE THE COPY IN ONE SUBROUTINE CALL  (ASSUME ASCII FILES)
C
        IF (FMPCOPY(FILE1,ERR1,FILE2,ERR2,BUFFER,528,'AD') .GE. 0)  STOP
C
        IF (ERR1 .LT. 0) CALL FMPREPORTERROR(ERR1,FILE1)
        IF (ERR2 .LT. 0) CALL FMPREPORTERROR(ERR2,FILE2)
C
        STOP
        END
```

```
FTN7X,L
C
C       /LAB/SOLUTION/LAB9/S4.FTN
C
        PROGRAM COPY2
C
        IMPLICIT INTEGER(A-Z)
C
C PROGRAM TO COPY A FILE TO ANOTHER FILE USING FMP READS AND WRITES
C
        INTEGER DCB1(528), DCB2(528), BUFFER(128)
        CHARACTER FILE1*64, FILE2*64
C
C GET THE SOURCE AND DESTINATION FILE NAMES
C
        CALL FPARM(FILE1,FILE2)
C
C MAKE THE COPY IN ONE SUBROUTINE CALL  (ASSUME ASCII FILES)
C
        IF (FMPOPEN(DCB1,ERR,FILE1,'ROS',4) .LT. 0)   GO TO 10
        IF (FMPOPEN(DCB2,ERR,FILE2,'WOC',4) .LT. 0)   GO TO 20
C
        DO WHILE (.TRUE.)
             LEN = FMPREAD(DCB1,ERR,BUFFER,256)
             IF (ERR .LT. 0) GO TO 10
             IF (LEN .EQ. -1) GO TO 30
             IF (FMPWRITE(DCB2,ERR,BUFFER,LEN) .LT. 0) GO TO 20
        END DO
C
10      CALL FMPREPORTERROR(ERR,FILE1)
        GO TO 30
20      CALL FMPREPORTERROR(ERR,FILE2)
30      CALL FMPCLOSE(DCB1,ERR)
        CALL FMPCLOSE(DCB2,ERR)
C
        STOP
        END
```

```
{
   /lab/solution/lab9/s4a.pas                                  }

PROGRAM copy1(INPUT,OUTPUT);

TYPE
     int = -32768..32767;
     buffertype = packed array [1..64] of char;
     intarray = array [1..528] of int;
VAR
     startchar, nchars, result, err1, err2 : int;
     filenamr, file1, file2 : buffertype;
     filedesc1, filedesc2 : integer;
     buffer : intarray;

FUNCTION strdsc (filenamr : buffertype; startchar, nchars : int): integer;
       EXTERNAL;

FUNCTION fmpcopy(VAR file1 : integer; VAR error1 : int; VAR file2 : integer;
                 VAR error2 : int; copybuffer : intarray; length : int): int;
       EXTERNAL;

PROCEDURE fmpreporterror (VAR error : int; VAR filename : integer);
        EXTERNAL;

BEGIN

{  Input the source and destination file names.  }

      writeln('Enter the name of the source file to copy : ');
      read(file1);
      writeln('Enter the name of the destination file : ');
      read(file2);

{  Convert the file names to a FORTRAN compatible character string.  }

      filedesc1 := strdsc(file1,1,64);
      filedesc2 := strdsc(file2,1,64);

{  Copy the source file to the destination file using FMPCOPY.  }

      result := fmpcopy(filedesc1,err1,filedesc2,err2,buffer,528);

{  Error checking.  }

      IF (result < 0) THEN writeln ('copy failed, error = ', result);
      IF (err1 < 0) THEN fmpreporterror (err1,filedesc1);
      IF (err2 < 0) THEN fmpreporterror (err2,filedesc2);

END.
```

```
 {
    /lab/solution/lab9/s4b.pas
  }
PROGRAM copy2(INPUT,OUTPUT);

TYPE
     int = -32768..32767;
     buffertype = packed array [1..64] of char;
     optiontype = packed array [1..3] of char;
     intarray = array [1..528] of int;
     barray = array [1..128] of int;

VAR
     I, length, err, error : int;
     file1, file2 : buffertype;
     opt1, opt2, filedesc1, filedesc2 : integer;
     dcb1, dcb2 : intarray;
     buffer : barray;
     rosstring, wocstring : optiontype;

FUNCTION strdsc (filenamr : buffertype; startchar, nchars : int): integer;
         EXTERNAL;

FUNCTION optdsc $ ALIAS 'strdsc' $
         (option : optiontype; startchar, nchars : int): integer;
         EXTERNAL;

FUNCTION fmpopen (VAR dcb : intarray; VAR error: int; name,options : integer;
                  buffers : int) : int;  EXTERNAL;

FUNCTION fmpread (VAR dcb : intarray; VAR error : int; buffer : barray;
                  maxlen : int) : int;  EXTERNAL;

FUNCTION fmpwrite (VAR dcb : intarray; VAR error : int; buffer : barray;
                   length : int) : int; EXTERNAL;

FUNCTION fmpclose (dcb : intarray; error : int) : int;  EXTERNAL;

PROCEDURE close;
    BEGIN
          error := fmpclose(dcb1,err);
          error := fmpclose(dcb2,err);
    END;

PROCEDURE fmpreporterror (VAR error : int; VAR filename : integer);
          EXTERNAL;

BEGIN

{  Input the source and destination file names.  }
```

```
    writeln('Enter the name of the source file to copy : ');
    read(file1);
    writeln('Enter the name of the destination file : ');
    read(file2);

{  Convert the file names to a FORTRAN compatible character string.  }

    filedesc1 := strdsc(file1,1,64);
    filedesc2 := strdsc(file2,1,64);

{  Convert the options to a FORTRAN compatible character string.  }

    rosstring := 'ros';
    wocstring := 'wc';

    opt1 := optdsc(rosstring,1,3);
    opt2 := optdsc(wocstring,1,2);

{  Open the source and the destination files.  }

    IF (fmpopen(dcb1,err,filedesc1,opt1,4) < 0)
        THEN fmpreporterror(err,filedesc1);
    IF (fmpopen(dcb2,err,filedesc2,opt2,4) < 0)
        THEN fmpreporterror(err,filedesc2);
    WHILE (length <> -1)  DO
        BEGIN
             length := fmpread(dcb1,err,buffer,256);
             IF (length <> -1)  THEN
                 BEGIN
                     IF (fmpwrite(dcb2,err,buffer,length) < 0)
                         THEN  fmpreporterror(err,filedesc2);
                 END;
             IF (length < 0) AND (length = err)
                 THEN fmpreporterror(err,filedesc1);
        END;

    close;

END.
```

```
FTN7X,L
C
C       /LAB/SOLUTION/LAB9/S5.FTN
C
C       THIS PROGRAM DETERMINES THE PARENT DIRECTORY OF YOUR WORKING DIRECTORY
C
        PROGRAM DIRECT
C
        INTEGER ERROR, TRIMLEN, LENGTH, IFOUND(10), NEWLEN
        CHARACTER * 64 WD, NEWWD
        DATA IFOUND /10*0/
C
C FIRST DETERMINE WHAT YOUR WORKING DIRECTORY IS
C
        ERROR = FMPWORKINGDIR(WD)
        IF (ERROR .LT. 0) THEN
            CALL FMPREPORTERROR(ERROR)
            STOP
        ENDIF
C
C PARSE THE WORKING DIRECTORY TO DETERMINE WHAT THE PARENT DIRECTORY IS
C
        CALL FMPHIERARCHNAME(WD)
        LENGTH = TRIMLEN(WD)
        I = 1
        DO J = 1,LENGTH
             IF (WD(J:J) .EQ. '/') THEN
                 IFOUND(I) = J
                 I = I + 1
             ENDIF
        END DO
        I = 1
        DO WHILE (IFOUND(I) .NE. 0)
             I = I + 1
        END DO
        NEWLEN = IFOUND(I-1) - 1
        NEWWD = WD(1:NEWLEN)
C
C SET THE WORKING DIRECTORY TO THE PARENT DIRECTORY
C
        ERROR = FMPSETWORKINGDIR(NEWWD)
        IF (ERROR .LT. 0) THEN
             CALL FMPREPORTERROR(ERROR)
             STOP
        ENDIF
        END
```

```
  {
      /lab/solution/lab9/s5.pas
  }
PROGRAM direct(input,output);
TYPE
      int = -32768..32767;
      intarray = array [1..10] of int;
      dirtype = packed array [1..64] of char;

VAR
      error, length, newlen  : int;
      ifound : intarray;
      i, j, wd, newwd : integer;
      initdir, newdir : dirtype;

FUNCTION strdsc(dirnam : dirtype; startchar,nchars : int): integer; EXTERNAL;

FUNCTION fmpworkingdir(dir : integer) : int;  EXTERNAL;

FUNCTION fmpsetworkingdir(VAR dir : integer) : int;  EXTERNAL;

FUNCTION trimlen(dirstring : integer) : int;  EXTERNAL;

PROCEDURE fmphierarchname(dir : integer);  EXTERNAL;

PROCEDURE fmpreporterror(VAR error : int);  EXTERNAL;

BEGIN

{  FIRST DETERMINE WHAT YOUR WORKING DIRECTORY IS  }

       wd := strdsc(initdir,1,64);
       error := fmpworkingdir(wd);
       IF (error < 0) THEN  fmpreporterror(error);

{  DETERMINE WHAT THE PARENT DIRECTORY IS  }

       fmphierarchname(wd);
       length := trimlen(wd);
       i := 1;
       FOR j := 1 TO length DO
           BEGIN
                IF (initdir[j] = '/') THEN
                    BEGIN
                         ifound[i] := j;
                         i := i + 1;
                    END;
           END;
       i := 1;
       WHILE (ifound[i] <> 0)  DO
            i := i + 1;
```

```
        newlen := ifound[i-1] - 1;
        FOR j := 1 TO newlen DO
             newdir[j] := initdir[j];

{  SET THE WORKING DIRECTORY TO THE PARENT DIRECTORY    }

        newwd := strdsc(newdir,1,64);
        error := fmpsetworkingdir(newwd);
        IF (error < 0) THEN  fmpreporterror(error);
        writeln('Your new working directory is now : ',newdir);

END.
```

```
FTN7X,L
C
C     /LAB/SOLUTION/LAB9/S6.FTN
C
C     THIS PROGRAM ACCEPTS  A PARAMETER WHICH INDICATES THE NUMBER OF
C     LEVELS UP IN THE TREE STRUCTURE AND SET THAT DIRECTORY TO YOUR
C     WORKING DIRECTORY.   A '-1' SETS YOUR GLOBAL DIRECTORY TO YOUR
C     WORKING DIRECTORY.
C
      PROGRAM TREE
C
      INTEGER ERROR, TRIMLEN, LENGTH, IFOUND(10), NEWLEN
      INTEGER IPAR(5), LEVEL, TIMES
      CHARACTER * 64 WD, NEWWD
      DATA IFOUND /10*0/
C
C PICK UP THE PARAMETERS PASSED IN THE RUN STRING
C
      CALL RMPAR(IPAR)
      LEVEL = IPAR(1)
C
C DETERMINE WHAT YOUR WORKING DIRECTORY IS
C
      ERROR = FMPWORKINGDIR(WD)
      IF (ERROR .LT. 0) THEN
          CALL FMPREPORTERROR(ERROR)
          STOP
      ENDIF
C
C PLACE THE WORKING DIRECTORY IN HIERARCHIAL FORM
C
      CALL FMPHIERARCHNAME(WD)
      LENGTH = TRIMLEN(WD)
      I = 1
C
C DETERMINE WHERE THE SLASHES ARE IN THE WORKING DIRECTORY
C
      DO J = 1,LENGTH
            IF (WD(J:J) .EQ. '/') THEN
                 IFOUND(I) = J
                 I = I + 1
            ENDIF
      END DO
C
C FIND THE APPROPRIATE WORKING DIRECTORY VALUE DEPENDING ON LEVEL.
C IF -1 THEN MAKE THE GLOBAL DIRECTORY THE WORKING DIRECTORY
C
      IF (LEVEL .EQ. -1) THEN
            NEWLEN = IFOUND(2) - 1
            NEWWD = WD (1:NEWLEN)
      ELSE
```

```
          I = 1
          DO WHILE (IFOUND(I) .NE. 0)
              I = I + 1
          END DO

          DO TIMES = 1,LEVEL
              NEWLEN = IFOUND(I-1) - 1
              NEWWD = WD(1:NEWLEN)
              I = I - 1
          END DO
      END IF
C
C SET THE WORKING DIRECTORY TO THE PARENT DIRECTORY
C
      ERROR = FMPSETWORKINGDIR(NEWWD)
      IF (ERROR .LT. 0) THEN
           CALL FMPREPORTERROR(ERROR)
           STOP
      ENDIF
      END
```

```
  {
      /lab/solution/lab9/s6.pas
  }
PROGRAM tree (input,output);
TYPE
      int = -32768..32767;
      intarray = array [1..10] of int;
      dirtype = packed array [1..64] of char;
      ptype = array [1..5] of int;

VAR
      level, times, error, length, newlen  : int;
      ifound : intarray;
      i, j, wd, newwd : integer;
      initdir, newdir : dirtype;
      pram : ptype;

FUNCTION strdsc(dirnam : dirtype; startchar,nchars : int): integer; EXTERNAL;

FUNCTION fmpworkingdir(dir : integer) : int;  EXTERNAL;

FUNCTION fmpsetworkingdir(VAR dir : integer) : int;  EXTERNAL;

FUNCTION trimlen(dirstring : integer) : int;  EXTERNAL;

PROCEDURE param $ ALIAS 'Pas.NumericParms' $ (VAR pram : ptype); EXTERNAL;

PROCEDURE fmphierarchname(dir : integer);  EXTERNAL;

PROCEDURE fmpreporterror(VAR error : int);  EXTERNAL;

BEGIN

{  PICK UP PARAMETERS PASSED FROM THE PROGRAM RUNSTRING  }

       param(pram);
       level := pram[3];

{  FIRST DETERMINE WHAT YOUR WORKING DIRECTORY IS  }

       wd := strdsc(initdir,1,64);
       error := fmpworkingdir(wd);
       IF (error < 0) THEN  fmpreporterror(error);

{  PLACE THE WORKING DIRECTORY IN HIERARCHIAL FORM  }

       fmphierarchname(wd);
       length := trimlen(wd);
       i := 1;

{  DETERMINE WHERE THE SLASHES ARE IN THE WORKING DIRECTORY  }
```

```
        FOR j := 1 TO length DO
            BEGIN
                IF (initdir[j] = '/') THEN
                    BEGIN
                        ifound[i] := j;
                        i := i + 1;
                    END;
             END;

{ FIND THE APPROPRIATE WORKING DIRECTORY VALUE DEPENDING ON THE LEVEL.  }
{ IF EQUAL TO -1, THEN MAKE THE GLOBAL DIRECTORY THE WORKING DIRECTORY. }

        IF (level = -1) THEN
            BEGIN
                newlen := ifound[2] - 1;
                FOR j := 1 to newlen DO
                    newdir[j] := initdir[j];
            END
            ELSE IF (level = 0) THEN
                BEGIN
                    newdir := initdir;
                    writeln('There were no changes made to your directory !');
                    writeln;
                END
            ELSE
                BEGIN
                     i := 1;
                     WHILE (ifound[i] <> 0)  DO
                            i := i + 1;
                     FOR times := 1 to level DO
                         BEGIN
                             newlen := ifound[i-1] - 1;
                             i := i - 1;
                         END;
                     FOR j := 1 TO newlen DO
                         newdir[j] := initdir[j];
                END;

{  SET THE WORKING DIRECTORY TO THE PARENT DIRECTORY    }

        newwd := strdsc(newdir,1,64);
        error := fmpsetworkingdir(newwd);
        IF (error < 0) THEN  fmpreporterror(error);
        writeln('Your new working directory is now : ',newdir);

