UNISYS

1100/2200 MAPPER® Advanced Run Design Efficiency

Student Guide

Volume 1

Level 35R1

October 1991

Printed in U S America

SE 1565

UE 8476R2

Correspondence regarding this publication should be forwarded to Unisys Corporation, Education Publications, P.O. Box 1110, Princeton, NJ 08543 U.S.A.

Contents

Course Description

Audience

Experienced run designers who build and maintain MAPPER runs.

Prerequisites

AL 2815 – MAPPER Basic Run Design or AL 2805 – MAPPER Jump Start for Programmers, and six months experience using run function and their options.

Objectives

Upon completion of this course, the students should be able to:

- Apply advanced run writing techniques to create, process and update the MAPPER database.
- Demonstrate an understanding of internal MAPPER run processing by using advanced techniques to design efficient MAPPER runs.

Description

- This course provides the experienced MAPPER run designer with the tools and techniques to design complex MAPPER runs; emphasis is placed on efficiency. Lectures are enhanced by hands-on exercises to apply methods of analyzing and improving run efficiency.

Topics

- Components of MAPPER internal run processing
- Components of the MAPPER database
- Structure of a run control report
- Function power curve
- Run efficiency analysis tools
- Input/Output considerations
- Recovery tape concerns
- Screen design methods
- Variable stacks
- Run Debug
- Branching, looping, and subroutines
- Function efficiency within runs
- Run error and recovery techniques
- Importing and exporting data
- Data indexing schemes

Duration

5 days

Agenda

Day 1

- Module 1
- Module 2

Day 2

- Module 3
- Module 4
- Module 5

Day 3

- Module 6
- Module 7

Day 4

- Module 8
- Module 9

Day 5

- Module 10
- Module 11

1

MAPPER Overview

Module 1
MAPPER Overview

Objectives

Upon completion of this module, you should be able to:

1. List the steps involved when a request is issued from the MAPPER control line.
2. Define PreRun.
3. List the files in a simple MAPPER Database.
4. Recognize the MAPPER tables and database files in which a MAPPER report can reside.

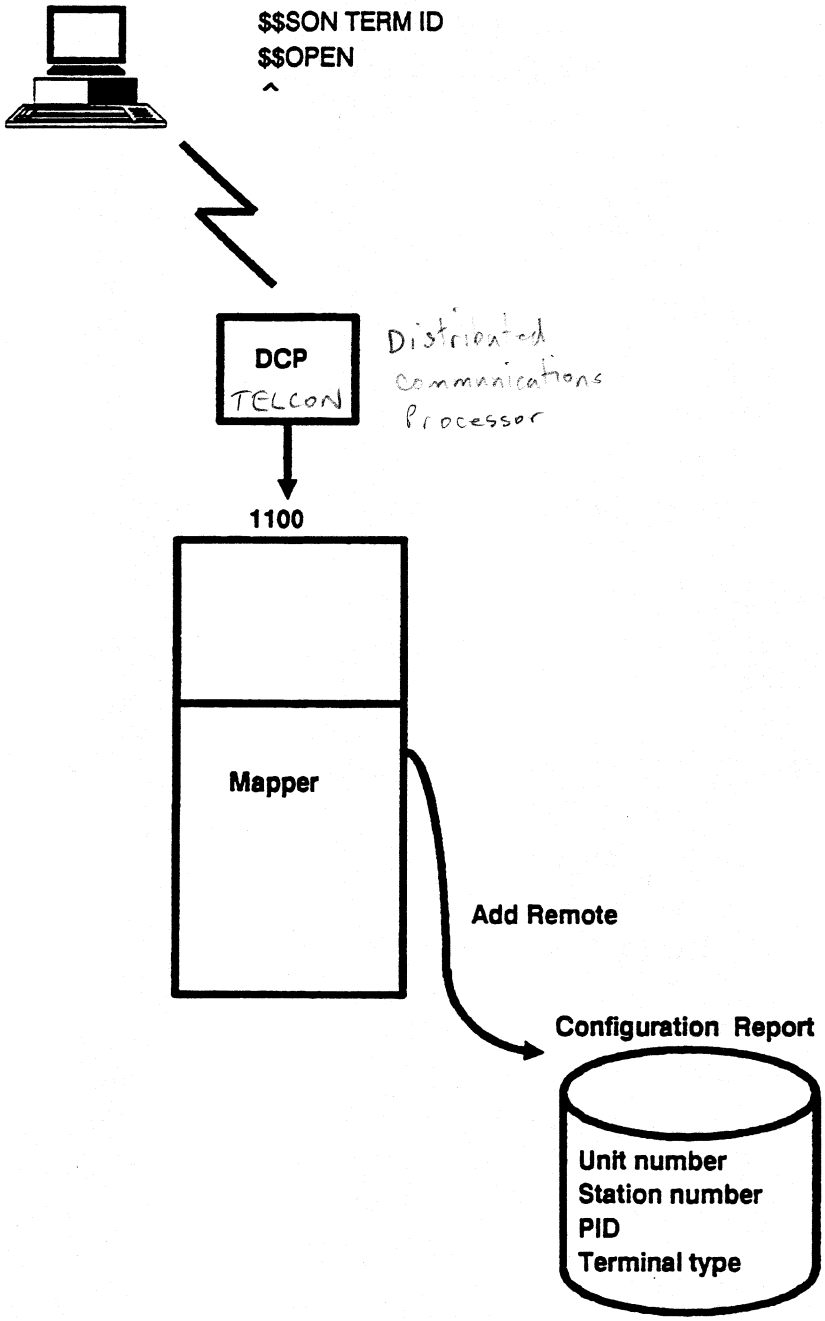
Establishing Communications with the 1100

- **$$SON term-id**
  - TELCON commands to Distributed Communications Processor (DCP) identifying the terminal on which the user wants to work
- **$$OPEN MAPPER**
  - (DCP) command identifying the system on which the user wants to work.

Establishing Communication with MAPPER

- A caret or some other identifier to the MAPPER application is transmitted. At this point, the (ADD REMOTE) routine is executed
- ADD REMOTE:
  - Checks the Configuration Report in the coordinator's cabinet for the:
    - Unit Number/Station Number/Pid
    - Type of terminal
    - Printers attached
  - Enters information into the users Station Table
  - User receives the idle logo if this terminal is configured for MAPPER usage
  - User will receive the error message if terminal is not configured. **< Unauthorized Station >.**

Establishing Communcations



User Sign On

- When Idle Logo returns to screen
- Enter **]userid,dept#,password** on line zero
- One character command (]) executes the MAPPER routine called SIGN-ON.

  - SIGN-ON enters the Department's User Registration Report in the coordinator's database and verifies:

    - User-id
    - Department#
    - Password

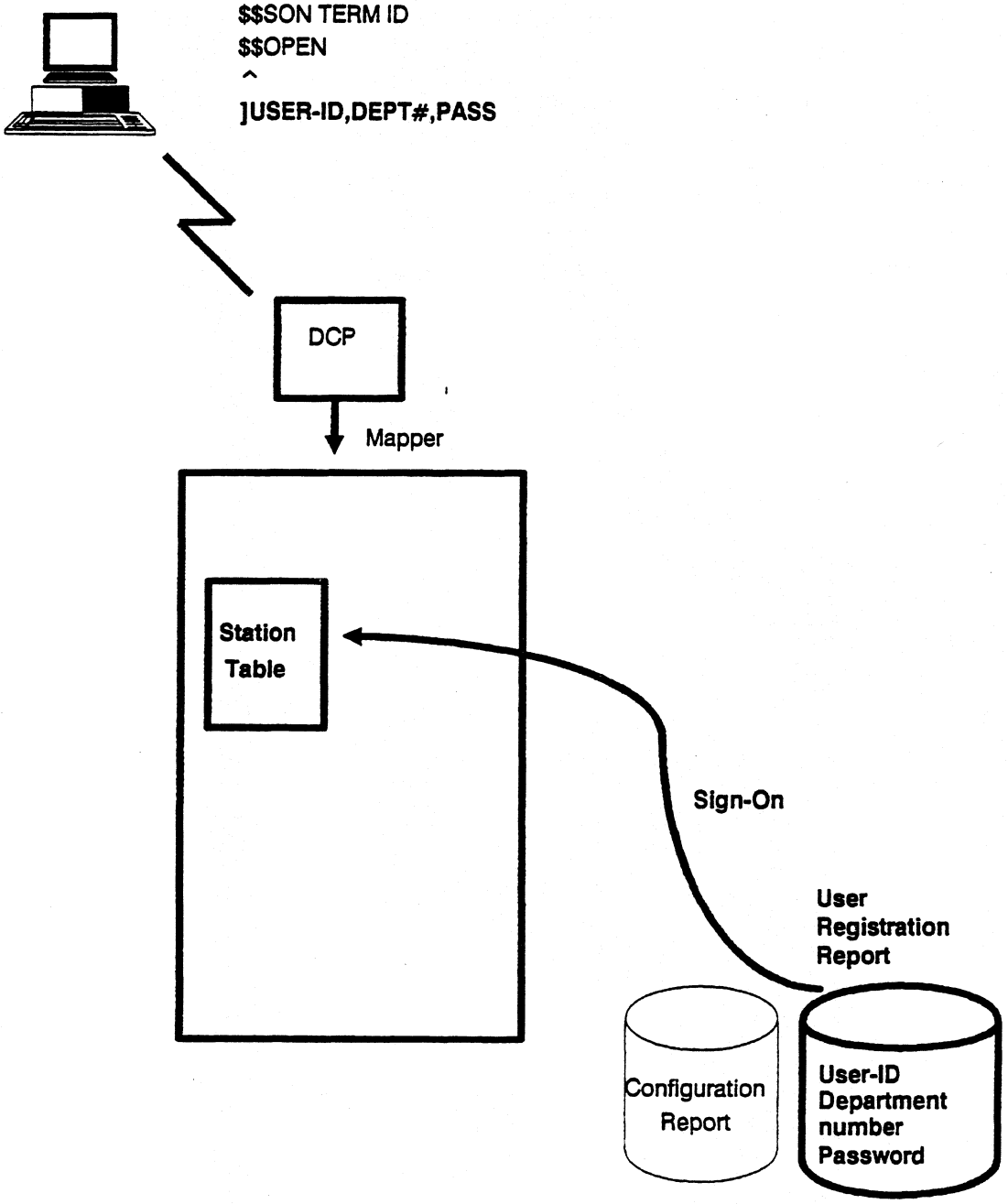
- If valid, the active logo displays on the screen and the SIGN-ON routine:

  - Places the user in a cabinet
  - Updates the Station Table with information as to which functions this user may or may not use

- If not valid, one of three error messages may appear:

  - **< Invalid Department >**
  - **< Invalid User-id >**
  - **< Invalid Password >**

Signing On


$$SON TERM ID
$$OPEN
^
]USER-ID,DEPT#,PASS
DCP
Mapper
Station Table
Sign-On
User Registration Report
Configuration Report
User-ID
Department number
Password

Internal Process

Anything transmitted on the control line must be interpreted by MAPPER:

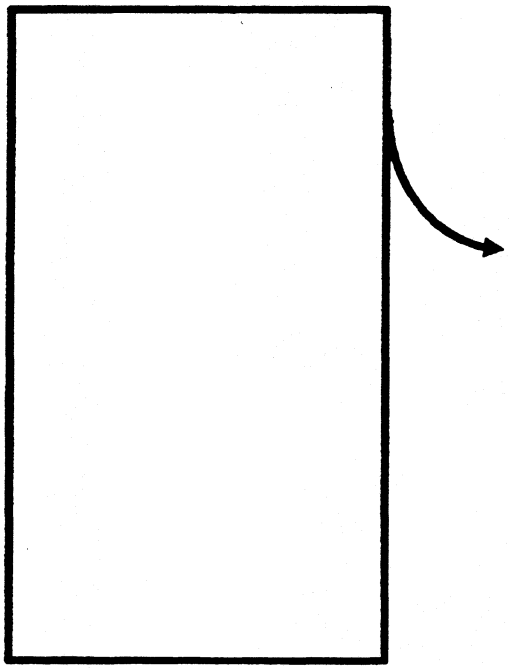
- Is it a Function that this user can execute?
- Is it a Department Run?
- Is it an External Run (Local Code)?
- Is it a Global Run?
- If none of the above, produces an error message **< Invalid Request >**

Internal Process

$$SON TERM ID
$$OPEN
^
]USER-ID,DEPT#,PASS
Manual Function? or Run name?

DCP

Mapper



MANUAL FUNCTION?

DEPARTMENT RUN?

EXTERNAL RUN?

GLOBAL RUN?

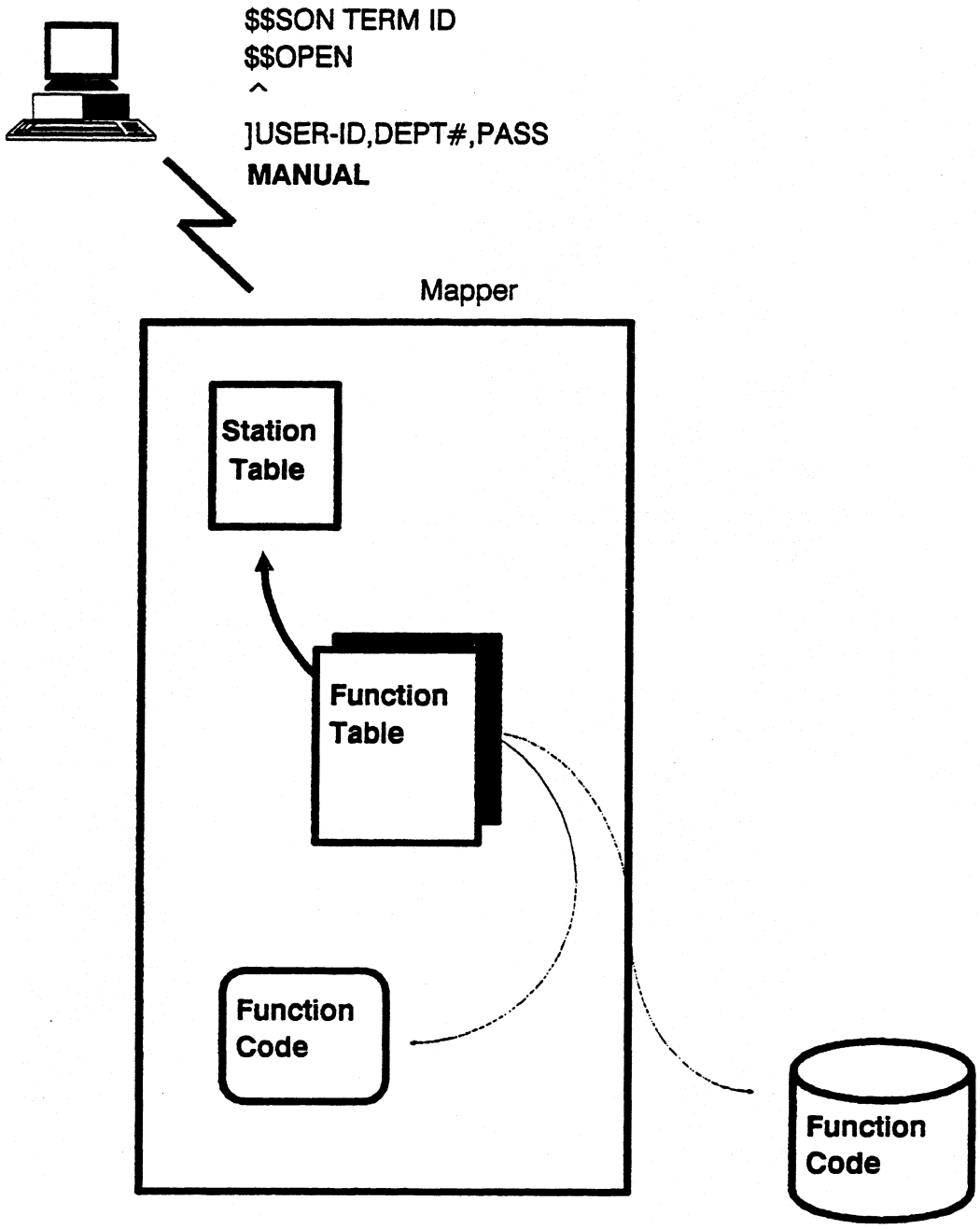
< INVALID REQUEST >

Manual Function Sequence

MAPPER routines implemented in order to execute a manual function:

Step 1:	Scans the system Function Table to see if this is a valid manual command.
Step 2:	Checks the Station Table to see if this user has permission from coordination to execute this function.
Step 3:	If function is in memory, execute.
Step 4:	If function is not in memory, is there enough room in memory for another item.
Step 5:	If yes, brings in the function code!
Step 6:	If no, swaps an unused function out and swap the specified function in.

Manual Function Sequence



PreRun

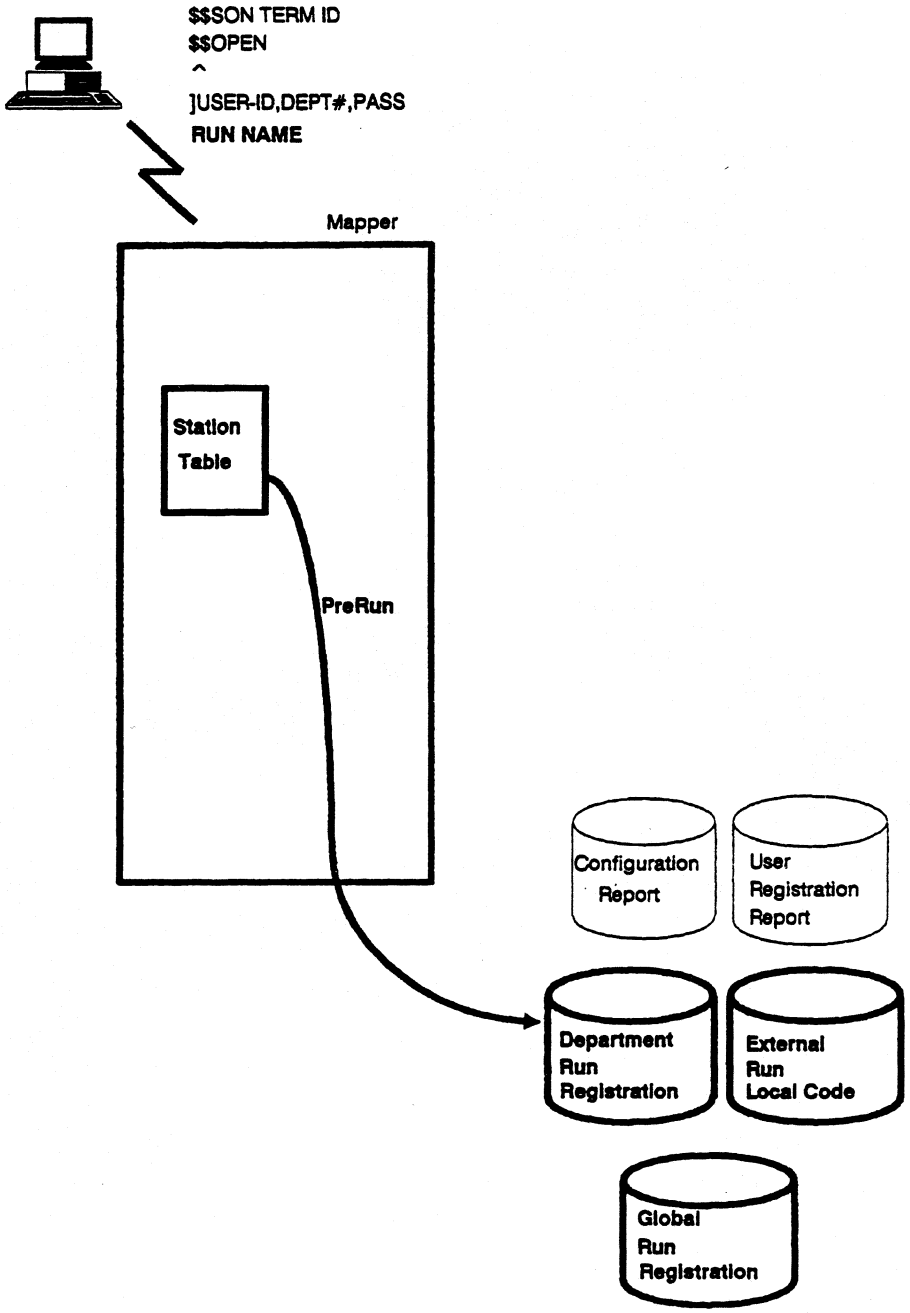
If MAPPER cannot find the manual function entered on the control line in its function tables, it then starts a MAPPER routine called **PreRun.**