END.
```

```
************************************************************************
*                                                                      *
*      This is the solution set to LAB Chapter 10 example 4.           *
*                                                                      *
************************************************************************
```

I. CDS Segmentation

A1. How would default CDS segmentation create code segments for the following modules ?

Default Segmentation

```
Module 1 (A.rel) = 15 pages
Module 2 (B.rel) = 12 pages
Module 3 (C.rel) =  5 pages
Module 4 (D.rel) = 27 pages
Module 5 (E.rel) =  5 pages
```

The following code segments are created from the above modules :

```
Code Segment 0 (27 pages) = Module 1 + Module 2
Code Segment 1 ( 5 pages) = Module 3
Code Segment 2 (27 pages) = Module 4
Code Segment 3 ( 5 pages) = Module 5
```

A2. How could you reorder the above modules manually to create fewer segments ?

Manual Segmentation

If we reorder and relocate the modules like this:

```
Module 1 (A.rel) = 15 pages
Module 2 (B.rel) = 12 pages
Module 3 (D.rel) = 27 pages
Module 4 (C.rel) =  5 pages
Module 5 (E.rel) =  5 pages
```

then we obtain the following segments:

```
Code Segment 0 (27 pages) = Module 1 + Module 2
Code Segment 1 (27 pages) = Module 3
Code Segment 2 (10 pages) = Module 4 + Module 5
```

```
************************************************************************
************************************************************************
```

B1. How would default CDS segmentation create code segments for the following modules ?

Default Segmentation

```
Module 1 (A.rel) = 15 pages
Module 2 (B.rel) = 17 pages
Module 3 (C.rel) = 15 pages
Module 4 (D.rel) =  7 pages
Module 5 (E.rel) = 10 pages
Module 6 (F.rel) = 12 pages
```

The following code segments are created from the above modules :

```
Code Segment 0 (15 pages) = Module 1
Code Segment 1 (17 pages) = Module 2
Code Segment 2 (22 pages) = Module 3 + Module 4
Code Segment 3 (22 pages) = Module 5 + Module 6
```

B2. How could you reorder these modules manually to create fewer, but larger segments ?

Manual Segmentation

If we reorder and relocate the modules like this:

```
Module 1 (A.rel) = 15 pages
Module 2 (C.rel) = 15 pages
Module 3 (D.rel) =  7 pages
Module 4 (E.rel) = 10 pages
Module 5 (F.rel) = 12 pages
Module 6 (B.rel) = 17 pages
```

then we obtain the following segments:

```
Code Segment 0 (30 pages) = Module 1 + Module 2
Code Segment 1 (29 pages) = Module 3 + Module 4 + Module 5
Code Segment 2 (17 pages) = Module 6
```

```
FTN7X,L
$CDS ON
C      /LAB/SOLUTION/LAB10/S1.FTN
C
       PROGRAM cdsprg
C
       WRITE(1,*)'GOOD MORNING, HUMAN !!!'
       PAUSE
       WRITE(1,*)'HAVE A NICE DAY !'
C
       END
C
C      LOOK AT THE PARTITION - 'WH,PA' - WHILE THE PROGRAM IS SUSPENDED.
C      THE PARTITON SHOULD LOOK SOMETHING LIKE THE FOLLOWING :
C
C  CI> sl101a.run
C  GOOD MORNING, HUMAN !!!

C  SL101 Suspended.

C  CM> wh,pa
C  Ptn#    Page Range   Size   Occupant                 Status           Priority
C  ------------------------------------------------------------------------------
C   1        56-  57      2    free
C            58-  85     28    CI/75      (shared)                         51
C            86- 101     16    free
C           102- 133     32    D.RTR                 saving resources     1
C           134- 149     16    CI/77      (data)                           51
C           150- 152      3    free
C           153- 155      3    QCLM                                        28
C           156- 159      4    EXECW                                       30
C           160- 162      3    PTOPM                                       30
C           163- 166      4    EXECM                                       30
C           167- 182     16    CI/71      (data)                           51
C           183- 194     12    WH/75                                        5
C           195- 211     17    free
C           212- 227     16    CM         (data)                            2
C           228- 251     24    free
C           252- 254      3    GRPM                                         4
C           255- 256      2    QUEUE                 serial reusable        2
C           257- 262      6    IOMAP/75              serial reusable       90
C           263- 273     11    LOGON                                        2
C           274- 281      8    PROMT                 saving resources       3
C           282- 297     16    CI/75      (data)                           51
C           298- 304      7    SL101/75   (data)                           99
C           305- 307      3    SL101/75   (code)                           99
C           308- 483    176    free
C           484- 486      3    UPLIN                 serial reusable        3
C           487- 511     25    free
C  ----------------------------------------------------------------------
C  Thu Apr  7, 1983   7:51 am
```

```
  {
     /lab/solution/lab10/s1.pas
     Look at equivalent FORTRAN solution for wh,pa
     output.
  }

$CDS ON
PROGRAM CDSPRG (input,output);

CONST
      exec7 = 7;

TYPE
      int = -32768..32767;

PROCEDURE suspend $ ALIAS 'EXEC' $  (icode : int);  EXTERNAL;

BEGIN
      writeln('Good Morning !');
      suspend(exec_7);
      writeln('Have a nice day.  Bye, Bye !!);
END.
```

```
FTN7X,L
$CDS ON
C
C      /LAB/SOLUTION/LAB10/S2.FTN
C
       PROGRAM CDSPRG
C
       WRITE(1,*)'GOOD MORNING, HUMAN !!!'
       PAUSE
       WRITE(1,*)'HAVE A NICE DAY !'
C
       END
C
C      LOOK AT THE PARTITION - 'WH,PA' - WHILE THE PROGRAM IS SUSPENDED.
C      THE PARTITON SHOULD LOOK SOMETHING LIKE THIS :
C
C  CM> wh,pa
C  Ptn#   Page Range   Size   Occupant                  Status            Priority
C  ------------------------------------------------------------------------------
C  1       56-  57       2    free
C          58-  85      28    CM          (shared)                           2
C          86- 101      16    free
C         102- 133      32    D.RTR                     saving resources     1
C         134- 149      16    CI/77       (data)                            51
C         150- 152       3    free
C         153- 155       3    QCLM                                          28
C         156- 159       4    EXECW                                         30
C         160- 162       3    PTOPM                                         30
C         163- 166       4    EXECM                                         30
C         167- 182      16    CI/71       (data)                            51
C         183- 191       9    free
C         192- 208      17    DSRTR                     saving resources    30
C         209- 211       3    free
C         212- 227      16    CM          (data)                             2
C         228- 251      24    free
C         252- 254       3    GRPM                                           4
C         255- 256       2    QUEUE                     serial reusable      2
C         257- 262       6    IOMAP/75                  serial reusable     90
C         263- 273      11    LOGON                                          2
C         274- 281       8    PROMT                     saving resources     3
C         282- 297      16    CI/75       (data)                            51
C         298- 327      30    free
C         328- 334       7    SL1.A/75    (data)                            99
C         335- 341       7    SL1.B/75    (data)                            99
C         342- 348       7    SL1.C/75    (data)                            99
C         349- 355       7    SL1.D/75    (data)                            99
C         356- 362       7    SL1.E/75    (data)                            99
C         363- 374      12    WH/75                                          5
C         375- 483     109    free
C         484- 486       3    UPLIN                     serial reusable      3
C         487- 493       7    SL101/75    (data)                            99
```

```
C        494- 496      3    SL1.E/75   (shared)                   99
C        497- 511     15    free
C -----------------------------------------------------------------
C  Thu Apr  7, 1983   9:16 am
ftn7x,l
C
C    /lab/solution/lab10/s3c.ftn
C
      program S3C(5)

      real radius, area

      write (1,'("radius: _")')
      read (1,*) radius
      area = 3.14159 * radius * radius
      if (radius .GT. 0) then
         write (1,'("area =", f4.2)') area
      else
         write (1,'("invalid data")')
      end if
      call segrt
      end
```

```
ftn7x,l
C
C        /lab/solution/lab10/s3m.ftn
C

      program seg

      integer selection, circle(3), square(3), triangle(3)
      data circle/'S3C'/, square/'S3S'/, triangle/'S3T'/

      write (1,'("area program")')
      selection = -1
      do while (selection .NE. 0)
         write(1,'("select one:")')
         write(1,'(
     +        "0 = finished, 1 = circle, 2 = square, 3 = triangle")')
         read (1,*) selection
         if (selection .EQ. 0) write (1,'("finished")')
         if (selection .EQ. 1)
     +      call segld (circle,ierr)
         if (selection .EQ. 2)
     +      call segld (square,ierr)
         if (selection .EQ. 3)
     +      call segld (triangle,ierr)
       end do
       end
```

```
ftn7x,l
C
C    /lab/solution/lab10/s3s.ftn
C
      program S3S(5)

      real side, area

      write (1,'("side: _")')
      read (1,*) side
      area = side * side
      if (side .GT. 0) then
         write (1,'("area =", f4.2)') area
      else
         write (1,'("invalid data")')
      end if
      call segrt
      end
```

```
ftn7x,l
C
C     /lab/solution/lab10/s3t.ftn
C
      program S3T(5)

      real base, height, area

      write (1,'("base: _")')
      read (1,*) base
      write (1,'("height: _")')
      read (1,*) height
      area = 0.5 * base * height
      if ((base .GT. 0) .AND. (height .GT. 0)) then
         write (1,'("area =", f4.2)') area
      else
         write (1,'("invalid data")')
      end if
      call segrt
      end
```

```
{                  /lab/solution/lab10/s3c.pas

  This segment calculates the area of a circle, prints it out, and
  returns to the main program.   The file 'include' contains all the
  variables that are needed by the main program and by its segments.}

$segment$
program S3C ;

$include 'include'$

procedure procl;
   begin
       rewrite(out,'1');
       reset(inp,'1');
       writeln (out,'radius: _');
       read (inp,radius);
       area := 3.14159 * radius * radius;
       if radius > 0
          then writeln (out,'area =', area:4:2)
          else writeln (out,'invalid data')
   end;
   .
```

```
{               /lab/solution/lab10/s3m.pas

  In this solution, the child programs are segments and not separate
  programs.  The parent program schedules the  segment according to
  what the user desires.  The file 'include' contains all the
  variables that are accessed by the main program and its segments. }

program S3M ;

$include 'include'$

procedure segload $alias 'Pas.SegmentLoad'$  (name : progname); external;

procedure proc1;  external;

procedure proc2;  external;

procedure proc3;  external;

begin
   rewrite(out,'1');
   reset(inp,'1');

   circle := 'S3C';
   square := 'S3S';
   triangle := 'S3T';

   writeln (out,'area program');
   repeat
      writeln (out,'select one:');
      writeln (out,'0 = finished, 1 = circle, 2 = square, 3 = triangle');
      read (inp,selection);
      case selection of
         0 : writeln (out,'finished');
         1 : begin
                  segload(circle);
                  proc1;
             end;
         2 : begin
                  segload(square);
                  proc2;
             end;
         3 : begin
                  segload(triangle);
                  proc3;
             end;
      end;
   until selection = 0
end.
```

```
{                    /lab/solution/lab10/s3s.pas

  This segment calculates the area of a square, prints it out, and
  returns to the main program.   The file 'include' contains all the
  variables that are needed by the main program and by its segments. }

$segment$
program S3S;

$include 'include'$

procedure proc2;
    begin
        rewrite(out,'1');
        reset(inp,'1');
        writeln (out,'side: _');
        read (inp,side);
        area := side * side;
        if side > 0
           then writeln (out,'area =', area:4:2)
           else writeln (out,'invalid data')
    end;
  .
```

```
{                    /lab/solution/lab10/s3t.pas

  This segment calculates the area of a triangle, prints it out, and
  returns to the main program.  The file 'include' contains all the
  variables that are needed by the main program and by its segments. }

$segment$
program S3T ;

$include 'include'$

procedure proc3;
    begin
        rewrite(out,'1');
        reset(inp,'1');
        writeln (out,'base: _');
        read (inp,base);
        writeln (out,'height: _');
        read (inp,height);
        area := 0.5 * base * height;
        if (base > 0) and (height > 0)
           then writeln (out,'area =', area:4:2)
           else writeln (out,'invalid data')
    end;
  .
```

```
FTN7X,L
$EMA /BIG/
C
C     /LAB/SOLUTION/LAB11/S1A.FTN
C
      PROGRAM EMAX1

C**************************************************************
C*    LAB 11-1 - This program will find the average of an     *
C*               array of elements in EMA                     *
C*                                                            *
C*    The labeled COMMON statement defines the EMA variables. *
C*    These variables are manipulated just like any other    *
C*    variable.                                               *
C**************************************************************

      COMMON /BIG/ RVAL(16384)
      DIMENSION IDCB(144),RBUF(65)

      SUM  = 0
      ITYPE = FMPOPEN(IDCB,IERR,'/LAB/PROBLEM/RNDFIL','RO',1)
      IENUM = 1
      IF (IERR .LT. 0) GOTO 9000

C**************************************************************
C*    Loop to store values from disc file into EMA            *
C**************************************************************

      DO 20 I=1,256
        IFACTR = (I-1) * 64
        LENGTH = FMPREAD(IDCB,IERR,RBUF,130)
        IENUM = 2
        IF (IERR .LT. 0) GOTO 9000

C**************************************************************
C*    Buffer holds 64 real values(128 WORDS). Assign these    *
C*    values to the appropriate rval elements                 *
C**************************************************************

        DO 10 J=1,64
          K = IFACTR + J
          RVAL(K) = RBUF(J)
10      CONTINUE
20    CONTINUE
      IERR = FMPCLOSE(IDCB,IERR)
      IENUM = 3
      IF (IERR .LT. 0) GOTO 9000

C**************************************************************
C*    Find and print the average of all values in rval        *
C**************************************************************
```

```
      DO 30 I=1,16384
        SUM  = SUM  + RVAL(I)
30    CONTINUE
      AVG = SUM  / 16384.
      WRITE(1,'(/"THE AVERAGE IS : ",F15.5/)') AVG
      STOP

C*****************************************************************
C*    Handle FMP call errors                                     *
C*****************************************************************

9000  WRITE(1,9100) IERR,IENUM
9100  FORMAT(/"ENCOUNTERED ERROR #",I4," IN FMP CALL #",I1)
      END
```

```
FTN7X,L
$EMA /BIG/
C
C     /LAB/SOLUTION/LAB11/S1B.FTN
C
      PROGRAM VMAX1

C**************************************************************
C*    LAB 11-1 - This program will find the average of an     *
C*               array of elements in EMA                     *
C*                                                            *
C*    The labeled COMMON statement defines the EMA variables. *
C*    These variables are manipulated just like any other     *
C*    variable.  The values are stored in VMA if you load the *
C*    the program with the VM LINK command.                   *
C*                                                            *
C**************************************************************

      COMMON /BIG/ RVAL(16384)
      DIMENSION IDCB(144),RBUF(65)

      SUM  = 0
      ITYPE = FMPOPEN(IDCB,IERR,'/LAB/PROBLEM/RNDFIL','RO',1)
      IENUM = 1
      IF (IERR .LT. 0) GOTO 9000

C***************************************************************
C*    Loop to store values from disc file into EMA             *
C***************************************************************

      DO 20 I=1,256
        IFACTR = (I-1) * 64
        LENGTH = FMPREAD(IDCB,IERR,RBUF,130)
        IENUM = 2
        IF (IERR .LT. 0) GOTO 9000

C***************************************************************
C*    Buffer holds 64 real values(128 WORDS). Assign these     *
C*    values to the appropriate rval elements                  *
C***************************************************************