PreRun's job is to:

Step 1:	Validate the run name
Step 2:	Check applicable restrictions
Step 3:	"Prep" the run

- **Step 1:** Validate the Run Name by checking:
  - Department's Run Registration Report
  - Global Run Registration Report.
  - If no match is found the user receives an error message **< Invalid Request >**

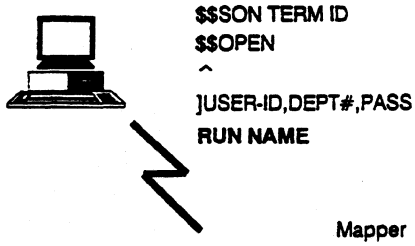
PreRun


$$SON TERM ID
$$OPEN
^
]USER-ID,DEPT#,PASS
RUN NAME
Mapper
Station Table
PreRun
Configuration Report
User Registration Report
Department Run Registration
External Run Local Code
Global Run Registration

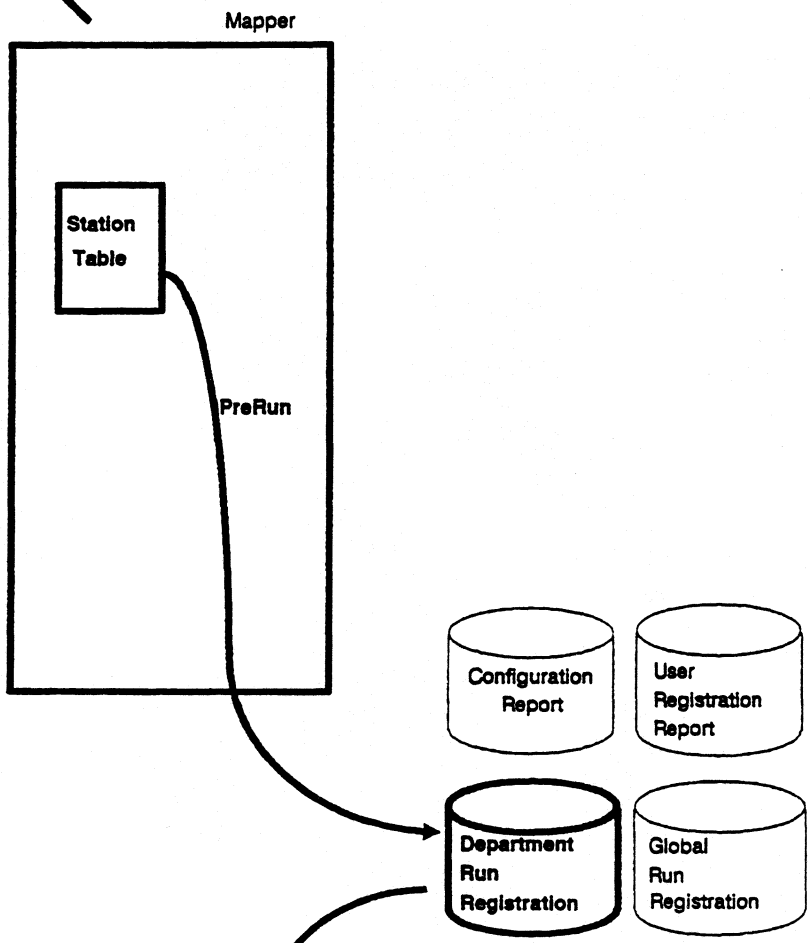
PreRun

- **Step 2:** PreRun checks for applicable restrictions:
  - Is the run restricted to a particular station number? (UNIT field)
  - Is the run restricted to a particular user-id? (USER field)
    - A blank UNIT field or USER field indicates that any station or user may execute this run.
  - Is the run restricted to execute at a certain time of day? (BTIME/ETIME fields)
  - If no restrictions apply for this run name, Step 3 is started. Otherwise, an applicable error message will be displayed.

Run Registration Rid



$$SON TERM ID
$$OPEN
^
]USER-ID,DEPT#,PASS
RUN NAME




```
LINE▶ 1     FMT▶   RL▶ -     SHFT▶     HLD CHRS▶    HLD LN▶   UNDO▶   8E218    ▶
.DATE 19 SEP 90  15:26:33  RID      8E   19 SEP 90  WEBMJG
.RUN CONTROL FOR :                                                        E003330
.                    *****KEEP RID SORTED BY RUN NAME*****
*  RUNID     B.   USER     P. UNIT. TYPE . RID.F. B TIME . E TIME . I/O .LINES.MOD
*============,============,=====,======.====.=,========,========,=====.=====.===
 ADD2                              000010    9                      1000  2000 0
 ADD3                              000010   13                      1000  2000 0
 ENTER                             000010  110                      1000  2000 0
 EXECUTE                           000010   50                      1000  2000 0
 FIDO                              000010   24                      1000  2000 0
 MELVIN                            000010   52                      1000  2000 0
```

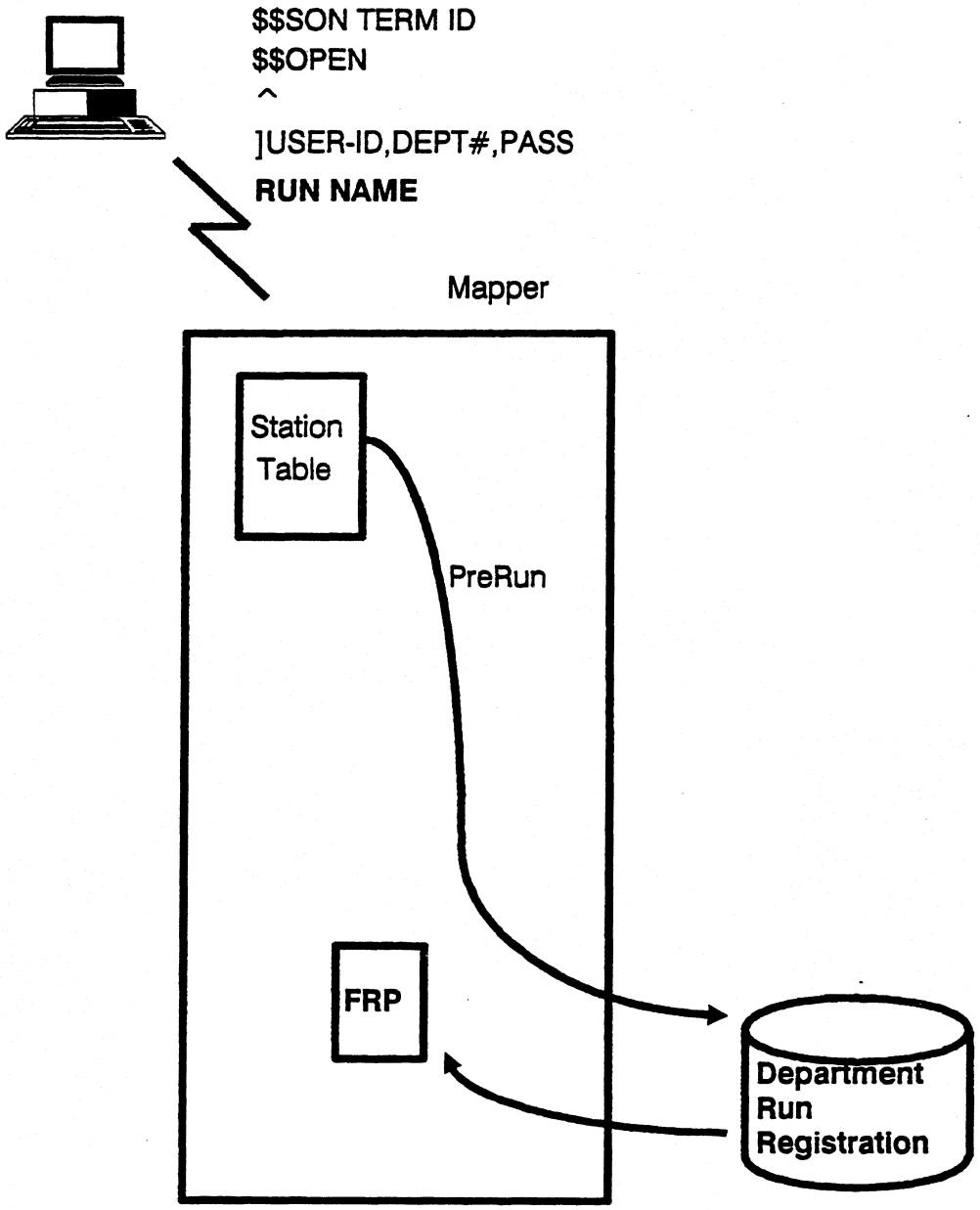
PreRun

Step 3: PreRun "preps" the run. PreRun preps the run by building what is known as a Function Request Packet (FRP).

- Function Request Packet contains the rest of the Run Control Report (RCR) characteristics:
  - Cabinet access (CABINET field)
  - Priority of execution, I/0, LLP limits (P field, I/O field, LINES field)
  - Cabinet and drawer of Run Control Report (DRAWER field, RID field)
- Function Request Packet is passed between MAPPER routines
  - Each routine decides what information it needs, utilizes it and replaces the information back into the Function Request Packet.

Note: PreRun I/Os are counted against your run!

Function Request Packet



Finding the Run Control Report (RCR)

Before the RCR may execute, MAPPER must determine where the RCR resides and acquire space for it in main storage.

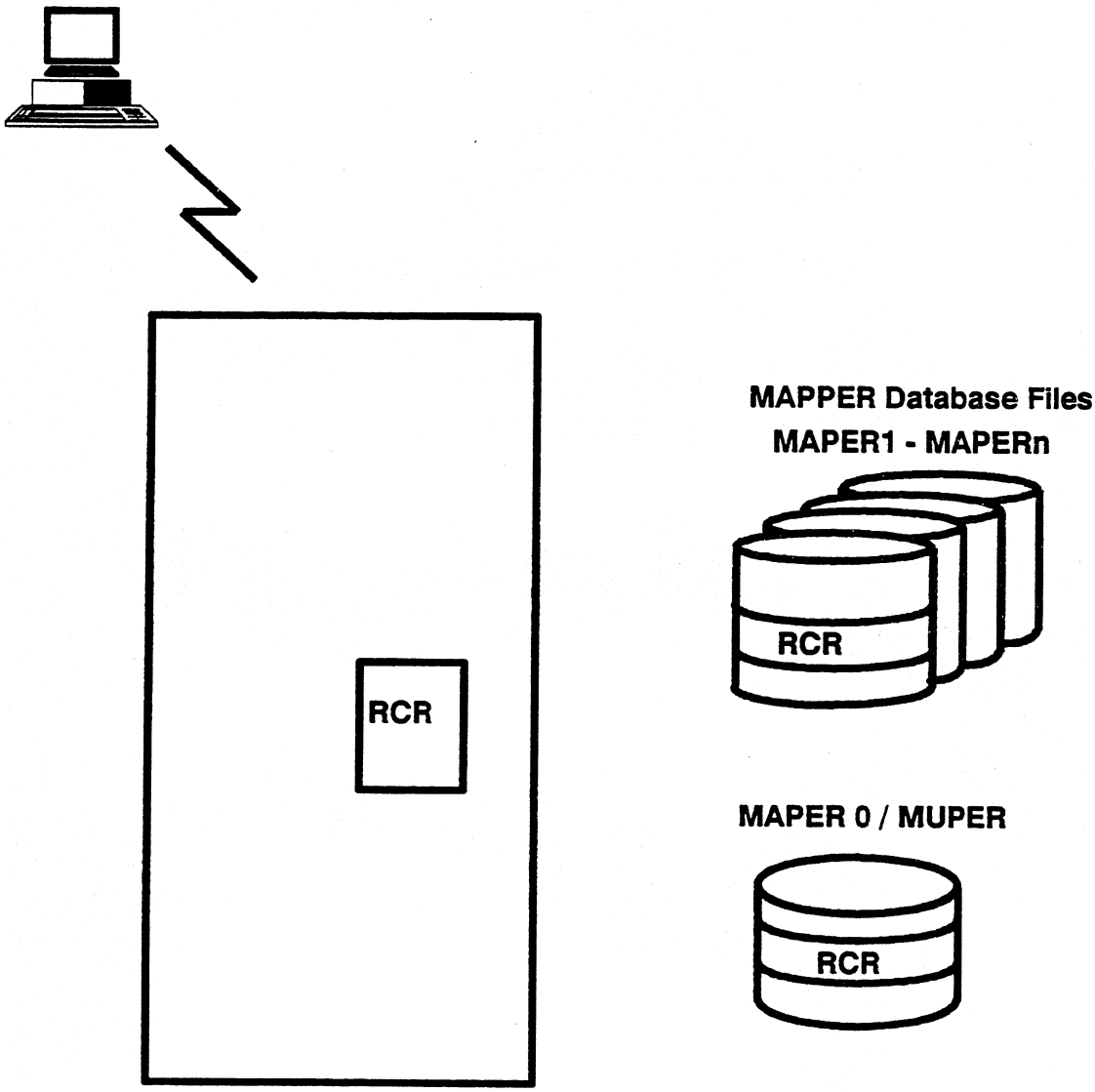
Note: An RCR is just a report that contains run statements. An RCR is no different from reports 2B,1D,2C which usually contain data.

Database Files

A report can exist in any one of three places:

- In Main Storage
- In a MAPERn file
- In MAPER0/MUPERx

Finding the RCR



Database Files

How does MAPPER know where the report is? Before answering this question, let's examine the database... What does the database look like?

There are the database files, MAPERn files, which are simply 1100 Program files. 1100 Program files have a specific structure. Each file has a:

- Table of Contents(TOC) containing pointers to the element's residence in the file
- Elements which are known to us as reports

The naming convention for 1100 Program files is as follows:

Qualifier*Filename.Elementname/Version

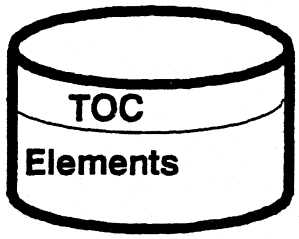
MAPPER*MAPER1.Type-000002/Rid-1
(octal)

An 1100 Program file can contain more than one Drawer. The maximum number of MAPERn files is MAPER1-MAPER1024. All reports initially reside on MAPERn files - the "home" file.

@PRT,T Lists elements in program file.

MAPERn

MAPER1 - MAPERn



1100 Program File

TOC	
Element/Rid	000002-1
Element/Rid	000002-2
Element/Rid	000002-3
Element/Rid	000002-4
Element/Rid	000002-5

each entry here is 10 words long (max 5004 entries)

MAPER0

MAPER0 is also part of the database. However, there is only one MAPER0 per system. MAPER0 is divided into three sections:

- FPOS which contains system tables
- MPOS which contains results (-0 through -7)
- SPOS which contains updated reports and newly created reports

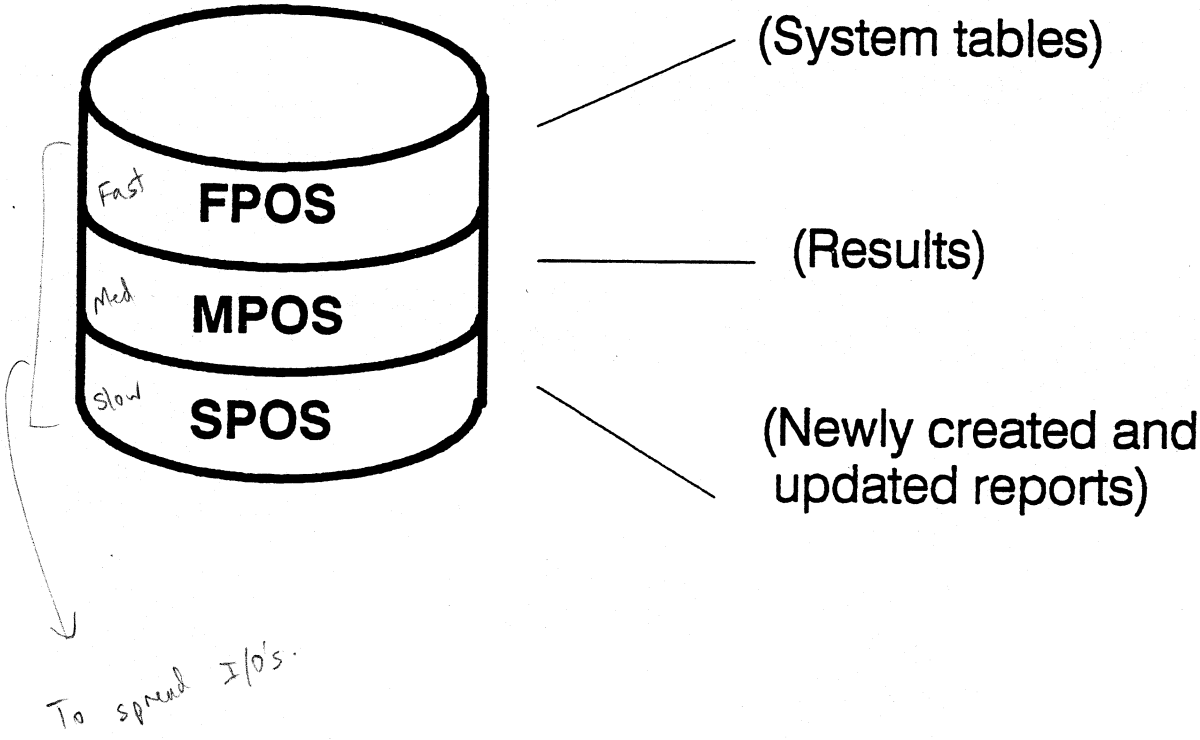
These areas derived from older days when the 1100 worked with various speed devices -- FPOS = Fast position, MPOS = Medium position, SPOS = Slow position. They were there to prevent device contention.

Today, speed is no longer a consideration. MAPER0 continues to be divided to prevent it from becoming overburdened and slow. If MAPER0 fills to maximum capacity, the message **< Scratch file full >** appears. Wait a few seconds and try again. MAPER0 doesn't have room for your update and so coordination will either delete reports from MAPER0 or Purge, Cycle/Merge.