        DO 10 J=1,64
          K = IFACTR + J
          RVAL(K) = RBUF(J)
10      CONTINUE
20    CONTINUE
      IERR = FMPCLOSE(IDCB,IERR)
      IENUM = 3
      IF (IERR .LT. 0) GOTO 9000

C***************************************************************
```

```
C*    Find and print the average of all values in rval          *
C****************************************************************

      DO 30 I=1,16384
        SUM  = SUM  + RVAL(I)
30    CONTINUE
      AVG = SUM  / 16384.
      WRITE(1,'(/"THE AVERAGE IS : ",F15.5/)') AVG
      STOP

C****************************************************************
C*    Handle FMP call errors                                    *
C****************************************************************

9000  WRITE(1,9100) IERR,IENUM
9100  FORMAT(/"ENCOUNTERED ERROR #",I4," IN FMP CALL #",I1)
      END
```

```
FTN7X,L
$EMA /BIG/
C
C      /LAB/SOLUTION/LAB11/S2.FTN
C
       PROGRAM EMA2
C      ************************************************
C      * This program will share the EMA space BIG with *
C      * other programs and calculate the standard      *
C      * deviation of the array RVAL. Link with SH.     *
C      ************************************************
       COMMON /BIG/ RVAL(16384)
C
       SUM = 0.0
       SUMSQ = 0.0
C
C      Find average of array elements
C
       DO 50  I = 1,16384
50        SUM = SUM + RVAL(I)
       AVG = SUM / FLOAT(16384)
C
C      Find the standard deviation
C
       DO 60  I = 1,16384
         DEV = RVAL(I) - AVG
60     SUMSQ = SUMSQ + DEV**2
       STDDEV = SQRT(SUMSQ)
C
       WRITE (1,'(/" THE AVERAGE = ",F10.5)') AVG
       WRITE (1,'(/" THE STANDARD DEVIATION = ",F10.5)') STDDEV
       CALL ULEMA
       END
```

```
FTN7X,L
$EMA /BIG/
C
C     /LAB/SOLUTION/LAB11/S2A.FTN
C
      PROGRAM EMAX2

C*************************************************************
C*    LAB 11-2 - This program will find the average of an   *
C*                array of elements in EMA                   *
C*                                                           *
C*    The labeled COMMON statement defines the EMA variables. *
C*    Link with large SHEMA program. Notice that this program *
C*    uses LKEMA.                                            *
C*************************************************************

      COMMON /BIG/ RVAL(16384)
      DIMENSION IDCB(144),RBUF(65)

      SUM  = 0
      CALL LKEMA
      ITYPE = FMPOPEN(IDCB,IERR,'/LAB/PROBLEM/RNDFIL','RO',1)
      IENUM = 1
      IF (IERR .LT. 0) GOTO 9000

C*************************************************************
C*    Loop to store values from disc file into EMA          *
C*************************************************************

      DO 20 I=1,256
        IFACTR = (I-1) * 64
        LENGTH = FMPREAD(IDCB,IERR,RBUF,130)
        IENUM = 2
        IF (IERR .LT. 0) GOTO 9000

C*************************************************************
C*    Buffer holds 64 real values(128 WORDS). Assign these  *
C*    values to the appropriate rval elements               *
C*************************************************************

        DO 10 J=1,64
          K = IFACTR + J
          RVAL(K) = RBUF(J)
10      CONTINUE
20    CONTINUE
      IERR = FMPCLOSE(IDCB,IERR)
      IENUM = 3
      IF (IERR .LT. 0) GOTO 9000

C*************************************************************
C*    Find and print the average of all values in rval      *
```

```
C*************************************************************

      DO 30 I=1,16384
        SUM  = SUM  + RVAL(I)
30    CONTINUE
      AVG = SUM  / 16384.
      WRITE(1,'(/"THE AVERAGE IS : ",F15.5/)') AVG
      STOP

C*************************************************************
C*    Handle FMP call errors                                 *
C*************************************************************

9000  WRITE(1,9100) IERR,IENUM
9100  FORMAT(/"ENCOUNTERED ERROR #",I4," IN FMP CALL #",I1)
      END
```

```
FTN7X,L
$EMA /BIG/

C
C     /LAB/SOLUTION/LAB11/S3.FTN
C

*********************************************************
*     This program will initialize a VMA/EMA array, and *
*     print out the time the operation took. To increase *
*     the working set, use the WS link command. By       *
*     increasing the working set, less pages will have   *
*     to be swapped to disc and thus the program may     *
*     run faster.                                        *
*********************************************************

      PROGRAM EMAEX3

C     ***********************************
C     *     Define the VMA/EMA array    *
C     ***********************************

      COMMON /BIG/ I(1024,1024)

      TIME=ETIME(-10)
      DO 10 M=1,100
      DO 10 N=1,1024
      I(N,M)=0
   10 CONTINUE
      WRITE(1,33) ETIME(10)
   33 FORMAT("TIME IN SECONDS =",F10.3)
      END
```

```
{
   /lab/solution/lab11/s4.pas
}
  $HEAP 2$

  program vmain (input,output);

  type

     int = -32768 .. 32767;
     num = 1 .. 100;
     matrix = array [num, num] of int;
     ptr = ^matrix;
     double = array [1..2] of int;

  var

     ile,                       { LENGTH OF TRANSFER }
     icode : int;               { REQUEST CODE }
     icnwd : double;            { CONTROL WORD }
     ibuff : ptr;               { BUFFER }

  procedure vmaio
        ( icode : int; icnwd : double; var ibuff: ptr;
          ilen : int );
        external;

  begin

   { CREATE POINTER TO VMA/EMA AREA }

     new ( ibuff );

     icode := 1;          { READ }
     icnwd[1] := 1;       { TO TERMINAL }
     icnwd[2] := 0;
     ile := 1000;         { 1000 WORDS ONLY BECAUSE OF TERMINAL BUFFER }

   { READ IN VMA/EMA DATA REFERENCED BY "IBUFF" }

     vmaio ( icode, icnwd, ibuff, ile );

     icode := 2;          { WRITE }
     ile := 1000;
     icnwd[1] := 1;
     icnwd[2] := 0;

   { WRITE OUT VMA/EMA DATA REFERENCED BY "IBUFF" }

     vmaio ( icode, icnwd, ibuff, ile );
  end.
```

```
FTN7X,L
$EMA/BIG/
C
C       /LAB/SOLUTION/LAB11/S5A.FTN
C
C       THIS PARENT SCHEDULES THE CHILD WITH WAIT AND PRINTS OUT THE
C       VALUES IT RETRIEVES FROM SHAREABLE EMA.  LINK WITH 'SH' COMMAND.
C
        PROGRAM PARENT
C
        COMMON /BIG/INUM(1000)
C
        INTEGER PNAME(3)
        DATA PNAME/6HCHILD /

        CALL EXEC(9,PNAME)
        WRITE(1,*)'THE PRIME NUMBERS BETWEEN 1 AND 1000 ARE : '
        DO 100, I = 1,1000
           IF (INUM(I) .EQ. 0) GOTO 100
           WRITE(1,*)INUM(I)
 100    CONTINUE
C
        END
```

```
ftn7x,l
$ema/big/
C
C       /LAB/SOLUTION/LAB11/S5B.FTN
C
C       THE CHILD PROGRAM CALCULATES THE PRIME NUMBERS BETWEEN 1 AND 1000
C       AND PUTS THE RESULT IN SHAREABLE EMA.  LINK PROGRAM WITH 'SH' COMMAND.
C
        program CHILD
        implicit integer (a-z)
        common /big/iarray(1000)

        do 25 i = 1, 1000
           iarray(i) = i

           do 10 j = 2, 1000
              if (i .eq. j) go to 25
              if (i .lt. j) go to 25
              if (mod (i,j) .eq. 0) then
                  iarray(i) = 0
                  go to 25
              endif
 10        continue
 25     continue

        end
$heap 2$
```

```
{     /LAB/SOLUTION/LAB11/S5A.FTN

  THIS PARENT PROGRAM SCHEDULES THE CHILD WITH WAIT AND PRINTS OUT
  THE VALUES IT RECEIVES FROM SHAREABLE EMA.  LINK WITH 'SC' AND
  'SH,label' COMMANDS.                                              }

PROGRAM PARENT (input,output);

TYPE
      int = -32768..32767;
      big = array [1..1000] of int;
      bigptr = ^big;
      com  =  record
                 biggyptr : bigptr;
              end;
      comptr = ^com;
      name = packed array [1..6] of char;

VAR
      i, sizeblank : int;
      sizeshared, start, heap_stack : integer;
      biggy : bigptr;
      commy : comptr;
      pname : name;

FUNCTION common_blank $ ALIAS 'Pas.BlankCom2' $ : comptr; EXTERNAL;

FUNCTION blank_size $ ALIAS 'Pas.BlankSize' $ : int; EXTERNAL;

FUNCTION sharedsize $ ALIAS 'Pas.AlSharedSize' $ : integer; EXTERNAL;

FUNCTION setshared $ ALIAS 'Pas.AlSetShared' $
         (start, heap_stack : integer) : boolean;  EXTERNAL;

PROCEDURE exec_9 $ALIAS 'EXEC'$ (icode : int; pname : name); EXTERNAL;

BEGIN
     sizeblank := blank_size;
     if sizeblank <> 0 then
        begin
           sizeshared := sharedsize;
           if sizeshared <> 0 then
              begin
                 commy := common_blank;
                 if (setshared(0,1050)) then
                     begin
                          new(biggy);
                          commy^.biggyptr := biggy;
                          pname := 'CHILD';
                          exec_9(9,pname);
```

```
                writeln('The prime #''s between 1 and 1000 are :');

                FOR i := 1 to 1000 DO
                    IF (biggy^[i] <> 0) THEN writeln(biggy^[i]);

            end else writeln('Heap stack setup failure !!');
        end else writeln('No access to SHEMA !!');
    end else writeln('No common available !!');

END.
$heap 2$
```

```
{     /LAB/SOLUTION/LAB11/S5B.PAS

  THIS CHILD PROGRAM CALCULATES THE PRIME NUMBERS BETWEEN 1 AND 1000
  AND STORE THE RESULTS IN SHAREABLE EMA, SO THAT THE PARENT CAN
  RETRIEVE THEM.   LINK WITH 'SC' AND 'SH,label' COMMAND.           }

PROGRAM CHILD;

TYPE
      int = -32768..32767;
      big = array [1..1000] of int;
      bigptr = ^big;
      com = record
                biggyptr : bigptr;
            end;
      comptr = ^com;

VAR
      i,j,sizeblank : int;
      sizeshared, start, heap_stack : integer;
      biggy : bigptr;
      out : text;
      commy : comptr;

FUNCTION common_blank $ ALIAS 'Pas.BlankCom2' $  : comptr;  EXTERNAL;

FUNCTION blank_size $ ALIAS 'Pas.BlankSize' $  : int;  EXTERNAL;

FUNCTION sharedsize $ ALIAS 'Pas.AlSharedSize' $  : integer;  EXTERNAL;

FUNCTION setshared $ ALIAS 'Pas.AlSetShared' $
        (start, heap_stack : integer) : boolean;  EXTERNAL;

BEGIN
      rewrite(out,'l');

      sizeblank := blank_size;
      if sizeblank <> 0  then
         begin
            sizeshared := sharedsize;
            if sizeshared <> 0 then
               begin
                  commy := common_blank;
                  biggy := commy^.biggyptr;
                  if (setshared(1051,2100)) then
                     begin
                        FOR i := 1 to 1000 DO
                            BEGIN
                               j := 2;
                               commy^.biggyptr^[i] := i;
```

```
                WHILE (j < 1000) AND (i > j) AND (i <> j) D
                     BEGIN
                         if ((i mod j) = 0) then
                             begin
                             commy^.biggyptr^[i] := 0;
                                  j := 1000;
                             end;
                         j := j + 1;
                     END;
            END;
      end else writeln(out,'Heap stack setup failure !!');
   end else writeln(out,'No access to SHEMA !!');
end else writeln(out,'No common available !!');

END.
```

```
FTN7X,L
C
C          /lab/solution/lab12/s1.ftn
C
C  This program uses an LU lock to the line printer.   This enables the
C  program to have exclusive access to it; no other program can access it.
C  When the program pauses, type in 'WH' to make sure that the LU is locked.
C
      PROGRAM LULOCK
C
      OPTION = 40001B
      CALL LURQ(OPTION,6,1,KEYWD)
      GO TO 100
C
 15   DO 25, I=1,10
          WRITE(1,*)'I HAVE EXCLUSIVE ACCESS TO THE PRINTER !!'
 25   CONTINUE
C
      PAUSE
C
      OPTION = 0
      CALL LURQ(OPTION,6,1,KEYWD)
      WRITE(1,*)'WE HAVE COMPLETED SUCCESSFULLLY !!'
      WRITE(6,*)'WE HAVE COMPLETED SUCCESSFULLLY !!'
      STOP
C
100   CALL ABREG(IA,IB)
      WRITE(1,*) ('ERROR IS : ',IA,IB)
C
 999  END
```

```
{           /lab/solution/lab12/s1.pas

  This program uses an LU lock to the line printer.   This enables the
  program to have exclusive access to it; no other program can access it.
  When the program pauses, type in 'WH' to make sure that the LU is locked. }

$RECURSIVE OFF$
PROGRAM LULOCK ;
CONST
     exec7 = 7;
     no_abort = 16384;
     no_wait = -32768;

TYPE
     int = -32768..32767;

VAR
     a,b,i,option,lu,numlu,keywd : int;
     out, outlu : text;

PROCEDURE lurq(option,lu,numlu,keywd : int);  EXTERNAL;

PROCEDURE abreg(VAR a,b : int);  EXTERNAL;

PROCEDURE suspend $ ALIAS 'EXEC' $ (icode : int);  EXTERNAL;

PROCEDURE errorcheck;
$DIRECT$
     BEGIN
          abreg (a,b);
          writeln(out,'A is ', A,' B is ', B);
     END;

BEGIN
     rewrite(out,'1');
     rewrite(outlu,'6');

     option := 1 + no_wait + no_abort;
     lu := 6;
     numlu := 1;
     lurq(option,lu,numlu,keywd);
     errorcheck;

     IF (a = 0) THEN
        BEGIN
             FOR I:= 1 TO 10 DO
                 writeln(outlu,'I have complete access to the printer !!');

             suspend(exec7);

             option := 0;
```

```
        lurq(option,lu,numlu,keywd);
        writeln;
        writeln(out,'We have completed successfully !!');
        writeln(outlu,'We have completed successfully !!');
    END ELSE
        writeln(out,'Unable to lock LU 6 !!');

END.
```

```
FTN7X,L
C
C      /LAB/SOLUTION/LAB12/S2A.FTN
C
C This program along with PROG2 will compete for the use of the line printer.
C
       PROGRAM PROG1
       INTEGER PNAME(3)
       DATA PNAME/'PROG2'/
C
       CALL EXEC(10,PNAME)
5      DO 10 I=1,25
            WRITE(6,*)'I''M PROGRAM 1 !!'
10     CONTINUE
C
       END
```

```
FTN7X,L
C
C      /LAB/SOLUTION/LAB12/S2B.FTN
C
C This program along with PROG1 will compete for the use of the line printer.
C
      PROGRAM PROG2
C
      DO 10 I=1,25
          WRITE(6,*)'I''M PROGRAM 2 !!'
10    CONTINUE
C
      END
```

```
FTN7X,L
C
C       /LAB/SOLUTION/LAB12/S2C.FTN
C
C This program competes with PROG4 for the use of the line printer by using
C resource numbers.  This program allocates the resource number, prints out
C five lines and then passes the resource number to PROG4.  PROG4 prints
C out five lines and passes the resource number back to PROG3.  This
C continues until each program has finished printing out it's messages.
C
        PROGRAM PROG3