Note: If the recovery technique utilizes Cycle/Merge, then the function of SPOS in MAPER0 is replaced by UPOS in the two files MUPER1 and MUPER2.

MAPER0/MUPER Files

(MAPPER UPDATE)



More System Tables

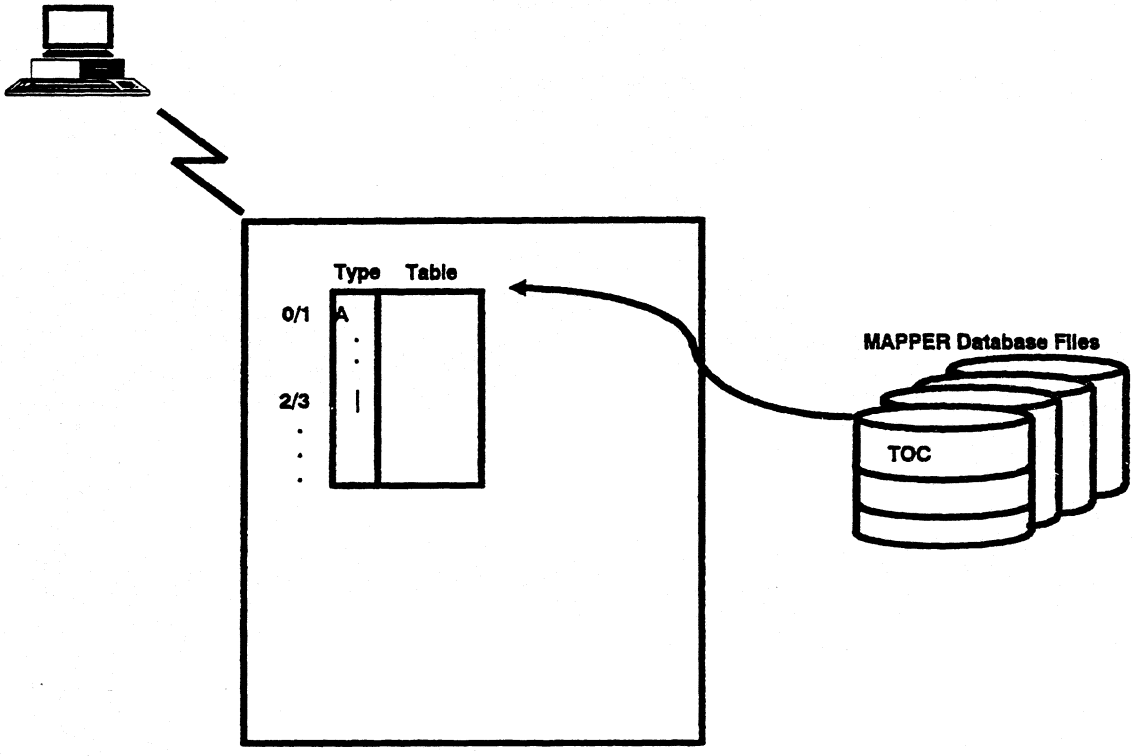
It would be costly in terms of I/O's and system resources for MAPPER to search up to 1024 MAPERn files Table of Contents to find a report.

So, during initialization MAPPER builds two more system tables: Type Table and Rid Table.

The Type Table:

- Is an index of all the cabinets and drawers existing on a system (One Type Table per system).
- Is built from the TOCs of the MAPERn files.
- Remains resident in main storage at all times.
- Contains the "Home" file/MAPERn file of report
- Contains the Address of Rid Table
- Has a Flag indicating if Rid Table is in main storage or not

Type Table



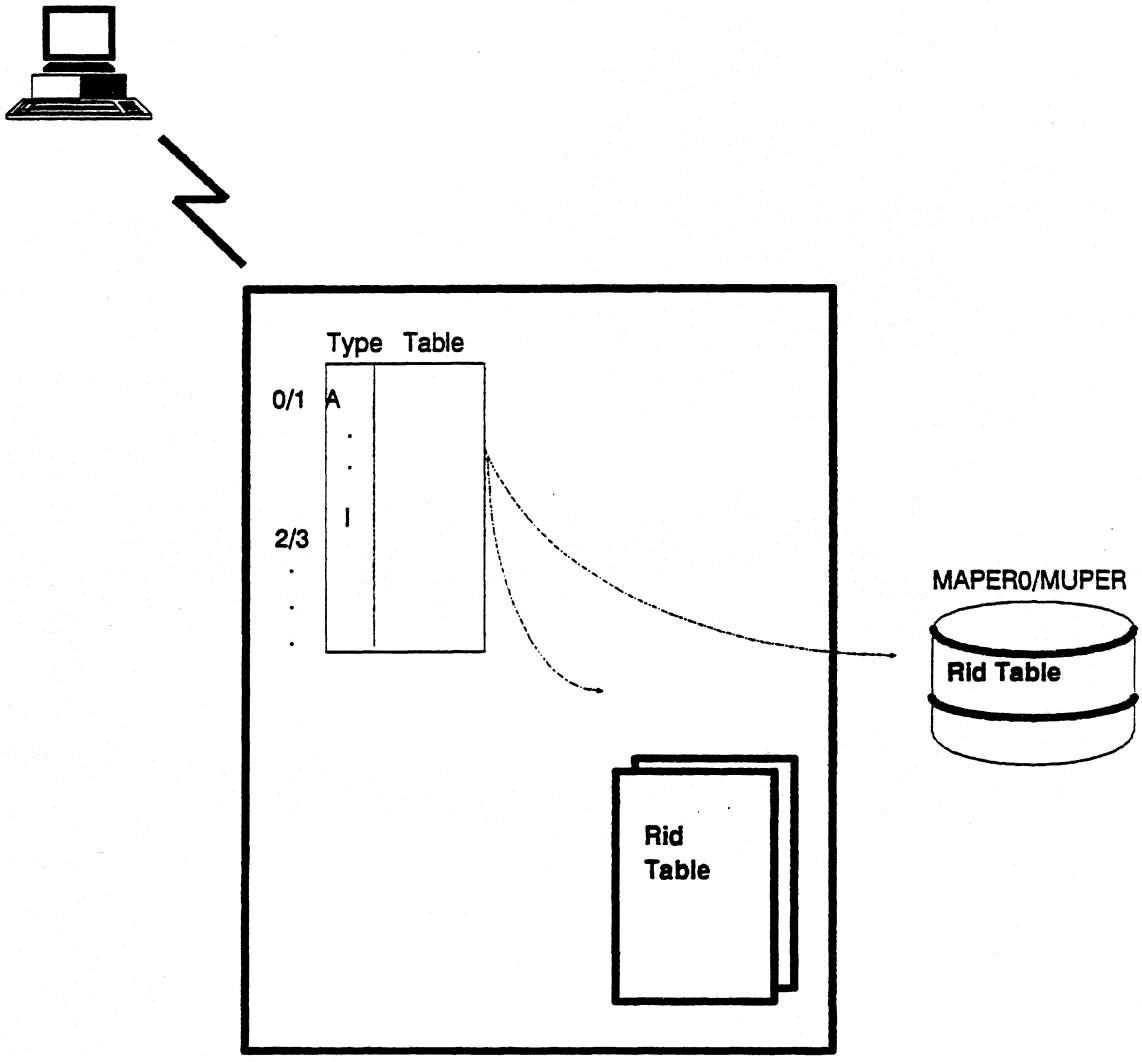
More System Tables

The Type Table points to the appropriate Rid Table. The Rid Table:

- Resides in main storage as long as it is being used.
- When no longer in use, it may be swapped out to FPOS of MAPER0 (I/O).
- Contains the file number of where the report currently resides (MAPERn or MAPER0)
- Contains the start address of the report (sector address of report)

A copy of the Type Table and Rid Table are kept in MAPER0's FPOS section for back-up purposes.