        IMPLICIT INTEGER (A-Z)
        INTEGER PNAME(3)

        DATA PNAME/'PROG4'/
        DATA ALLOC_GLOBAL/20B/
        DATA DEALLOCATE_RN/40B/
        DATA UNLOCK_RN/4/
        DATA LOCK_RN/1/
        DATA NO_ABORT/40000B/
C
        CALL RNRQ(ALLOC_GLOBAL+NO_ABORT,RN,STAT)
        GO TO 888
C
        CALL EXEC(10,PNAME,RN)
        DO I = 1,5
           CALL RNRQ(LOCK_RN+NO_ABORT,RN,STAT)
           GO TO 888

           DO J = 1,5
              WRITE(6,*)'I''M PROGRAM 1 !!'
           END DO

           WRITE(1,*)'PROGRAM 1 IS UNLOCKING THE RN !!'
           CALL RNRQ(UNLOCK_RN+NO_ABORT,RN,STAT)
           GO TO 888

           CALL EXEC(12,0,2,0,-1)
        END DO
        STOP
C
 888    WRITE(1,*)'RNRQ ABORT ERROR'
        CALL RNRQ(UNLOCK_RN,RN,STAT)
        CALL RNRQ(DEALLOCATE_RN,RN,STAT)

        END
```

```
{              /lab/solution/lab12/s2a.pas

       This program along with PROG2 will compete
       for the use of the line printer .                   }

PROGRAM PROG1 (input,output);

CONST
     exec10 = 10;

TYPE
     int = -32768..32767;
     name = packed array [1..6] of char;

VAR
     pname : name;
     i     : int;

PROCEDURE exec_10 $ ALIAS 'EXEC' $ (icode : int; pname : name);  EXTERNAL;

BEGIN
     pname := 'PROG2';
     exec_10(exec10,pname);
     FOR i := 1 to 25 DO
         writeln('I''m program # 1 !!');
END.
```

```
{              /lab/solution/lab12/s2b.pas

       This program along with PROG1 will compete
       for the use of the line printer .                  }

PROGRAM S122F;

TYPE
     int = -32768..32767;

VAR
     i   : int;
     out : text;

BEGIN
     rewrite(out,'6');

     FOR i := 1 to 25 DO
         writeln(out,'I''m program # 2 !!');
END.
```

```
FTN7X,L
C
C       /LAB/SOLUTION/LAB12/S2D.FTN
C
C This program competes with PROG3 for the use of the line printer by using
C resource numbers.  PROG3 allocates the resource number, prints out five
C lines and then passes the resource number to PROG4.  PROG4 prints out five
C lines and passes the resource number back to PROG3.  This continues until
C each program has finished printing out it's messages.  Since PROG4 is the
C last program to use the resource number, it will deallocate the number.
C
      PROGRAM PROG4

      IMPLICIT INTEGER (A-Z)
      DIMENSION IPARM(5)

      DATA DEALLOCATE_RN/40B/
      DATA UNLOCK_RN/4/
      DATA LOCK_RN/1/
      DATA NO_ABORT/40000B/
C
      CALL RMPAR(IPARM)
      RN = IPARM(1)
C
      WRITE(1,*)'THE RESOURCE NUMBER IS',RN
      DO I = 1,5
         CALL RNRQ(LOCK_RN+NO_ABORT,RN,STAT)
         GO TO 883
C
         DO J = 1,5
            WRITE(6,*)'I''M PROGRAM 2 !!'
         END DO
C
         WRITE(1,*)'PROGRAM 2 IS UNLOCKING THE RN !!'
         CALL RNRQ(UNLOCK_RN+NO_ABORT,RN,STAT)
         GO TO 888
         CALL EXEC(12,0,2,0,-1)
      END DO
C
      CALL RNRQ(DEALLOACATE_RN+NO_ABORT,RN,STAT)
      STOP
C
 888  WRITE(1,*)'RNRQ ABORT ERROR'
      CALL RNRQ(UNLOCK_RN,RN,STAT)
      CALL RNRQ(DEALLOCATE_RN,RN,STAT)
C
      END
```

```
FTN7X,L
C
C      /lab/solution/lab12/s3a.ftn
C
       PROGRAM parent
c
c
c     Daddy is an example of program scheduling using EXEC CALL,
c     of locking LU's using LURQ, and of passing keys from
c     program to program to share lu locks.
c
c
c
c
c          Lock LU 1 with wait.
c
       call lurq(1,1,1,key)
c
c          write message that the LU is locked
c
       write(1,'(" FATHER HAS LOCKED LU 1")')
c
c
c          Now schedule the son, passing the key # to him/her
c
       call exec(9,6HCHILD ,KEY)
c
c          Now signal that the father is executing
c
       write(1,'(" SON HAS COMPLETED, FATHER IS EXECUTING.")')
c
c          Now terminate daddy
c
       call exec(6)
       end
```

```
FTN7X,L
C
C   /lab/solution/lab12/s3b.ftn
C
      PROGRAM CHILD
      integer parms(5)
c
c
c        This program, with program S123A are a demonstration of
c        shareable LU locks.  S123A passes a KEYNUM parm returned
c        by the LURQ call, after locking LU-1.  S123B then writes
c        through that LU lock by using the KEYNUM parm with an exec
c        write.
c
c
c
      call rmpar(parms)
c
c        Now put the key value in "key"
c
      key = parms(1)
c
c
c
      call exec(2,1,29H *****************************,-29,0,0,0,0,key)
      call exec(2,1,29H ****** Son is writing ******,-29,0,0,0,0,key)
      call exec(2,1,29H *** through the LU lock  ***,-29,0,0,0,0,key)
      call exec(2,1,29H *****************************,-29,0,0,0,0,key)
c
c        terminate the son
c
      call exec(6,0)
c
c
c
      end
```

```
  {
     /lab/solution/lab12/s3a.pas
 }
PROGRAM parent;

CONST
     exec6 = 6;
     exec9 = 9;

TYPE
     int = -32768..32767;
     name = packed array [1..6] of char;

VAR
     option,lu,numlu,keywd : int;
     pname : name;
     out   : text;

PROCEDURE lurq(option,lu,numlu,keywd : int);  EXTERNAL;

PROCEDURE exec_6 $ALIAS 'EXEC'$ (icode : int);  EXTERNAL;

PROCEDURE exec_9 $ALIAS 'EXEC'$ (icode : int; pname : name; key : int);
          EXTERNAL;
BEGIN
     rewrite(out,'1');

     option := 1;
     lu := 1;
     numlu := 1;
     lurq(option,lu,numlu,keywd);

     writeln(out,'The father has locked LU 1 !!');

     pname := 'CHILD';
     exec_9(exec9,pname,keywd);

     writeln(out,'Son has completed; Father is executing !!');

     exec_6(exec6);

END.
```

```
{

  /lab/solution/lab12/s3b.pas
 }
PROGRAM CHILD ;

CONST
     exec2 = 2;
     exec6 = 6;

TYPE
     int = -32768..32767;
     btype = packed array [1..29] of char;
     ptype = array [1..5] of int;

VAR
     prams : ptype;
     key   : int;

PROCEDURE params $ALIAS 'Pas.NumericParms'$ (VAR prams : ptype); EXTERNAL;

PROCEDURE exec_2 $ALIAS 'EXEC'$ (icode, lu : int; bufr : btype; bufln,optl,
                 opt2,opt3,opt4,key : int);  EXTERNAL;

PROCEDURE exec_6 $ALIAS 'EXEC'$ (icode : int);  EXTERNAL;

BEGIN
     params(prams);
     key := prams[1];

     exec_2(exec2,1,'*****************************',-29,0,0,0,0,key);
     exec_2(exec2,1,'****** Son is writing *******',-29,0,0,0,0,key);
     exec_2(exec2,1,'**** through the LU lock ****',-29,0,0,0,0,key);
     exec_2(exec2,1,'*****************************',-29,0,0,0,0,key);

     exec_6(exec6);

END.
```

```
ftn7x,l
$ema/big/
C
C    /lab/solution/lab12/s4a.ftn
C
      program parent
      implicit integer (a-z)
      INTEGER PNAME(3)
      common /big/iarray(1000)
      DATA  PNAME/6HCHILD /

      CALL LKEMA

      CNTWD = 21B
      CALL RNRQ(CNTWD,RN,STAT)

      CALL EXEC(10,PNAME,RN)


      do 25 i = 1, 1000
         iarray(i) = i

         do 10 j = 2, 1000
            if (i .eq. j) go to 25
            if (i .lt. j) go to 25
            if (mod (i,j) .eq. 0) then
                iarray(i) = 0
                go to 25
            endif
 10      continue
 25   continue

      CALL RNRQ (4B,RN,STAT)

      END
```

```
FTN7X,L
$EMA/BIG/
C
C         /lab/solution/lab12/s4b.ftn
C
C
      PROGRAM child
      IMPLICIT INTEGER(A-Z)
      INTEGER PARM(5)
      COMMON /BIG/INUM(1000)
C
      CALL RMPAR(PARM)
      RN = PARM(1)

      CNTWD = 1B
      CALL RNRQ (CNTWD,RN,STAT)

C     WRITE(1,*)'THE PRIME NUMBERS BETWEEN 1 AND 1000 ARE : '

      DO 100, I = 1,1000
         IF (INUM(I) .EQ. 0) GOTO 100
         WRITE(1,*)INUM(I)
 100  CONTINUE
C

     CNTWD = 44B
     CALL RNRQ (CNTWD,RN,STAT)

     CALL ULEMA

     END
```

```
{                        /lab/solution/lab12/s4a.pas
   This parent program calculates the prime numbers between 1 and 1000
   and stores them in an array in Shareable EMA.  The parent program
   allocates a resource number, so that the array can be accessed.
   The child program is scheduled with that resource number.        }

$heap 2$
$recursive off$
PROGRAM S4A (input,output);

CONST
      no_abort = 16384;
      alloc_global = 16;
      lock_rn = 1;
      unlock_rn = 4;
TYPE
      int = -32768..32767;
      big = array [1..1000] of int;
      bigptr = ^big;
      com  =  record
                 biggyptr : bigptr;
                 resource : int;
              end;
      comptr = ^com;
      name = packed array [1..6] of char;
      rtype = packed array [1..12] of char;
VAR
      i, j, a, b, cntwd, rn, stat, sizeblank : int;
      sizeshared, start, heap_stack : integer;
      biggy : bigptr;
      commy : comptr;
      pname : name;
      runstr : rtype;

FUNCTION common_blank $ ALIAS 'Pas.BlankCom2' $  : comptr;  EXTERNAL;
FUNCTION blank_size $ ALIAS 'Pas.BlankSize' $  : int;  EXTERNAL;
FUNCTION sharedsize $ ALIAS 'Pas.AlSharedSize' $  : integer;  EXTERNAL;
FUNCTION setshared $ ALIAS 'Pas.AlSetShared' $
       (start, heap_stack : integer) : boolean;  EXTERNAL;
PROCEDURE exec_10 $ALIAS 'EXEC'$ (icode : int; pname:name; v,x,y,z,w : int;
        runst : rtype; len : int); external;
PROCEDURE abreg (VAR a,b : int);  EXTERNAL;
PROCEDURE lkema;  EXTERNAL;
PROCEDURE rnrq (cntwd, rn, stat : int);  EXTERNAL;
PROCEDURE errorcheck;
$direct$
   BEGIN
      abreg(a,b);
      writeln('A = ',a:5,' B = ',b:5);
   END;
```

```
BEGIN
     lkema;
     cntwd := alloc_global + lock_rn + no_abort;
     rnrq(cntwd,rn,stat);
     errorcheck;
     writeln('The parent  has the resource number !');

     runstr := 'ru,S4B,1,1';
     pname := 'S4B';
     exec_10(10,pname,0,0,0,0,0,runstr,-10);

     sizeblank := blank_size;
     if sizeblank <> 0 then
        begin
            sizeshared := sharedsize;
            if sizeshared <> 0 then
               begin
                   commy := common_blank;
                   if (setshared(0,1050)) then
                      begin
                          new(biggy);
                          commy^.biggyptr := biggy;
                          commy^.resource := rn;

                          FOR i := 1 to 1000 DO
                              BEGIN
                                 j := 2;
                                 commy^.biggyptr^[i] := i;

                                 WHILE (j < 1000) AND (i > j) AND (i <> j) DO
                                       BEGIN
                                           if ((i mod j) = 0) then
                                                begin
                                                    commy^.biggyptr^[i] := 0;
                                                    j := 1000;
                                                end;
                                           j := j + 1;
                                       END;
                              END;
                      end else writeln('Heap stack setup failure !!');
               end else writeln('No access to SHEMA !!');
        end else writeln('No common available !!');
    writeln('The parent is unlocking the resource number !');
    cntwd := unlock_rn + no_abort;
    rnrq(cntwd, rn, stat);
    errorcheck;
    writeln('The parent is finished !');
END.
```

```
{                   /lab/solution/lab12/s4b.pas

  This child program receives a resource number from the parent.
  With this number, the program is able to access the array and
  print out the prime numbers 1 and 1000.                       }


$heap 2$
$recursive off$
PROGRAM S4B(input,output);

CONST
      lock_rn = 1;
      unlock_rn = 4;
      no_abort = 16384;
      deallocate_rn = 32;

TYPE
      int = -32768..32767;
      big = array [1..1000] of int;
      bigptr = ^big;
      com = record
                biggyptr : bigptr;
                resource : int;
            end;
      comptr = ^com;

VAR
      i, j, a, b, cntwd, rn, stat, sizeblank : int;
      sizeshared, start, heap_stack : integer;
      biggy : bigptr;
      commy : comptr;

FUNCTION common_blank $ ALIAS 'Pas.BlankCom2' $  : comptr;  EXTERNAL;

FUNCTION blank_size $ ALIAS 'Pas.BlankSize' $  : int;  EXTERNAL;

FUNCTION sharedsize $ ALIAS 'Pas.AlSharedSize' $  : integer;  EXTERNAL;

FUNCTION setshared $ ALIAS 'Pas.AlSetShared' $
        (start, heap_stack : integer) : boolean;  EXTERNAL;

PROCEDURE ulema;  EXTERNAL;

PROCEDURE abreg (VAR a,b : int);  EXTERNAL;

PROCEDURE rnrq (cntwd, rn, stat : int);  EXTERNAL;

PROCEDURE errorcheck;
$direct$
   BEGIN
```

```
      abreg(a,b);
      writeln('A = ',a:5,' B = ',b:5);
   END;

BEGIN
     sizeblank := blank_size;
     if sizeblank <> 0  then
        begin
            commy := common_blank;
            rn := commy^.resource;
            cntwd := lock_rn + no_abort;
            rnrq(cntwd,rn,stat);
            errorcheck;
            writeln('The child has locked the resource number !');

            biggy := commy^.biggyptr;
            sizeshared := sharedsize;
            if sizeshared <> 0 then
               begin
                   if (setshared(1051,2100)) then
                      begin
                         FOR i := 1 to 1000 DO
                            IF (biggy^[i] <> 0)
                               THEN writeln(biggy^[i]);
                      end else writeln('Heap stack setup failure !!');
               end else writeln('No access to SHEMA !!');
        end else writeln('No common available !!');

   cntwd := unlock_rn + deallocate_rn + no_abort;
   rnrq(cntwd,rn,stat);
   errorcheck;
   writeln('The child has unlocked the resource number and is now exiting !');

   ulema;

END.
```

```
FTN7X,L
C
C    /lab/solution/lab12/s5.ftn
C    This program will not accept LUs greater than 63.
C    Use XLUEX to do so.
C
      PROGRAM updown
      IMPLICIT INTEGER (A-Z)

      WRITE(1,*)'PLEASE ENTER AN LU NUMBER ?'
      READ(1,*)LU

      CALL EXEC(13,LU,STAT1)

      IF ( BTEST(STAT1,14) ) THEN
          WRITE (1,*)'THE LU IS DOWN !!'
      ELSE
          WRITE (1,*)'THE LU IS UP !!'
      ENDIF

      END
```

```
  {
     /lab/solution/lab12/s5.pas
     This program will not accept LUs greater than 63. (use XLUEX)
  }
PROGRAM updown(input,output);

TYPE
     int = -32768..32767;

VAR
     lu, status : int;

PROCEDURE exec_13 $ ALIAS 'EXEC' $  (icode,lu,status : int); EXTERNAL;

BEGIN
     writeln('Please enter an LU number ?');
     read(lu);

     exec_13(13,lu,status);
     writeln('Status is ',status:3);

     IF (status > 16384) AND (status < 32767) THEN writeln('The LU is down!!')
         ELSE IF (status >= -16384) AND (status <= -1)
                  THEN writeln('The LU is down !!')
         ELSE writeln('The LU is up !!');

END.
```

1. 7912 configuration:

	Disc LU 16	17	18	19	20	21
DP1 HP-IB addr.	0	0	0	0	0	0
DP2 Unit/Vol #	0	0	0	0	0	0
DP3 ms \ start	0	0	0	0	0	0
DP4 > blk	0	0	0	0	1	2
DP5 ls / number	0	19200	38400	57600	58160	58720
DP6 Tracks	400	400	400	1377	1377	1379
DP7 Blocks/track	48	48	48	48	48	48
DP8 reserved	0	0	0	0	0	0

	CTD LU 24
DP1 HP-IB addr	0
DP2 CTD U/V #	400B
DP3 Cache U/V #	100000B
DP4 \ Start blk > of disc	3
DP5 / cache	59376
DP6 reserved	0
DP7 reserved	0
DP8 reserved	0

The disc allocation unit for the CI volumes is 1 block.

2. 7933 configuration:

	LU 16	17	18	19	20	21	22
DP1 HP-IB addr.	0	0	0	0	0	0	0
DP2 Unit/Vol #	0	0	0	0	0	0	0
DP3 ms \ start	0	0	0	0	0	0	0
DP4 > blk	0	0	4	8	12	16	20
DP5 ls / number	0	19200	17172	15144	13116	11088	6660
DP6 Tracks	400	5419	5419	5419	5419	5419	5369
DP7 Blocks/track	48	48	48	48	48	48	48
DP8 reserved	0	0	0	0	0	0	0

3. The mythical flexible disc configuration:

	LU 30	31
DP1 HP-IB addr.	0	0
DP2 Unit number	0	1
DP3 Start head	0	0
DP4 Start Cylinder	0	0
DP5 spares	0	0
DP6 Tracks	112	112
DP7 Blocks/track	16	16
DP8 Surfaces	2	2

4. In order to boot off of the 7925, it must be configured in cylinder mode. 7925 configuration:

	LU			
	10	11	12	13
DP1 HP-IB addr.	0	0	0	0
DP2 Unit number	0	0	0	0
DP3 Start head	0	0	0	0
DP4 Start Cylinder	0	206	412	618
DP5 spares	37	37	37	33
DP6 Tracks	1817	1817	1817	1808
DP7 Blocks/track	64	64	64	64
DP8 Surfaces	9	9	9	9

5. 7920 configuration:

	LU				
	10	11	12	13	14
DP1 HP-IB addr.	0	0	0	0	0
DP2 Unit number	0	0	0	0	0
DP3 Start head	0	0	0	0	0
DP4 Start Cylinder	0	0	0	0	0
DP5 spares	16	16	16	16	16
DP6 Tracks	807	807	807	807	807
DP7 Blocks/track	48	48	48	48	48
DP8 Surfaces	1	1	1	1	1

Each LU is 38736 blocks. The only advantage to surface mode is that it is easier to configure than cylinder mode.

1. The default DVT parameters for %DDC12 are:

Entry point	DDC12
Driver parameter area	7
Driver extension area	98
Device type	12B
timeout	500 (5 sec)

The default IFT parameters for %ID.00 are:

Entry point	ID.00
IFT extension area	33

The default DVT parameters for %DD.00 for a 2621 terminal are:

Device type	0
Driver parameter area	12
Driver parameter 1	1
Driver parameter 2	0
Driver parameter 3	10400B
Driver parameter 4	0
Driver parameter 5	FM
Driver parameter 6	GR
Driver parameter 7	20400B
Driver parameter 8	0
Driver parameter 9	CO
Driver parameter 10	MN
Driver parameter 11	D
Driver parameter 12	0

The driver parameters will enable FMGR as the primary program and COMND as the secondary program. (These programs were the equivalents of CI and CM from the previous revision of RTE-A. Typically, they would be overridden in the generation answer file).

2. See file /LAB/SOLUTION/LAB15/S2.ANS

3. See file /LAB/SOLUTION/LAB15/S3.ANS

4. See file /LAB/SOLUTION/LAB15/S4.ANS

```
* /lab/solution/lab15/s2.ans
* RTE-A.2  PRIMARY SYSTEM GENERATION ANSWER FILE
* D.K.G    REV. 2326     830624.1613
* L.E.N    Modified for generation lab for RTE-A course 830809
*
* Solution for Lab 15 question 2, file = S2.ANS
* Modification of answer file ANS1
*
* Look for *!!* to denote additions or deletions
*
*
* %RPL60 CONTAINS THE A-600 RPL'S WITH NO CDS AND NO DOUBLE PRECISION
* FLOATING POINT.
* %RPL61 CONTAINS THE A-600 RPL'S WITH NO CDS AND DOUBLE PRECISION
* FLOATING POINT.
* %RPL62 CONTAINS THE A-600 PRL'S WITH CDS AND NO DOUBLE PRECISION
* FLOATING POINT.
* %RPL63 CONTAINS THE A-600 RPL'S WITH CDS AND DOUBLE PRECISION
* FLOATING POINT.
*
* %RPL70 CONTAINS THE A-700 RPL'S WITH NO CDS AND NO HARDWARE FLOATING POINT??
* %RPL71 CONTAINS THE A-700 RPL'S WITH NO CDS AND HARDWARE FLOATING POINT.
* %RPL72 CONTAINS THE A-700 RPL'S WITH CDS AND NO HARDWARE FLOATING POINT.
* %RPL73 CONTAINS THE A-700 RPL'S WITH CDS AND HARDWARE FLOATING POINT.
*
* %RPL90 CONTAINS THE A-900 RPL'S WITH NO CDS.
* %RPL91 CONTAINS THE A-900 RPL'S WITH CDS.
*
* THE CARTRIDGE REFERENCE NUMBERS "MS" AND "A2" HAVE BEEN
* USED IN THIS ANSWER FILE FOR GENERATION PURPOSES IN THE
* SOFTWARE PRODUCTION ENGINEERING DEPT.  THEY MUST BE CHANGED
* TO MATCH YOUR CARTRIDGE REFERENCE NUMBERS FOR REGENERATION.
*
*
LINKS,CP
RE,%VCTR::17
* RE,%SPOOL::A2
RE,%EXEC::17
RE,%MEMRY::17
* RE,%CDSFH::A2
RE,%RPL60::17
* RE,%RPL61::17
* RE,%RPL62::A2
* RE,%RPL63::A2
* RE,%RPL70::17
* RE,%RPL71::17
* RE,%RPL72::A2
* RE,%RPL73::A2
* RE,%RPL90::17
* RE,%RPL91::A2
RE,%SAM::17
```

```
RE,%TIME::17
RE,%SCHED::17
RE,%STRNG::17
RE,%LOCK::17
RE,%ERLOG::17
RE,%OPMSG::17
RE,%XCMND::17
RE,%SYCOM::17
RE,%STAT::17
RE,%LOAD::17
RE,%RTIOA::17
RE,%IOMOD::17
RE,%PERR::17
RE,%CLASS::17
RE,%ID.43::17
* RE,%#SPLU::A2
MS,$SYSA.LIB::LIBRARIES
SE,$SYSLB.LIB::LIBRARIES
END
*
RE,%DD.33::17
RE,%ID.52::17
END
*
* RE,%ID.66::A2
RE,%ID.00::17
END
*
RE,%ID.37::17
RE,%DD.30::17
END
*
RE,%IDM00::17
RE,%DD.23::17
END
*
RE,%DD.00::17
ALIGN
RE,%ID.27::17
RE,%ID.50::17
END
*
* RE,%ADV00::A2
RE,%DD.20::17
END
*
*!!* Relocate driver for the 2608S lineprinter
*
RE,%DDC12::17
END
*
```

```
*!!* Relocate driver for 2631 line printer
*
RE,%DD.12::17
END
*
*
*    end driver partition
END
*
*  BEGIN TABLE GENERATION
*  CONFIGURE LU TABLES
*
* ASIC FOR 2621A/P SYSTEM CONSOLE WITH VCP
*
IFT,%ID.00::17,SC:20B
*
DVT,%DD.00::17,M26XX,LU:1,QU:FI,-
   DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.20::17,M264X:1,LU:64
DVT,%DD.20::17,M264X:2,LU:65
*
*!!* remove 2635A - remove the node list for these LUs also
*
*
* ASIC FOR 2635A AUXILIARY CONSOLE/PRINTER
*
*!!* IFT,%ID.00::17,SC:33B
*!!* DVT,%DD.00::17,M2635:0,LU:66,QU:FI,-
*!!*    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
*!!* DVT,%DD.00::17,M2635:1,LU:67
*
*!!* add a 262x terminal - this is just like LU 1 except no CTUs
*
*!!* create an interface table, %ID.00 is the default file, select code 33B
*
IFT,%ID.00::17,SC:33B
*
*!!* create a device table, %DD.00 is the default file, model number 26XX,
*!!* LU 66, queuing is FIFO, driver parameters set up CI as the primary
*!!* program and CM as the secondary program
*
DVT,%DD.00::17,M26XX,LU:66,QU:FI,-
   DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
*
*
* ASIC FOR 2645A AUXILIARY CONSOLE WITH DUAL MINI-CARTRIDGE
*
IFT,%ID.00::17,SC:22B
DVT,%DD.00::17,M26XX,LU:68,QU:FI,-
   DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.20::17,M264X:1,LU:69
```

```
DVT,%DD.20::17,M264X:2,LU:70
*
* PARALLEL INTERFACE CARD
*
IFT,%ID.50::17,SC:26B
DVT,,,LU:84,TO:5000,DX:2,DP:1:1:2,DT:45B
*
* PARALLEL INTERFACE FOR CPU TO CPU COMMUNICATION
*
IFT,%ID.52::17,SC:21B
*
DVT,,,LU:83,TO:0,DT:7,DX:0
*
* HP-IB #1
*
IFT,%ID.37::17,SC:27B
*
*HP-IB #1 BUS CONTROLLER LU
*
DVT,,,LU:9,TO:2000,DT:77B,TX:0,DX:1,DP:1:36B,PR:0
*
* 7908/40/11/41/12/14/33 DISC WITH COMPATIBLE CARTRIDGE TAPE
* LU 16-20,22,23,29-31,34,35,24  HP-IB ADDRESS 0
*
DVT,%DD.33::17,M7908_LF:0,LU:16,DP:1:0
DVT,%DD.33::17,M7908_LF:1,LU:17,DP:1:0
DVT,%DD.33::17,M7908_LF:2,LU:18,DP:1:0
DVT,%DD.33::17,M7908_LF:3,LU:19,DP:1:0
DVT,%DD.33::17,M7908_LF:4,LU:20,DP:1:0
DVT,%DD.33::17,M7911_LF:5,LU:22,DP:1:0
DVT,%DD.33::17,M7912_LF:6,LU:23,DP:1:0
DVT,%DD.33::17,M7912_LF:7,LU:29,DP:1:0
DVT,%DD.33::17,M7914_LF:8,LU:30,DP:1:0
DVT,%DD.33::17,M7914_LF:9,LU:31,DP:1:0
DVT,%DD.33::17,M7933_LF:10,LU:34,DP:1:0
DVT,%DD.33::17,M7933_LF:11,LU:35,DP:1:0
*
* COMPATIBLE CARTRIDGE TAPE CACHE  LU 24  HP-IB ADDRESS 0
*
DVT,%DD.33::17,MTAPE,LU:24,DP:1:0
*
* 7906H HARD DISC   LU 12-15  HP-IB ADDRESS 1
*
DVT,%DD.30::17,M7906:0,LU:12,DP:1:1
DVT,%DD.30::17,M7906:1,LU:13,DP:1:1
DVT,%DD.30::17,M7906:2,LU:14,DP:1:1
DVT,%DD.30::17,M7906:3,LU:15,DP:1:1
*
*3.5" OR  5.25" FLEXIBLE DISC  LU 32,33  HP-IB ADDRESS 2
*
DVT,%DD.30::17,M7902,LU:32,DP:1:2:0:0:0,DP:5:2:66:16:2,TO:3000
```

```
DVT,%DD.30::17,M7902:0,LU:33,DP:1:2:1:0:0,DP:5:2:66:16:2,TO:3000
*
* 7910H FIXED DISC   LU 40-43    HP-IB ADDRESS 3
*
DVT,%DD.30::17,M7910:0,LU:40,DP:1:3
DVT,%DD.30::17,M7910:1,LU:41,DP:1:3
DVT,%DD.30::17,M7910:2,LU:42,DP:1:3
DVT,%DD.30::17,M7910:3,LU:43,DP:1:3
*
* HP-IB TAPE DRIVE    LU 8  HP-IB ADDRESS 4
*
DVT,%DD.23::17,M7970E:0,LU:8,DP:1:4,PR:1
*
* 8" FLEXIBLE DISC  LU 10,11  HP-IB ADDRESS 5
*
DVT,%DD.30::17,M7902:0,LU:10,DP:1:5,to:3000,TO:3000
DVT,%DD.30::17,M7902:1,LU:11,DP:1:5,to:3000,TO:3000
*
* 5.25" FIXED DISC (9134A/B SINGLE VOL.)  LU 52,53,54  HP-IB ADDRESS 6
*
DVT,%DD.30::17,M9134L:0,LU:52,DP:1:6
DVT,%DD.30::17,M9134L:1,LU:53,DP:1:6
DVT,%DD.30::17,M9134L:2,LU:54,DP:1:6
*
* 5.25" FIXED DISC (91341 FOUR VOL.)  LU 48-51   HP-IB ADDRESS 7
*
DVT,%DD.30::17,M9134:0,LU:48,DP:1:7
DVT,%DD.30::17,M9134:1,LU:49,DP:1:7
DVT,%DD.30::17,M9134:2,LU:50,DP:1:7
DVT,%DD.30::17,M9134:3,LU:51,DP:1:7
*
* 248x INTEGRATED DISC INTERFACE
*
IFT,%ID.27::17,SC:32B
*
* Hard disc
*
DVT,%GEN27::17,M2480:0,LU:36
DVT,%GEN27::17,M2480:1,LU:37
DVT,%GEN27::17,M2480:2,LU:38
*
* Microfloppy
*
DVT,%GEN27::17,M2480:3,LU:39
*
*
* HP-IB #2:  INSTRUMENT BUS
*
IFT,%ID.37::17,SC:25B
*
* HP-IB #2 CONTROLLER
```