Rid Table

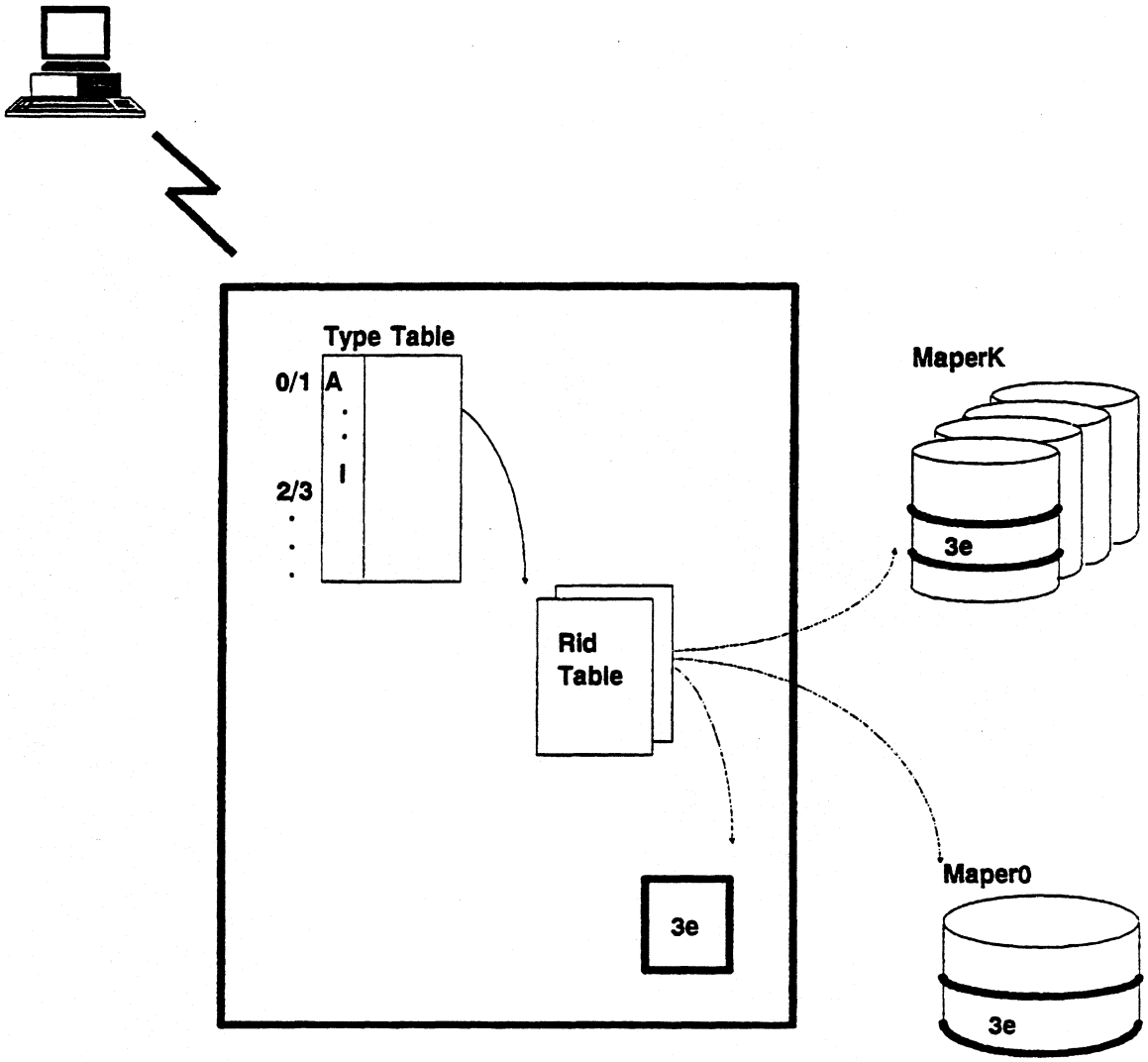


More System Tables

The Rid Table points to where the report actually resides. Many possibilities exist; however, three most common cases:

- If the report has been updated or just created, it will exist in MAPER0/MuperN - because updating to the MAPERn file's Table of Contents does not occur online.
  - One I/O to access the report
- If the report is being accessed for the first time, it will be in a MAPERn file.
  - Possibly one I/O to Rid Table
  - One I/O to access the report
- If the report has been accessed and not updated, it will exist in main storage.
  - Possibly one I/O to Rid Table

More System Tables



Exercise

d. Station Table	a. User-id, dept, psw
d. Configuration Report	b. Processor distributing communication
a. User Registration Report	c. Manual commands
b. DCP	d. Station#, terminal type
i. Telcon Commands	e. Home file, address, flag
c. Function Table	f. Tracks end user
j. Run Registration Report	g. Cart of information
h. FRP	h. Current residence and sector address
h. Type table	i. Double dollar signs
h. Rid table	j. Run name

Step through search sequence:

1. Manual function?
2. Department Run?
3. ~~Global Run~~? External Run?
4. Global Run?

Step through manual sequence:

1. Validate command is in Function Table
2. Validate user permissions in Station Table
3. Execute function if in memory
4. If function not in memory, try to load
5. If can load, do it
6. If can't load, swap with an unused function

What is PreRun's purpose?

Builds FRP - function request packet in memory to 'prep' the run, to save accessing Run Registration Report.

- Validate run name
- Check applicable restrictions

2

RCR Execution

Module 2
RCR Execution

Objectives

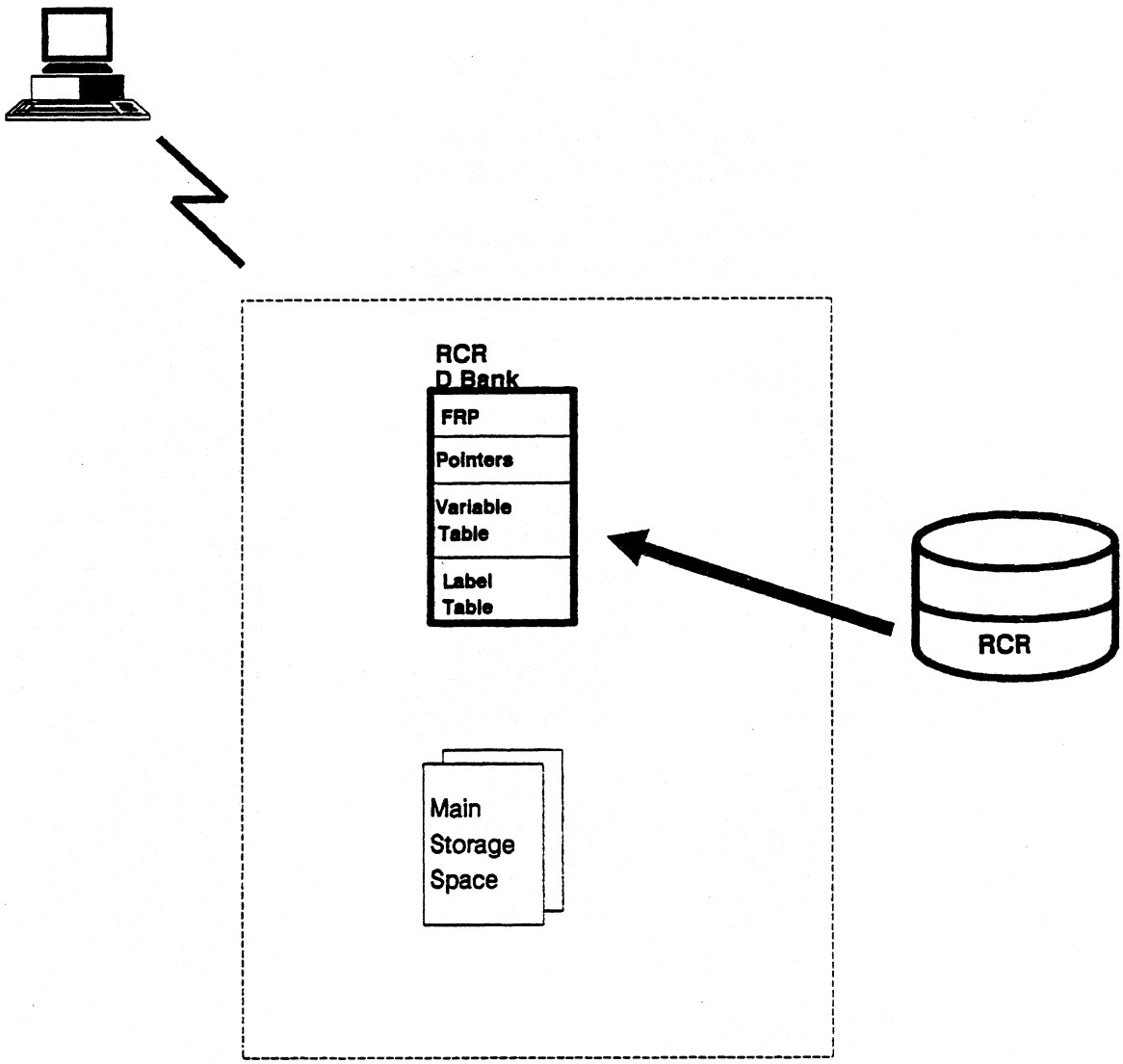
Upon completion of this module, you should be able to:

1. Describe the basic components of the RCR D-Bank.
2. Distinguish between RCR processing and function processing.
3. Calculate buffer sizes in terms of words and lines.

RCR Execution

- Control Table must be built
- Main storage space must be acquired
- Control Table
  - Often referred to as the RCR's Data Bank (D Bank)
  - One for every RCR
  - Contains all information for proper run execution
    - Function Request Packet
    - Pointers to sections of memory
    - Variable Table
    - Label Table