```
*
DVT,,,LU:21,TO:50,DT:77B,DX:1,DP:1:36B
*
* FOUR DEVICES
*
DVT,,,LU:25,TO:500,DT:77B,DX:1,DP:1:1
DVT,,,LU:26,TO:500,DT:77B,DX:1,DP:1:5
DVT,,,LU:27,TO:500,DT:77B,DX:1,DP:1:6
DVT,,,LU:28,TO:500,DT:77B,DX:1,DP:1:7
*
*!!* add an HP-IB with a 2608s and 2631
*!!*
*
*!!* HP-IB #3   PRINTERS
*
*!!* create an IFT for the HP-IB card
*
IFT,%ID.37::17,SC:30B
*
*!!* 2608S LINE PRINTER LU 85 HPIB ADDR 2
*!!* the first driver parameter is the HP-IB address
*
DVT,%DDC12::17,,LU:85,DP:1:2
*
*!!* HPIB LINE PRINTER LU 6 HPIB ADDR 6
*!!* the first driver parameter is the HPIB address
*
DVT,%DD.12::17,,LU:6,DT:12B,DP:1:6
*
* D.S. LINKS, TWO LUS FOR D.S., TWO FOR LU MAPPING
*
* NETWORK LINKS
*
* IFT,%ID.66::A2,EID.66,SC:24B,QU:FI,TX:18
*
* DVT,,,LU:79,DT:66B
* DVT,,,LU:80,DT:66B
*
* LU MAPPING
*
* IFT,%ADV00::A2,EIDV00,SC:31B,QU:FI,TX:2
*
* DVT,,,LU:81,EDDV00,TX:0
*
* DVT,,,LU:82,EDDV00,TX:5
*
* eight MUX LU'S  SELECT CODE 23, LU 71-78
*
IFT,%IDM00::17,SC:23B
*
DVT,%DD.00::17,M26XX,LU:71,QU:FI,DP:1:20004B,-
```

```
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:72,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:73,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:74,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:75,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:76,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:77,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:78,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
END
*
END
*
* DEFINE NODE LISTS
*
* SYSTEM CONSOLE AND TWO TAPE DRIVES
NODE,1,64,65
*
*!!* this node was for the 2635 which was removed
* AUXILIARY CONSOLE/PRINTER
*!!* NODE,66,67
*
* AUXILIARY CONSOLE WITH TWO TAPE DRIVES
NODE,68,69,70
*
* TWO 8" FLEXIBLE DISCS
NODE,10,11
*
* FOUR 7906 LU'S
NODE,12,13,14,15
*
* FOUR 7910 LU'S
NODE,40,41,42,43
*
* THIRTEEN 7908/11/12/14 LU'S AND A COMPATIBLE CARTRIDGE TAPE DRIVE
NODE,16,17,18,19,20,22,23,24,29,30,31,34,35
*
* TWO 3.5" OR  5.25" FLEXIBLE DISCS
*
NODE,32,33
*
* FOUR 5.25" FIXED DISC LU'S (9134 FOUR VOL.)
*
NODE,48,49,50,51
*
```

```
* THREE 5.25" FIXED DISC LU'S (9134A/B SINGLE VOL.)
*
NODE,52,53,54
*
* FOUR 248x INTEGRATED DISC LU'S
*
NODE,36,37,38,39
*
END,,,,NODE LIST
*
END,,,,INTERRUPT TABLE
*
CLAS,40
RESN,20
ID,40
RS,0
SAM,6144
SL,0,0
BG,30
QU,300,50
SP,0
MB,500
US,0
*
* SYSTEM COMMON
*
* $ABLIB CONTAINS THE BASIC TRAP TABLES  (DELETED)
*
* RE,$ABLIB::17
*
* DS/1000 HAS BEEN GEN'ED INTO THIS SYSTEM FOR VERIFICATION
* OF HARDWARE ONLY. IF PROGRAM DEVELOPMENT CAPABILITY FOR
* DS IS DESIRED REPLACE $FNDLB WITH $FDSLB. $BIGDS IS A
* MERGED LIBRARY CONTAINING ENTRY POINTS FROM SEVERAL DS
* LIBRARIES
*
* DS/1000 LABELED COMMON AREA
*
* RE,%RESA::A2
* RE,$BIGDS::A2,#NRVS
* RE,$BIGDS::A2,#RQUA
* RE,$BIGDS::A2,#LEVL
* RE,$BIGDS::A2,D$EQT
* MS,$BIGDS::A2
*
END,,,,LABELED SYSTEM COMMON RELOCATION
COM,10
*
RE,%MSGS::17
END
*
```

```
LIB,$FNDLB.LIB::LIBRARIES
LIB,$BIGLB.LIB::LIBRARIES
*
*!!* add the pascal library as a default library
*
LIB,$PLIB.LIB::LIBRARIES
*
*
END
```

```
* /lab/solution/lab15/s3.ans
* RTE-A.2  PRIMARY SYSTEM GENERATION ANSWER FILE
* D.K.G    REV. 2326     830624.1613
* L.E.N    Modified for generation lab for RTE-A course 830809
*
* Solution for Lab 15 question 3, file = S3.ANS
* Original answer file = ANS1
* w/ VC+
*
* Look for *!!* to denote additions or deletions
*
*
* %RPL60 CONTAINS THE A-600 RPL'S WITH NO CDS AND NO DOUBLE PRECISION
* FLOATING POINT.
* %RPL61 CONTAINS THE A-600 RPL'S WITH NO CDS AND DOUBLE PRECISION
* FLOATING POINT.
* %RPL62 CONTAINS THE A-600 PRL'S WITH CDS AND NO DOUBLE PRECISION
* FLOATING POINT.
* %RPL63 CONTAINS THE A-600 RPL'S WITH CDS AND DOUBLE PRECISION
* FLOATING POINT.
*
* %RPL70 CONTAINS THE A-700 RPL'S WITH NO CDS AND NO HARDWARE FLOATING POINT??
* %RPL71 CONTAINS THE A-700 RPL'S WITH NO CDS AND HARDWARE FLOATING POINT.
* %RPL72 CONTAINS THE A-700 RPL'S WITH CDS AND NO HARDWARE FLOATING POINT.
* %RPL73 CONTAINS THE A-700 RPL'S WITH CDS AND HARDWARE FLOATING POINT.
*
* %RPL90 CONTAINS THE A-900 RPL'S WITH NO CDS.
* %RPL91 CONTAINS THE A-900 RPL'S WITH CDS.
*
* THE CARTRIDGE REFERENCE NUMBERS "MS" AND "A2" HAVE BEEN
* USED IN THIS ANSWER FILE FOR GENERATION PURPOSES IN THE
* SOFTWARE PRODUCTION ENGINEERING DEPT.  THEY MUST BE CHANGED
* TO MATCH YOUR CARTRIDGE REFERENCE NUMBERS FOR REGENERATION.
*
*
LINKS,CP
RE,%VCTR::17
*!!* include spooling module
RE,%SPOOL::A2
RE,%EXEC::17
RE,%MEMRY::17
*!!* include CDS fault handler
RE,%CDSFH::A2
*!!* use the CDS RPLs
* RE,%RPL60::17
* RE,%RPL61::17
RE,%RPL62::A2
* RE,%RPL63::A2
* RE,%RPL70::17
* RE,%RPL71::17
* RE,%RPL72::A2
```

```
* RE,%RPL73::A2
* RE,%RPL90::17
* RE,%RPL91::A2
RE,%SAM::17
RE,%TIME::17
RE,%SCHED::17
RE,%STRNG::17
RE,%LOCK::17
RE,%ERLOG::17
RE,%OPMSG::17
RE,%XCMND::17
RE,%SYCOM::17
RE,%STAT::17
RE,%LOAD::17
RE,%RTIOA::17
RE,%IOMOD::17
RE,%PERR::17
RE,%CLASS::17
RE,%ID.43::17
* RE,%#SPLU::A2
MS,$SYSA.LIB::LIBRARIES
SE,$SYSLB.LIB::LIBRARIES
END
*
RE,%DD.33::17
RE,%ID.52::17
END
*
* RE,%ID.66::A2
RE,%ID.00::17
END
*
RE,%ID.37::17
RE,%DD.30::17
END
*
RE,%IDM00::17
RE,%DD.23::17
END
*
RE,%DD.00::17
ALIGN
RE,%ID.27::17
RE,%ID.50::17
END
*
* RE,%ADV00::A2
RE,%DD.20::17
END
*
* Relocate driver for the 2608S lineprinter
```

```
*
RE,%DDC12::17
END
*
* Relocate driver for 2631 line printer
*
RE,%DD.12::17
END
*
*
*    end driver partition
END
*
*  BEGIN TABLE GENERATION
*  CONFIGURE LU TABLES
*
* ASIC FOR 2621A/P SYSTEM CONSOLE WITH VCP
*
IFT,%ID.00::17,SC:20B
*
DVT,%DD.00::17,M26XX,LU:1,QU:FI,-
   DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.20::17,M264X:1,LU:64
DVT,%DD.20::17,M264X:2,LU:65
*
* remove 2635A - remove the node list for these LUs also
*
*
* ASIC FOR 2635A AUXILIARY CONSOLE/PRINTER
*
* IFT,%ID.00::17,SC:33B
* DVT,%DD.00::17,M2635:0,LU:66,QU:FI,-
*    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
* DVT,%DD.00::17,M2635:1,LU:67
*
* add a 262x terminal - this is just like LU 1 except no CTUs
*
* create an interface table, %ID.00 is the default file, select code 33B
*
IFT,%ID.00::17,SC:33B
*
* create a device table, %DD.00 is the default file, model number 26XX,
* LU 66, queuing is FIFO, driver parameters set up CI as the primary
* program and CM as the secondary program
*
DVT,%DD.00::17,M26XX,LU:66,QU:FI,-
   DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
*
*
* ASIC FOR 2645A AUXILIARY CONSOLE WITH DUAL MINI-CARTRIDGE
*
```

```
IFT,%ID.00::17,SC:22B
DVT,%DD.00::17,M26XX,LU:68,QU:FI,-
   DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.20::17,M264X:1,LU:69
DVT,%DD.20::17,M264X:2,LU:70
*
* PARALLEL INTERFACE CARD
*
IFT,%ID.50::17,SC:26B
DVT,,,LU:84,TO:5000,DX:2,DP:1:1:2,DT:45B
*
* PARALLEL INTERFACE FOR CPU TO CPU COMMUNICATION
*
IFT,%ID.52::17,SC:21B
*
DVT,,,LU:83,TO:0,DT:7,DX:0
*
* HP-IB #1
*
IFT,%ID.37::17,SC:27B
*
*HP-IB #1 BUS CONTROLLER LU
*
DVT,,,LU:9,TO:2000,DT:77B,TX:0,DX:1,DP:1:36B,PR:0
*
*
* 7908/40/11/41/12/14/33 DISC WITH COMPATIBLE CARTRIDGE TAPE
* LU 16-20,22,23,29-31,34,35,24  HP-IB ADDRESS 0
*
*
DVT,%DD.33::17,M7908_LF:0,LU:16,DP:1:0
DVT,%DD.33::17,M7908_LF:1,LU:17,DP:1:0
DVT,%DD.33::17,M7908_LF:2,LU:18,DP:1:0
DVT,%DD.33::17,M7908_LF:3,LU:19,DP:1:0
DVT,%DD.33::17,M7908_LF:4,LU:20,DP:1:0
DVT,%DD.33::17,M7911_LF:5,LU:22,DP:1:0
DVT,%DD.33::17,M7912_LF:6,LU:23,DP:1:0
DVT,%DD.33::17,M7912_LF:7,LU:29,DP:1:0
DVT,%DD.33::17,M7914_LF:8,LU:30,DP:1:0
DVT,%DD.33::17,M7914_LF:9,LU:31,DP:1:0
DVT,%DD.33::17,M7933_LF:10,LU:34,DP:1:0
DVT,%DD.33::17,M7933_LF:11,LU:35,DP:1:0
*
* COMPATIBLE CARTRIDGE TAPE CACHE  LU 24  HP-IB ADDRESS 0
*
DVT,%DD.33::17,MTAPE,LU:24,DP:1:0
*
* 7906H HARD DISC   LU 12-15  HP-IB ADDRESS 1
*
DVT,%DD.30::17,M7906:0,LU:12,DP:1:1
DVT,%DD.30::17,M7906:1,LU:13,DP:1:1
```

```
DVT,%DD.30::17,M7906:2,LU:14,DP:1:1
DVT,%DD.30::17,M7906:3,LU:15,DP:1:1
*
*3.5" OR  5.25" FLEXIBLE DISC  LU 32,33  HP-IB ADDRESS 2
*
DVT,%DD.30::17,M7902,LU:32,DP:1:2:0:0:0,DP:5:2:66:16:2,TO:3000
DVT,%DD.30::17,M7902:0,LU:33,DP:1:2:1:0:0,DP:5:2:66:16:2,TO:3000
*
* 7910H FIXED DISC   LU 40-43    HP-IB ADDRESS 3
*
DVT,%DD.30::17,M7910:0,LU:40,DP:1:3
DVT,%DD.30::17,M7910:1,LU:41,DP:1:3
DVT,%DD.30::17,M7910:2,LU:42,DP:1:3
DVT,%DD.30::17,M7910:3,LU:43,DP:1:3
*
* HP-IB TAPE DRIVE    LU 8  HP-IB ADDRESS 4
*
DVT,%DD.23::17,M7970E:0,LU:8,DP:1:4,PR:1
*
* 8" FLEXIBLE DISC  LU 10,11  HP-IB ADDRESS 5
*
DVT,%DD.30::17,M7902:0,LU:10,DP:1:5,to:3000,TO:3000
DVT,%DD.30::17,M7902:1,LU:11,DP:1:5,to:3000,TO:3000
*
* 5.25" FIXED DISC (9134A/B SINGLE VOL.)  LU 52,53,54  HP-IB ADDRESS 6
*
DVT,%DD.30::17,M9134L:0,LU:52,DP:1:6
DVT,%DD.30::17,M9134L:1,LU:53,DP:1:6
DVT,%DD.30::17,M9134L:2,LU:54,DP:1:6
*
* 5.25" FIXED DISC (91341 FOUR VOL.)  LU 48-51   HP-IB ADDRESS 7
*
DVT,%DD.30::17,M9134:0,LU:48,DP:1:7
DVT,%DD.30::17,M9134:1,LU:49,DP:1:7
DVT,%DD.30::17,M9134:2,LU:50,DP:1:7
DVT,%DD.30::17,M9134:3,LU:51,DP:1:7
*
*
* 248x INTEGRATED DISC INTERFACE
*
IFT,%ID.27::17,SC:32B
*
* Hard disc
*
DVT,%GEN27::17,M2480:0,LU:36
DVT,%GEN27::17,M2480:1,LU:37
DVT,%GEN27::17,M2480:2,LU:38
*
* Microfloppy
*
DVT,%GEN27::17,M2480:3,LU:39
```

```
*
*
* HP-IB #2:  INSTRUMENT BUS
*
IFT,%ID.37::17,SC:25B
*
* HP-IB #2 CONTROLLER
*
DVT,,,LU:21,TO:50,DT:77B,DX:1,DP:1:36B
*
* FOUR DEVICES
*
DVT,,,LU:25,TO:500,DT:77B,DX:1,DP:1:1
DVT,,,LU:26,TO:500,DT:77B,DX:1,DP:1:5
DVT,,,LU:27,TO:500,DT:77B,DX:1,DP:1:6
DVT,,,LU:28,TO:500,DT:77B,DX:1,DP:1:7
*
* add an HP-IB with a 2608s and 2631
*
*
* HP-IB #3   PRINTERS
*
* create an IFT for the HP-IB card
*
IFT,%ID.37::17,SC:30B
*
* 2608S LINE PRINTER LU 85 HPIB ADDR 2
* the first driver parameter is the HP-IB address
*
DVT,%DDC12::17,,LU:85,DP:1:2
*
* HPIB LINE PRINTER LU 6 HPIB ADDR 6
* the first driver parameter is the HPIB address
*
DVT,%DD.12::17,,LU:6,DT:12B,DP:1:6
*
* D.S. LINKS, TWO LUS FOR D.S., TWO FOR LU MAPPING
*
* NETWORK LINKS
*
* IFT,%ID.66::A2,EID.66,SC:24B,QU:FI,TX:18
*
* DVT,,,LU:79,DT:66B
* DVT,,,LU:80,DT:66B
*
* LU MAPPING
*
* IFT,%ADV00::A2,EIDV00,SC:31B,QU:FI,TX:2
*
* DVT,,,LU:81,EDDV00,TX:0
*
```

```
* DVT,,,LU:82,EDDV00,TX:5
*
* eight MUX LU'S  SELECT CODE 23, LU 71-78
*
IFT,%IDM00::17,SC:23B
*
DVT,%DD.00::17,M26XX,LU:71,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:72,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:73,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:74,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:75,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:76,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:77,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:78,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
END
*
END
*
* DEFINE NODE LISTS
*
* SYSTEM CONSOLE AND TWO TAPE DRIVES
NODE,1,64,65
*
* this node was for the 2635 which was removed
* AUXILIARY CONSOLE/PRINTER
* NODE,66,67
*
* AUXILIARY CONSOLE WITH TWO TAPE DRIVES
NODE,68,69,70
*
* TWO 8" FLEXIBLE DISCS
NODE,10,11
*
* FOUR 7906 LU'S
NODE,12,13,14,15
*
* FOUR 7910 LU'S
NODE,40,41,42,43
*
* THIRTEEN 7908/11/12/14 LU'S AND A COMPATIBLE CARTRIDGE TAPE DRIVE
NODE,16,17,18,19,20,22,23,24,29,30,31,34,35
*
* TWO 3.5" OR  5.25" FLEXIBLE DISCS
```

```
*
NODE,32,33
*
* FOUR 5.25" FIXED DISC LU'S (9134 FOUR VOL.)
*
NODE,48,49,50,51
*
* THREE 5.25" FIXED DISC LU'S (9134A/B SINGLE VOL.)
*
NODE,52,53,54
*
* FOUR 248x INTEGRATED DISC LU'S
*
NODE,36,37,38,39
*
END,,,,NODE LIST
*
*
END,,,,INTERRUPT TABLE
*
*
CLAS,40
RESN,20
ID,40
RS,0
SAM,6144
*!!* put in spool buffer limits
*!!* you could also use generator defaults by specifying just SL
SL,350,750
BG,30
QU,300,50
*!!* allow for shared  program
SP,10
MB,500
*!!* allocate space for the users' table
US,10
*
* SYSTEM COMMON
*
* $ABLIB CONTAINS THE BASIC TRAP TABLES  (DELETED)
*
* RE,$ABLIB::17
*
* DS/1000 HAS BEEN GEN'ED INTO THIS SYSTEM FOR VERIFICATION
* OF HARDWARE ONLY. IF PROGRAM DEVELOPMENT CAPABILITY FOR
* DS IS DESIRED REPLACE $ENDLB WITH $FDSLB. $BIGDS IS A
* MERGED LIBRARY CONTAINING ENTRY POINTS FROM SEVERAL DS
* LIBRARIES
*
* DS/1000 LABELED COMMON AREA
*
```

```
* RE,%RESA::A2
* RE,$BIGDS::A2,#NRVS
* RE,$BIGDS::A2,#RQUA
* RE,$BIGDS::A2,#LEVL
* RE,$BIGDS::A2,D$EQT
* MS,$BIGDS::A2
*
END,,,,LABELED SYSTEM COMMON RELOCATION
COM,10
*
RE,%MSGS::17
END
*
LIB,$FNDLB.LIB::LIBRARIES
LIB,$BIGLB.LIB::LIBRARIES
*
* add the pascal library as a default library
*
LIB,$PLIB.LIB::LIBRARIES
*
*
END
```

```
* /lab/solution/lab15/s4.ans
* RTE-A.2  PRIMARY SYSTEM GENERATION ANSWER FILE
* D.K.G    REV. 2326      830624.1613
* l.e.n.   modified for RTE-A course  830809 answer file w/errors
*
* Solution for Lab 15, question 4, file = S4.ANS
* Original answer file = ANS2
*
* Look for *!!* to denote corrections
*
* %RPL60 CONTAINS THE A-600 RPL'S WITH NO CDS AND NO DOUBLE PRECISION
* FLOATING POINT.
* %RPL61 CONTAINS THE A-600 RPL'S WITH NO CDS AND DOUBLE PRECISION
* FLOATING POINT.
* %RPL62 CONTAINS THE A-600 PRL'S WITH CDS AND NO DOUBLE PRECISION
* FLOATING POINT.
* %RPL63 CONTAINS THE A-600 RPL'S WITH CDS AND DOUBLE PRECISION
* FLOATING POINT.
*
* %RPL70 CONTAINS THE A-700 RPL'S WITH NO CDS AND NO HARDWARE FLOATING POINT??
* %RPL71 CONTAINS THE A-700 RPL'S WITH NO CDS AND HARDWARE FLOATING POINT.
* %RPL72 CONTAINS THE A-700 RPL'S WITH CDS AND NO HARDWARE FLOATING POINT.
* %RPL73 CONTAINS THE A-700 RPL'S WITH CDS AND HARDWARE FLOATING POINT.
*
* %RPL90 CONTAINS THE A-900 RPL'S WITH NO CDS.
* %RPL91 CONTAINS THE A-900 RPL'S WITH CDS.
*
* THE CARTRIDGE REFERENCE NUMBERS "MS" AND "A2" HAVE BEEN
* USED IN THIS ANSWER FILE FOR GENERATION PURPOSES IN THE
* SOFTWARE PRODUCTION ENGINEERING DEPT.  THEY MUST BE CHANGED
* TO MATCH YOUR CARTRIDGE REFERENCE NUMBERS FOR REGENERATION.
*
*!!* The cartridge reference designations in the primary don't match
*!!* my system and probably don't match the lab system, so I will change
*!!* them. On my system the RTE-A relocatables are on cartridge 17 and
*!!* the VC+ relocatables are on cartridge A2, and the libraries are
*!!* in /LIBRARIES.
*
*!!* This system is far too big to use base page linking
*!!* LINKS,BP
LINKS,CP
RE,%VCTR::17
* RE,%SPOOL::A2
RE,%EXEC::17
RE,%MEMRY::17
* RE,%CDSFH::A2
*!!* the correct RPL file must be gen'ed in
RE,%RPL60::17
* RE,%RPL61::17
* RE,%RPL62::A2
* RE,%RPL63::A2
```

```
* RE,%RPL70::17
* RE,%RPL71::17
* RE,%RPL72::A2
* RE,%RPL73::A2
* RE,%RPL90::17
* RE,%RPL91::A2
RE,%SAM::17
RE,%TIME::17
RE,%SCHED::17
RE,%STRNG::17
RE,%LOCK::17
RE,%ERLOG::17
RE,%OPMSG::17
RE,%XCMND::17
RE,%SYCOM::17
RE,%STAT::17
RE,%LOAD::17
RE,%RTIOA::17
RE,%IOMOD::17
RE,%PERR::17
RE,%CLASS::17
RE,%ID.43::17
* RE,%#SPLU::A2
*!!* you must do multiple searches of $SYSA to resolve all
*!!*   undefined externals
*!!* SE,$SYSA::17
MS,$SYSA.LIB::LIBRARIES
SE,$SYSLB.LIB::LIBRARIES
END
*
RE,%DD.33::17
RE,%ID.52::17
END
*
* RE,%ID.66::A2
RE,%ID.00::17
END
*
RE,%ID.37::17
RE,%DD.30::17
END
*
RE,%IDM00::17
END
*
RE,%DD.00::17
ALIGN
RE,%ID.27::17
RE,%ID.50::17
END
*
```

```
RE,%DD.12::17
* RE,%ADV00::A2
RE,%DD.20::17
END
*
RE,%DDC12::17
END
*
*!!* the mag tape device driver was left out
RE,%DD.23::17
END
*
*    end driver partition
END
*
*  BEGIN TABLE GENERATION
*  CONFIGURE LU TABLES
*
* ASIC FOR 2621A/P SYSTEM CONSOLE WITH VCP
*
IFT,%ID.00::17,SC:20B
*
DVT,%DD.00::17,M26XX,LU:1,QU:FI,-
   DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.20::17,M264X:1,LU:64
DVT,%DD.20::17,M264X:2,LU:65
*
* ASIC FOR 2635A AUXILIARY CONSOLE/PRINTER
*
IFT,%ID.00::17,SC:33B
DVT,%DD.00::17,M2635:0,LU:66,QU:FI,-
   DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M2635:1,LU:67
*
* ASIC FOR 2645A AUXILIARY CONSOLE WITH DUAL MINI-CARTRIDGE
*
IFT,%ID.00::17,SC:22B
DVT,%DD.00::17,M26XX,LU:68,QU:FI,-
   DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.20::17,M264X:1,LU:69
DVT,%DD.20::17,M264X:2,LU:70
*
* PARALLEL INTERFACE CARD
*
IFT,%ID.50::17,SC:26B
DVT,,,LU:84,TO:5000,DX:2,DP:1:1:2,DT:45B
*
* PARALLEL INTERFACE FOR CPU TO CPU COMMUNICATION
*
IFT,%ID.52::17,SC:21B
*
```

```
DVT,,,LU:83,TO:0,DT:7,DX:0
*
* HP-IB #1
*
IFT,%ID.37::17,SC:27B
*
*HP-IB #1 BUS CONTROLLER LU
*
DVT,,,LU:9,TO:2000,DT:77B,TX:0,DX:1,DP:1:36B,PR:0
*
*
* 7908/40/11/41/12/14/33 DISC WITH COMPATIBLE CARTRIDGE TAPE
* LU 16-20,22,23,29-31,34,35,24  HP-IB ADDRESS 0
*
*
DVT,%DD.33::17,M7908_LF:0,LU:16,DP:1:0
DVT,%DD.33::17,M7908_LF:1,LU:17,DP:1:0
DVT,%DD.33::17,M7908_LF:2,LU:18,DP:1:0
DVT,%DD.33::17,M7908_LF:3,LU:19,DP:1:0
DVT,%DD.33::17,M7908_LF:4,LU:20,DP:1:0
DVT,%DD.33::17,M7911_LF:5,LU:22,DP:1:0
DVT,%DD.33::17,M7912_LF:6,LU:23,DP:1:0
DVT,%DD.33::17,M7912_LF:7,LU:29,DP:1:0
DVT,%DD.33::17,M7914_LF:8,LU:30,DP:1:0
DVT,%DD.33::17,M7914_LF:9,LU:31,DP:1:0
DVT,%DD.33::17,M7933_LF:10,LU:34,DP:1:0
DVT,%DD.33::17,M7933_LF:11,LU:35,DP:1:0
*
* COMPATIBLE CARTRIDGE TAPE CACHE  LU 24  HP-IB ADDRESS 0
*
DVT,%DD.33::17,MTAPE,LU:24,DP:1:0
*
* 7906H HARD DISC   LU 12-15  HP-IB ADDRESS 1
*
DVT,%DD.30::17,M7906:0,LU:12,DP:1:1
DVT,%DD.30::17,M7906:1,LU:13,DP:1:1
DVT,%DD.30::17,M7906:2,LU:14,DP:1:1
DVT,%DD.30::17,M7906:3,LU:15,DP:1:1
*
*3.5" OR  5.25" FLEXIBLE DISC  LU 32,33  HP-IB ADDRESS 2
*
DVT,%DD.30::17,M7902,LU:32,DP:1:2:0:0:0,DP:5:2:66:16:2,TO:3000
DVT,%DD.30::17,M7902:0,LU:33,DP:1:2:1:0:0,DP:5:2:66:16:2,TO:3000
*
* 7910H FIXED DISC   LU 40-43   HP-IB ADDRESS 3
*
DVT,%DD.30::17,M7910:0,LU:40,DP:1:3
DVT,%DD.30::17,M7910:1,LU:41,DP:1:3
DVT,%DD.30::17,M7910:2,LU:42,DP:1:3
DVT,%DD.30::17,M7910:3,LU:43,DP:1:3
*
```

```
* HP-IB TAPE DRIVE    LU 8  HP-IB ADDRESS 4
*
DVT,%DD.23::17,M7970E:0,LU:8,DP:1:4,PR:1
*
* 8" FLEXIBLE DISC  LU 10,11  HP-IB ADDRESS 5
*
DVT,%DD.30::17,M7902:0,LU:10,DP:1:5,to:3000,TO:3000
DVT,%DD.30::17,M7902:1,LU:11,DP:1:5,to:3000,TO:3000
*
* 5.25" FIXED DISC (9134A/B SINGLE VOL.)  LU 52,53,54  HP-IB ADDRESS 6
*
DVT,%DD.30::17,M9134L:0,LU:52,DP:1:6
DVT,%DD.30::17,M9134L:1,LU:53,DP:1:6
DVT,%DD.30::17,M9134L:2,LU:54,DP:1:6
*
* 5.25" FIXED DISC (91341 FOUR VOL.)  LU 48-51   HP-IB ADDRESS 7
*
DVT,%DD.30::17,M9134:0,LU:48,DP:1:7
DVT,%DD.30::17,M9134:1,LU:49,DP:1:7
DVT,%DD.30::17,M9134:2,LU:50,DP:1:7
DVT,%DD.30::17,M9134:3,LU:51,DP:1:7
*
* 248x INTEGRATED DISC INTERFACE
*
IFT,%ID.27::17,SC:32B
*
* Hard disc
*
DVT,%GEN27::17,M2480:0,LU:36
DVT,%GEN27::17,M2480:1,LU:37
DVT,%GEN27::17,M2480:2,LU:38
*
* Microfloppy
*
DVT,%GEN27::17,M2480:3,LU:39
*
* HP-IB #2:  INSTRUMENT BUS
*
IFT,%ID.37::17,SC:25B
*
* HP-IB #2 CONTROLLER
*
DVT,,,LU:21,TO:50,DT:77B,DX:1,DP:1:36B
*
* FOUR DEVICES
*
DVT,,,LU:25,TO:500,DT:77B,DX:1,DP:1:1
DVT,,,LU:26,TO:500,DT:77B,DX:1,DP:1:5
DVT,,,LU:27,TO:500,DT:77B,DX:1,DP:1:6
DVT,,,LU:28,TO:500,DT:77B,DX:1,DP:1:7
*
```

```
* HP-IB #3   PRINTERS - SLOW DEVICES
*
IFT,%ID.37::17,SC:30B
*
* 2608S LINE PRINTER LU 85 HPIB ADDR 2
*
DVT,%DDC12::17,,LU:85,DP:1:2
*
* HPIB LINE PRINTER LU 6 HPIB ADDR 6
*
DVT,%DD.12::17,,LU:6,DT:12B,DP:1:6
*
* D.S. LINKS, TWO LUS FOR D.S., TWO FOR LU MAPPING
*
* NETWORK LINKS
*!!* remove DS if you haven't got it !!
*
*!!*IFT,%ID.66::A2,EID.66,SC:24B,QU:FI,TX:18
*
*!!*DVT,,,LU:79,DT:66B
*!!*DVT,,,LU:80,DT:66B
*
* LU MAPPING
*
*!!*IFT,%ADV00::A2,EIDV00,SC:31B,QU:FI,TX:2
*
*!!*DVT,,,LU:81,EDDV00,TX:0
*
*!!*DVT,,,LU:82,EDDV00,TX:5
*
* eight MUX LU'S  SELECT CODE 23, LU 71-78
*
IFT,%IDM00::17,SC:23B
*
DVT,%DD.00::17,M26XX,LU:71,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:72,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:73,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:74,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:75,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:76,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:77,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
DVT,%DD.00::17,M26XX,LU:78,QU:FI,DP:1:20004B,-
    DP:5:CI:20040B:20040B:0,DP:9:CM:20040B:20040B:CM
*
```