Control Table

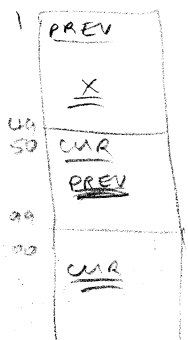


Acquiring Main Storage

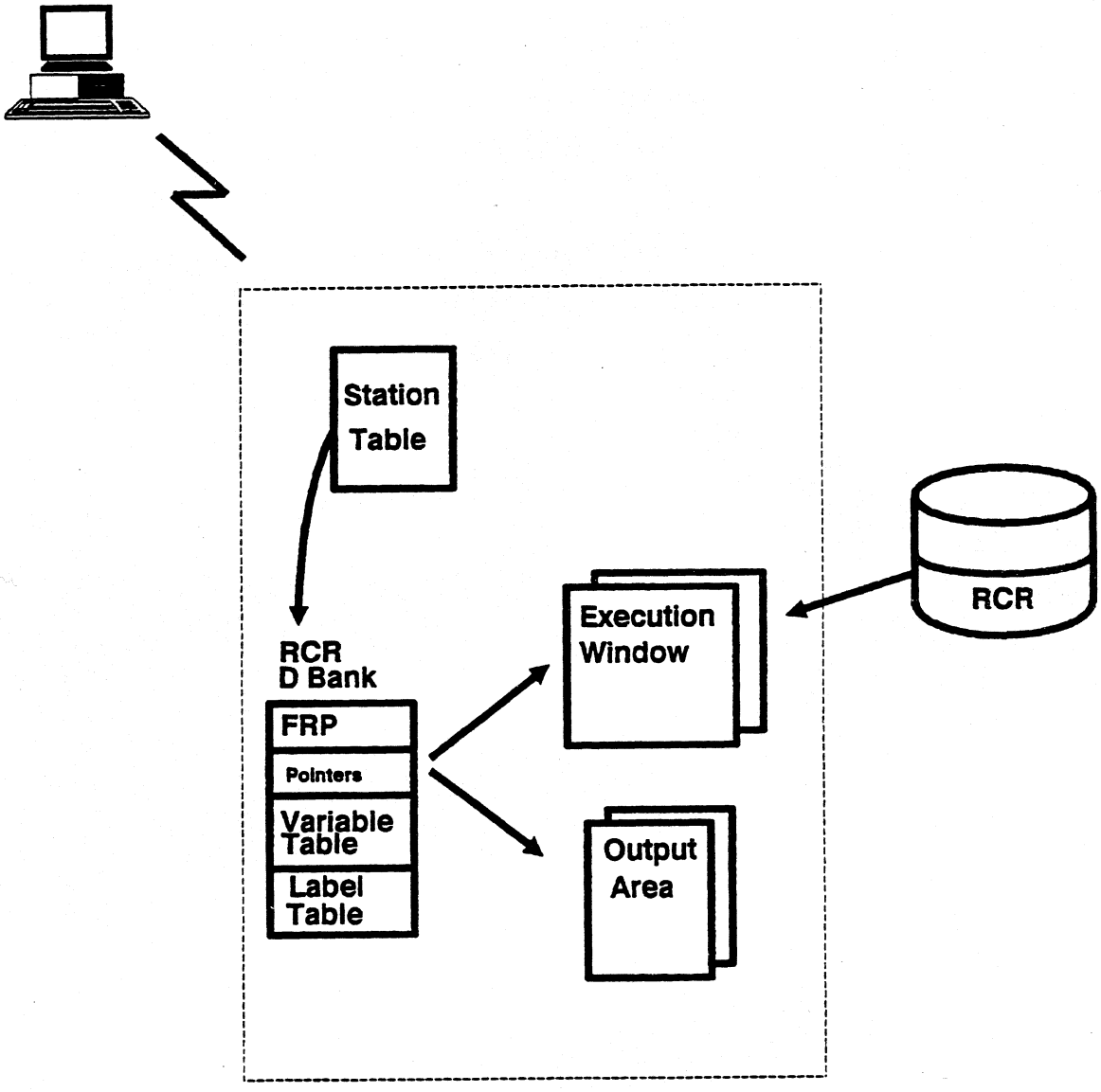
- **Memory is allocated in buffers**
- **Retrieval and storage always occurs in buffers**
- **There are two buffers the RCR needs**
  - **Input Buffer, referred to as the Execution Window, where a "page" of the RCR is kept.**
  - **Output Buffer where processed data will be kept, which we know as the Output Area**
- **MAPPER has to know where the control table and the buffers reside**
  - **Station Table points to the user's D Bank (Control Table).**
  - **D Bank, contains pointers to the RCR's Execution Window (Input Buffer) and the Output Area (Output Buffer).**
- **~~A~~ TWO "page~~s~~ of the RCR ~~is~~ are read (I/O) into the Execution Window**

Buffers split memory into shareable chunks

Processing is Top Down



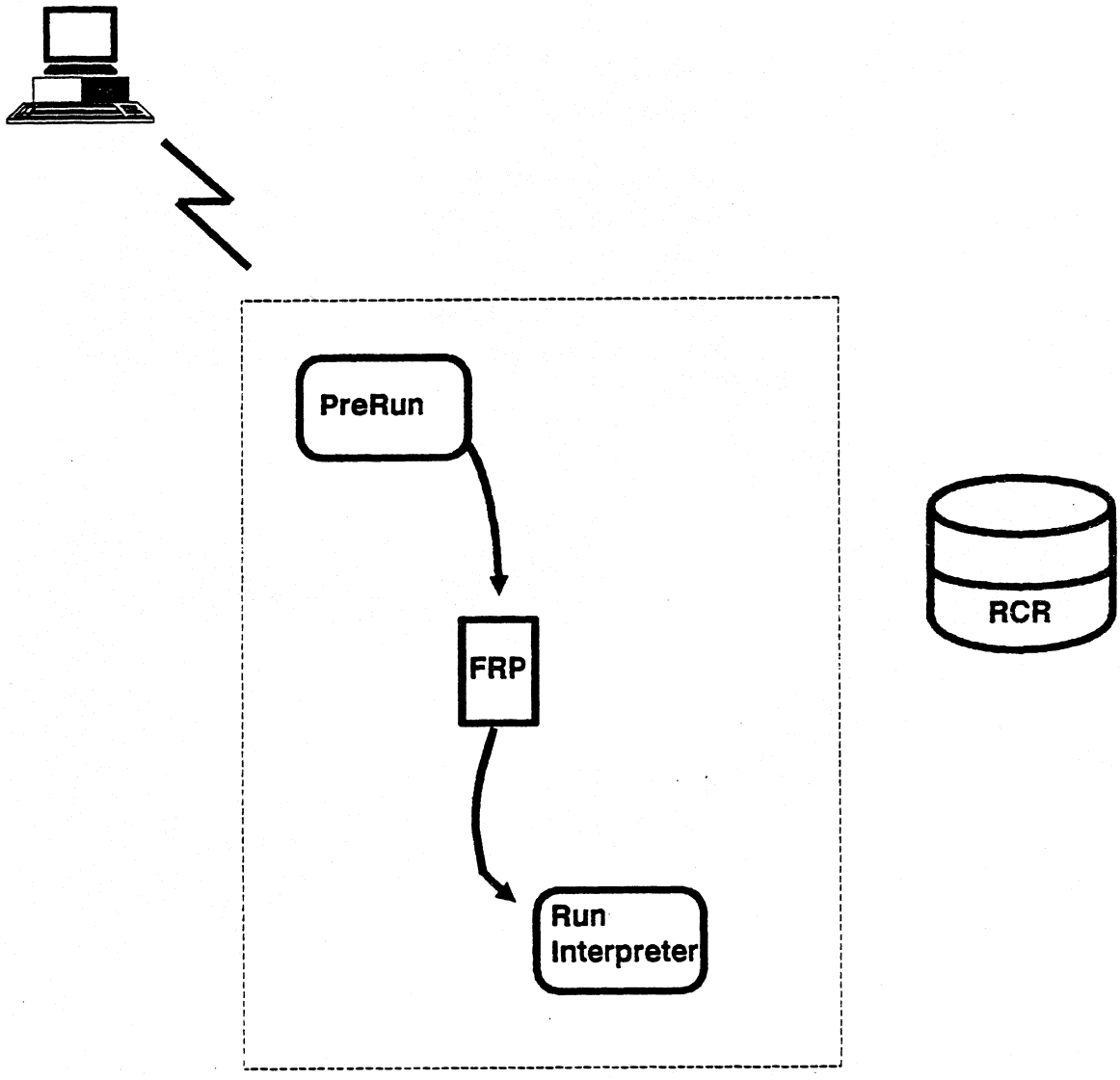
Acquiring Main Storage



PreRun

- Builds the Function Request Packet
- Calls another MAPPER routine, RPXFUN, referred to as the Run Interpreter.
- Passes the Function Request Packet to the Run Interpreter

PreRun

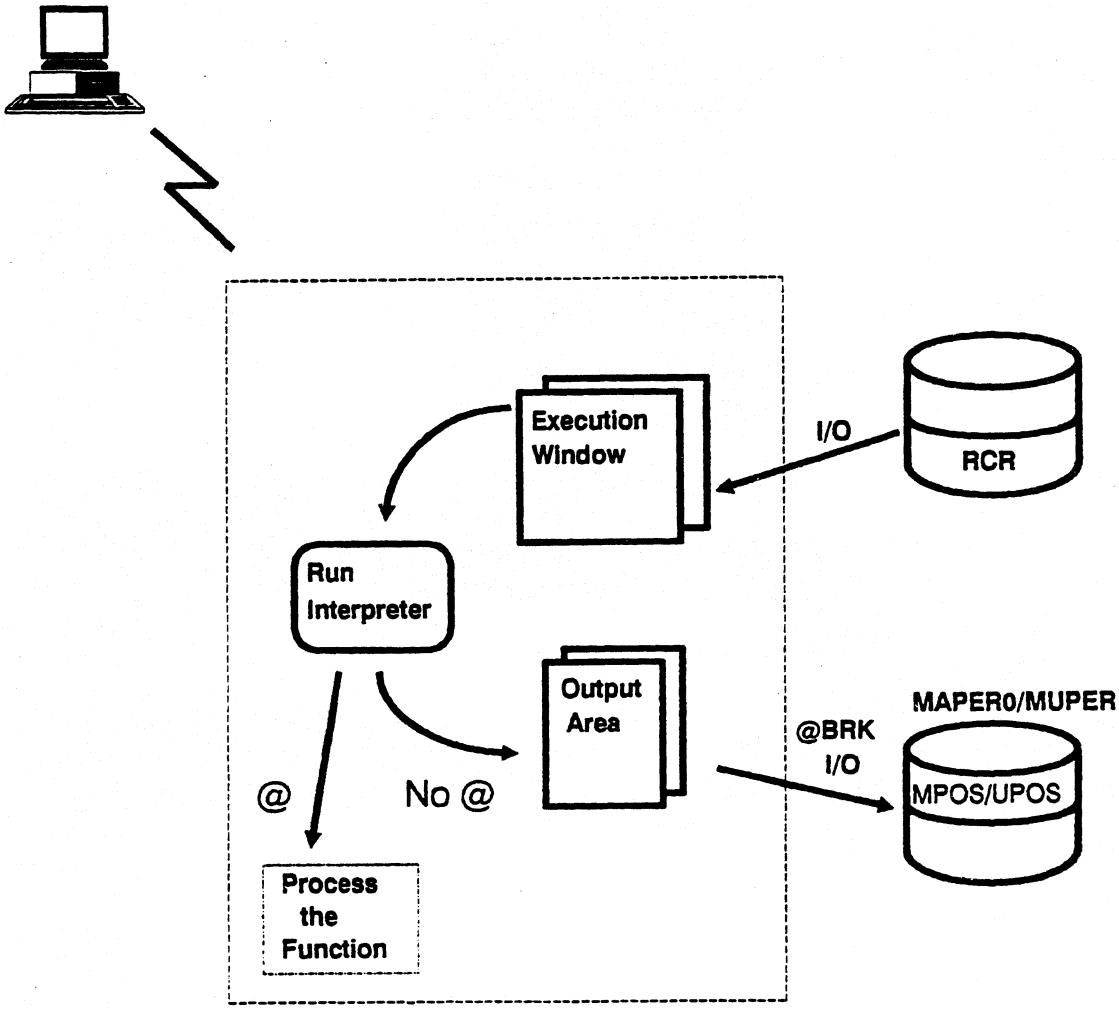


Run Interpreter

- Interprets the RCR's Execution Window.
- Starts at line 3 of the RCR and look in column one for an @ sign
  - If an @ sign exists in column one, it passes control to the appropriate run function for processing.
  - If no @ sign in column one, it places this line in the Output Area buffer and continues with RCR processing.