```
END
*
*!!* An END command was left out. There are 2 ENDs here - one to end the
*!!* end the IFT creation and one to end the DVT creation.
END
*
* DEFINE NODE LISTS
*
* SYSTEM CONSOLE AND TWO TAPE DRIVES
NODE,1,64,65
*
* AUXILIARY CONSOLE/PRINTER
NODE,66,67
*
* AUXILIARY CONSOLE WITH TWO TAPE DRIVES
NODE,68,69,70
*
* TWO 8" FLEXIBLE DISCS
NODE,10,11
*
* FOUR 7906 LU'S
NODE,12,13,14,15
*
* FOUR 7910 LU'S
NODE,40,41,42,43
*
* THIRTEEN 7908/11/12/14 LU'S AND A COMPATIBLE CARTRIDGE TAPE DRIVE
NODE,16,17,18,19,20,22,23,24,29,30,31,34,35
*
* TWO 3.5" OR  5.25" FLEXIBLE DISCS
*
NODE,32,33
*
* FOUR 5.25" FIXED DISC LU'S (9134 FOUR VOL.)
*
NODE,48,49,50,51
*
* THREE 5.25" FIXED DISC LU'S (9134A/B SINGLE VOL.)
*
NODE,52,53,54
*
* FOUR 248x INTEGRATED DISC LU'S
*
NODE,36,37,38,39
*
END,,,,NODE LIST
*
*!!* Following the node lists is the interrupt table creation. For this
*!!* gen, all interrupt table entries are automatically created by the
*!!* generator, but we still need to end the interrupt table generation.
*!!* The END was left out
```

```
END,,,,INTERRUPT TABLE
*
CLAS,40
RESN,20
ID,40
RS,0
SAM,6144
SL
BG,30
QU,300,50
SP,0
MB,500
*!!* The USers command was left out. All the commands in this phase for
*!!* creating table space must always be specified in the same order.
*!!* The commands relating to VC+ tables are always specified even if its
*!!* a non-VC+ system (set them to 0).
US,0
*
* SYSTEM COMMON
*
* $ABLIB CONTAINS THE BASIC TRAP TABLES  (DELETED)
*
* RE,$ABLIB::17
*
* DS/1000 HAS BEEN GEN'ED INTO THIS SYSTEM FOR VERIFICATION
* OF HARDWARE ONLY. IF PROGRAM DEVELOPMENT CAPABILITY FOR
* DS IS DESIRED REPLACE $FNDLB WITH $FDSLB. $BIGDS IS A
* MERGED LIBRARY CONTAINING ENTRY POINTS FROM SEVERAL DS
* LIBRARIES
*
* DS/1000 LABELED COMMON AREA
*
* RE,%RESA::A2
* RE,$BIGDS::A2,#NRVS
* RE,$BIGDS::A2,#RQUA
* RE,$BIGDS::A2,#LEVL
* RE,$BIGDS::A2,D$EQT
* MS,$BIGDS::A2
*
END,,,,LABELED SYSTEM COMMON RELOCATION
COM,10
*
RE,%MSGS::17
END
*
LIB,$FNDLB.LIB::LIBRARIES
LIB,$BIGLB.LIB::LIBRARIES
*
*
END
```

1. To make yourself a superuser and give yourself a hello file:

```
CI> users cuthbert
This program creates or modifies user accounts.
Use carriage return to take the choice in [brackets].
Use <cntl-D> to quit early.

Modifying user CUTHBERT
Your current logon name is CUTHBERT
Enter your new logon name: [CUTHBERT]
Your current real name is CuthbertQDivine
Enter your real name: [CuthbertQDivine]
Enter your password: [cr]
Set super user flag? (Yes or No) [No] yes
Your working directory is ::CUTHBERT
Enter your working directory: [::CUTHBERT]
Your current startup command is
RU CI.RUN::PROGRAMS
Enter your startup command: [RU CI.RUN::PROGRAMS]
RU CI HI::CUTHBERT

Modified user CUTHBERT

CI>
```

To create an additional account:

```
CI> users
This program creates or modifies user accounts.
Use carriage return to take the choice in [brackets].
Use <cntl-D> to quit early.

Creating a user
Enter your logon name: DIVINE
Enter your real name: [???] CuthbertQDivine
Enter your password: [cr] cqd
Set super user flag? (Yes or No) [No]
Enter your working directory: [::DIVINE] ::CUTHBERT
Enter your startup command: [RU CI.RUN::PROGRAMS]

Created user DIVINE

CI>
```

To remove this account:

```
CI> pu divine::users
```

2. This solution depends on the boot command string for your classroom system. The solutions here are for the boot command string:

   ```
   %bdc27sys78
   ```

   ```
   HPIB address = 0
   select code  = 27B
   boot command file = SYS78
   ```

   The welcome file is WELCOMEn.CMD, where n corresponds to the parameter in the ST command in the boot command file:

   ```
   ST,,<n>
   ```

   The system and snap files are denoted by the SY and SN commands, respectively, in the boot command file:

   ```
   SY,<system file>
   SN,<snap file>
   ```

   The mounted volumes are indicated by the MC commands in both the boot command file and the welcome file:

   ```
   MC,<lu>
   ```

   In the boot command file, the RP for the startup program is followed by the ST command. The startup program is CI.

   ```
   RP,CI.RUN::PROGRAMS
   ST,,<n>
   ```

   You can tell this is a VC+ system, because PROMT is enabled as the primary program for the terminals.

3. The boot command file is /LAB/SOLUTION/LAB16/S3.BOOT. The welcome file is /LAB/SOLUTION/LAB16/S3.CMD. All the required directories and programs are all ready available on the existing system.

4. The boot command file is /LAB/SOLUTION/LAB16/S4.BOOT. The welcome file is /LAB/SOLUTION/LAB16/S4.CMD. To initialize the spool system after booting:

   ```
   CI> sp in
   ```

```
* file /LAB/SOLUTION/LAB16/S3.BOOT
* Solution to Lab 16, question 3, Boot command file for non-VC+
* l.e.n. <840126.1303>
*
* echo commands
EC
*
* DEFINE SYSTEM AND SNAP FILES
SY,SYS1
SN,SNAP1
*
MC,-18
MC,-19
* DEFINE INITIALLY RP'ED PROGRAMS
* The disc directory program must be RPed as D.RTR
*
RP,DRTR::PROGRAMS,D.RTR
* On my system and probably on the lab systems, the non-CDS version
* of CI is called CINCD.RUN::PROGRAMS. On your own system, if it is
* a non-VC+ system, it will probably be called CI.RUN::PROGRAMS.
*
* RP the primary program, CI, and the secondary program, CM
*
RP,CINCD.RUN::PROGRAMS,CI
*
RP,CINCD.RUN::PROGRAMS,CM
* We need an extra copy of CI to be the startup program, because the
* startup CI will release its ID segment when it exits. This could be
* called anything.
*
RP,CINCD.RUN::PROGRAMS,START
*
* The welcome file is WELCOME3.CMD::SYSTEM
*
ST,,3
*
* END RP PHASE
END
*
* If you have enough memory in your system, you might want to assign
* D.RTR to a reserved partition.
*
*AS,D.RTR
* DEFINE SWAP FILE
*
* The SWAP file could be put on the boot lu or in any directory. Make
* sure you create the directory first
*
SW,SWAP::-16::2000
END
```

```
* file /LAB/SOLUTION/LAB16/S3.CMD
* Solution to Lab 16, question 3, welcome file for non-VC+
* l.e.n. <840126.1303>
*
* To correspond to  the boot command file solution this file
* should be called WELCOME3.CMD::SYSTEM
*
* Enabling Terminals - for every terminal, enable a copy of CI
* as the primary program, and CM as the secondary program. The
* parameter CM tells CI to act like CM.
*
* All terminals except LU 1 are commented out - just remove *
* to include terminal.
*
* One copy of CI and CM have all ready been RPed
*
cn 1 20b CI
cn 1 40b cm,,,CM
*
* Enable other async terminals - first we have to RP copies of CI and CM.
* Remember on my system the non-CDS CI is called CINCD
*
rp CINCD ci68
rp CINCD cm68
rp CINCD ci66
rp CINCD cm66
*
cn 68 20b ci68
cn 68 40b cm68,,,CM
cn 66 20b ci66
cn 66 40b cm66,,,CM
*
* Enabling Mux terminals - If I RP a copy of CI and CM for EVERY
* MUX terminal in the system, I will use up lots of ID segments !
* (That's why VC+ is good to have if you've got lots of terminals).
* This is just an example for 2 MUX terminals.
*
* RP a copy of CI and CM for each terminal
*
*rp cincd ci71
*rp cincd cm71
*rp cincd ci72
*rp cincd cm72
*
* Initialize MUXs and enable primary and secondary programs
*
*cn 71 30b 152330b
*cn 71 20b ci71
*cn 71 40b cm71,,,CM
*
*cn 72 30b 152331b
```

```
*cn 72 20b ci72
*cn 72 40b cm72,,,CM
*
* Print a welcome message to the system console
*
co "mess.txt::system 1
*
* You must set the system time whenever you re-boot the system.
* Set the time: tm <mon> <day> <year> <hr>:<min>:<sec>  <am/pm>
ex
* file /LAB/SOLUTION/LAB16/S4.BOOT
* Solution to Lab 16, question 4, Boot command file for VC+
* l.e.n. <840126.1303>
*
* echo commands
EC
*
* DEFINE SYSTEM AND SNAP FILES
SY,SYSVC
SN,SNAPVC
*
MC,-18
MC,-19
* DEFINE INITIALLY RP'ED PROGRAMS
* The disc directory program must be RPed as D.RTR
*
RP,DRTR::PROGRAMS,D.RTR
* We only need one version of CI to be the startup program. PROMT and
* LOGON take care of everything else. Since this CI is the startup, it
* will release its ID segment when it exits.
*
RP,CI::PROGRAMS,CI
*
* The welcome file is WELCOME4.CMD::SYSTEM
*
ST,,4
*
* END RP PHASE
END
*
* If you have enough memory in your system, you might want to assign
* D.RTR to a reserved partition.
*
*AS,D.RTR
* DEFINE SWAP FILE
* The SWAP file could be on the boot lu or in any directory. Make sure
* you create the directory first.
*
SW,SWAP::-16::2000
*
END
```

```
* file /LAB/SOLUTION/LAB16/S4.CMD
* Solution to Lab 16, question 4, welcome file for VC+
* l.e.n. <840126.1303>
*
* To correspond to the boot command file solution this file
* should be called WELCOME4.CMD::SYSTEM
*
* Enabling terminals for the multiuser environment
* All terminals except LU 1 are commented out - just remove * to
* include terminal
*
* Promt takes care of the scheduling of LOGON and CM. CI must be called
* CI.RUN::PROGRAMS
*
rp promt.run::programs
*
* mount additional cartridges
*
mc 22
mc 23
mc 29
*
* Enabling promt as the primary program for all the terminals on
* the system. No secondary program is needed. The CN command will
* timeout if the terminal is not physically connected.
*
cn 1 20b promt
cn 68 20b promt
cn 66 20b promt
*
* Initialize and enable the MUX terminals - remove the *'s to
* include the MUXs
*
* cn 71 30b 152330b
* cn 72 30b 152331b
* cn 73 30b 152332b
* cn 74 30b 152333b
* cn 75 30b 152334b
* cn 76 30b 152335b
* cn 77 30b 152336b
* cn 78 30b 152337b
*
*rp 71 20b promt
*rp 72 20b promt
*rp 73 20b promt
*rp 74 20b promt
*rp 75 20b promt
*rp 76 20b promt
*rp 77 20b promt
*rp 78 20b promt
*
```

```
* Print a welcome message to the system console
*
co "mess.txt::system 1
*
* You must set the system time whenever you re-boot the system.
*
* set the time: tm <mon> <day> <year> <hr>:<min>:<sec> <am/pm>
*
ex
*
```

1. The file /LAB/SOLUTION/LAB17/S1.CMD is the BUILD command file. To boot using the existing bootex:

    VCP> %bdc27<merged system file>

    Most of the CI commands can be used in this memory based version. The program DL was not built into the system, so you can't use the DL command, but you can use most of the other CI commands, such as RU, TM or LI.

2. The file /LAB/SOLUTION/LAB17/S2.CMD is the BUILD command file. To boot using the existing bootex:

    VCP> %bdc27<merged system file>

3. To install a memory based system on CTD:

    CI> ru csys <merged system file> <tape lu> 0

    To boot the system:

    VCP> %bdcl27

    To install a memory based system on magnetic tape:

    CI> co <merged system file> <tape lu>

    To boot the system:

    VCP> %bmt4027

```
 ,,,,, file /LAB/SOLUTION/LAB17/S1.CMD
yes
200
rp,cincd.run::programs,ci
rp,drtr.run::programs,d.rtr
rp,wh.run::programs
rp,area.run
pt
/e
 ,,,,,  file /LAB/SOLUTION/LAB17/S2.CMD
yes
200
rp,ci.run::programs,ci
rp,ci.run::programs,citoo
rp,drtr.run::programs,d.rtr
rp,wh.run::programs
rp,area.run
pt
/e
```

1. The "v" option will verify the backup or restore.

```
CI> wd /myglobal
CI> tf
TF: co tflab/@.@ 24 kv
TF: co 24 @.@ kdv
TF: co 24{tflab/@.@} moretflab/@.@ kv
TF: ex
```

2. When you copy the files to your DELTA subdirectory, the backup bits will be set. To do a full backup:

```
CI> tf
TF: co /myglobal/delta/@.@ 24 ck
```

After the changes, you can see which files have the backup bits set with the following:

```
CI> dl /myglobal/delta/@.@.b
```

The backup bits will be set for FILE4, FILE5 and NEWFILE. To do an incremental backup:

```
TF: co /myglobal/delta/@.@.b cak
```

To restore the files after you have purged them:

```
TF: co 24 @.@ v
```

After restoring, the backup bits will be set.

3. To copy all files on LU 16 beginning with % :

```
CI> fc co %-----::16 -24
```

4. This information can be obtained by running FREES on 23:

```
CI> frees 23
```

and on all the CI volumes:

```
CI> frees
```

To pack LU 23:

```
CI> fpack 23
```

FPACK will not increase the amount of free space on the disc - it merely rearranges it. The files which should be moved to increase the largest free space are listed by the FPACK program.

5. To verify a disc LU:

```
CI> fveri <lu>
```

Minor errors (low numbers) can be ignored or the error can probably be corrected by copying the affected files to another disc or tape, purging them and restoring them back. In the case of inconsistencies in the free space table (the bitmap), it may be necessary to back up the entire disc, reinitialize it and restore it.

6. The FMGR command shows what FMGR LUs are mounted and the last track available on the LU (that is, the size). LU 16 on the primary system is 19200 blocks. To determine how much space is available, subtract the next track (NXTR) from the last track.

7.

```
CI> asave
ASAVE: ta 24
ASAVE: sa 20 ve nl
ASAVE: ex
```

READER COMMENT SHEET

Manual Name: ______________________________
(Please Print)

Part Number: ______________________________

We welcome your evaluation of this publication. Your comments and suggestions will help us improve our training materials. Please use additional pages if necessary.

Is this book technically accurate?

Did it meet your expectations?

Was it complete?

Is it easy to read and use?

Other comments?

FROM:

Name ______________________________

Company ______________________________

Address ______________________________

____

____

Training Coordinator/Technical Marketing
Hewlett-Packard Co.
11000 Wolfe Road
Cupertino, California 95014