It's more efficient not to have headers in RCR's.

RCR Interpreters

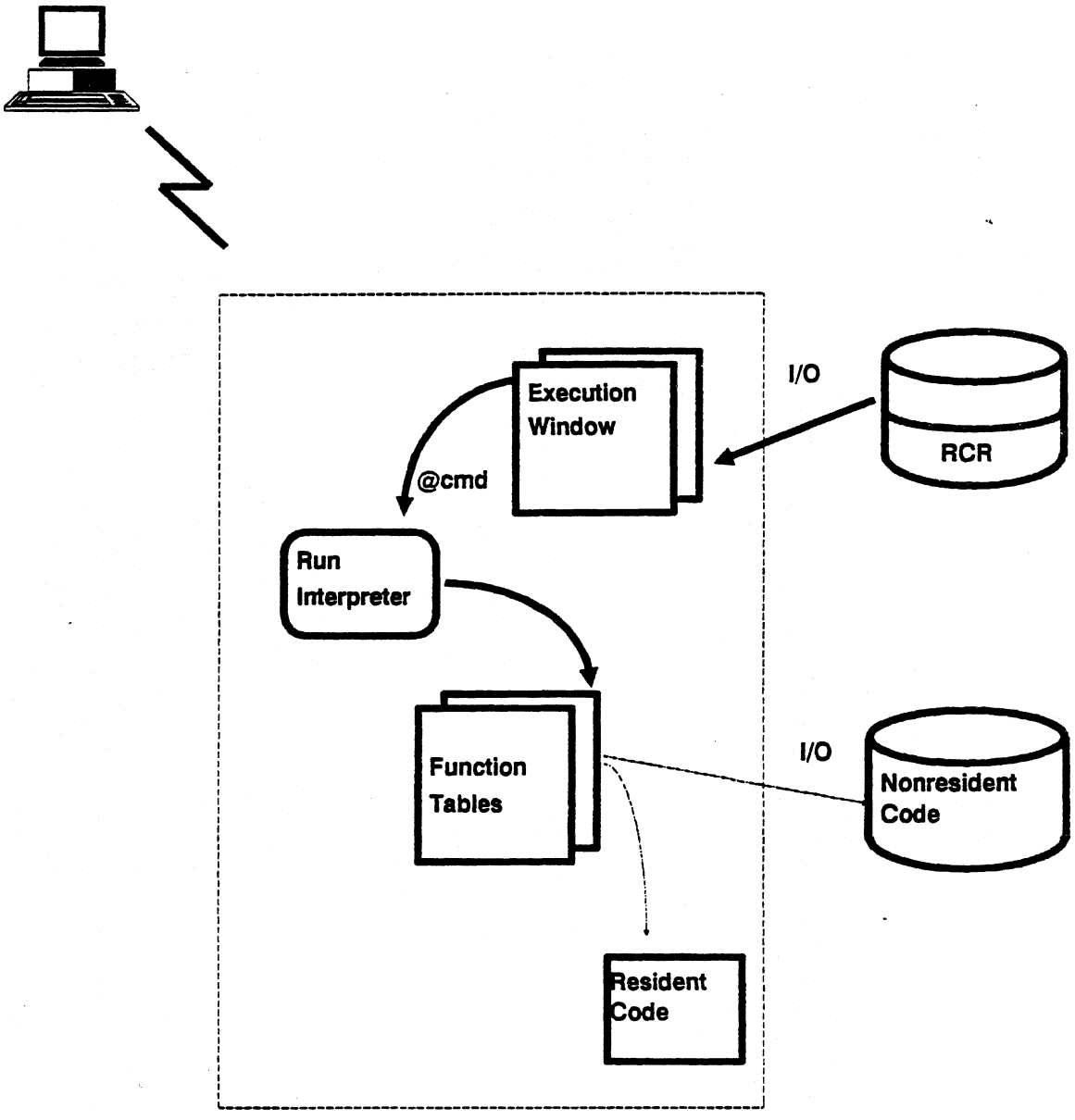


Function Processing

- **Step 1: Run Interpreter checks the Function Tables to determine**
  - Is it a valid function?
  - If it is valid, is it resident (currently in main storage) or nonresident (on disk)?
- **Step 2: The Function Table contains a flag indicating whether or not the function code is in main storage (Resident) or on disk storage (Nonresident)**
  - If the function is Resident, execute it.
  - If the function is Nonresident, check for available main storage space
    - If available, then bring in the function (I/O).
    - If not, then swap out an unused function and swap in the current function (I/O's).

Note: Nonresident functions accumulate "unseen" I/O's. MAPPER Coordination can provide you with a list of Resident vs Nonresident functions.

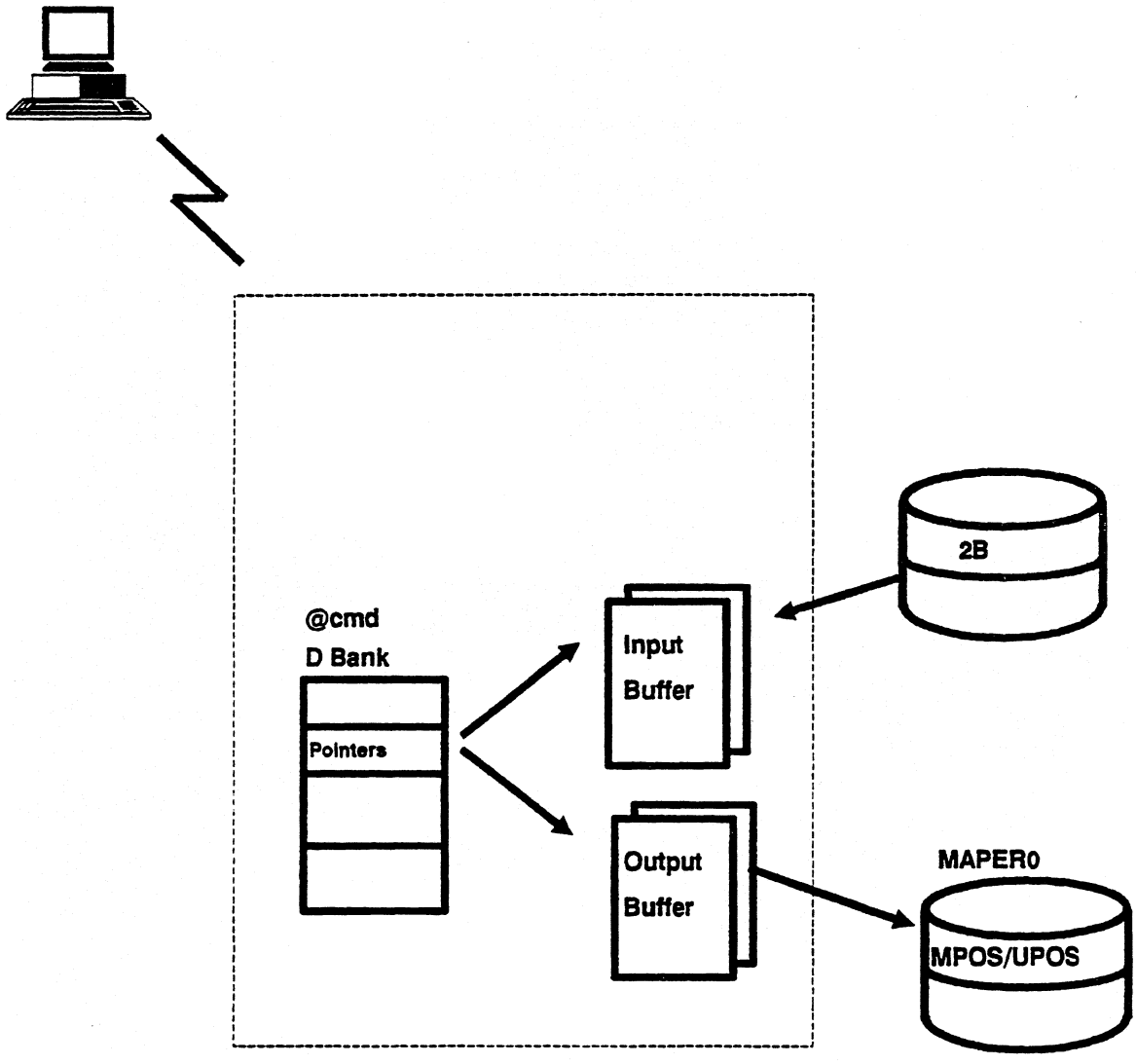
Function Processing



Function Processing

- **Function now has a D bank similar to the RCR D bank containing**
  - Function Request Packet
  - Pointers to buffers
    - Input Buffer which will contain a "page" of a report to be processed by this function
    - Output Buffer which will contain the result produced by this function

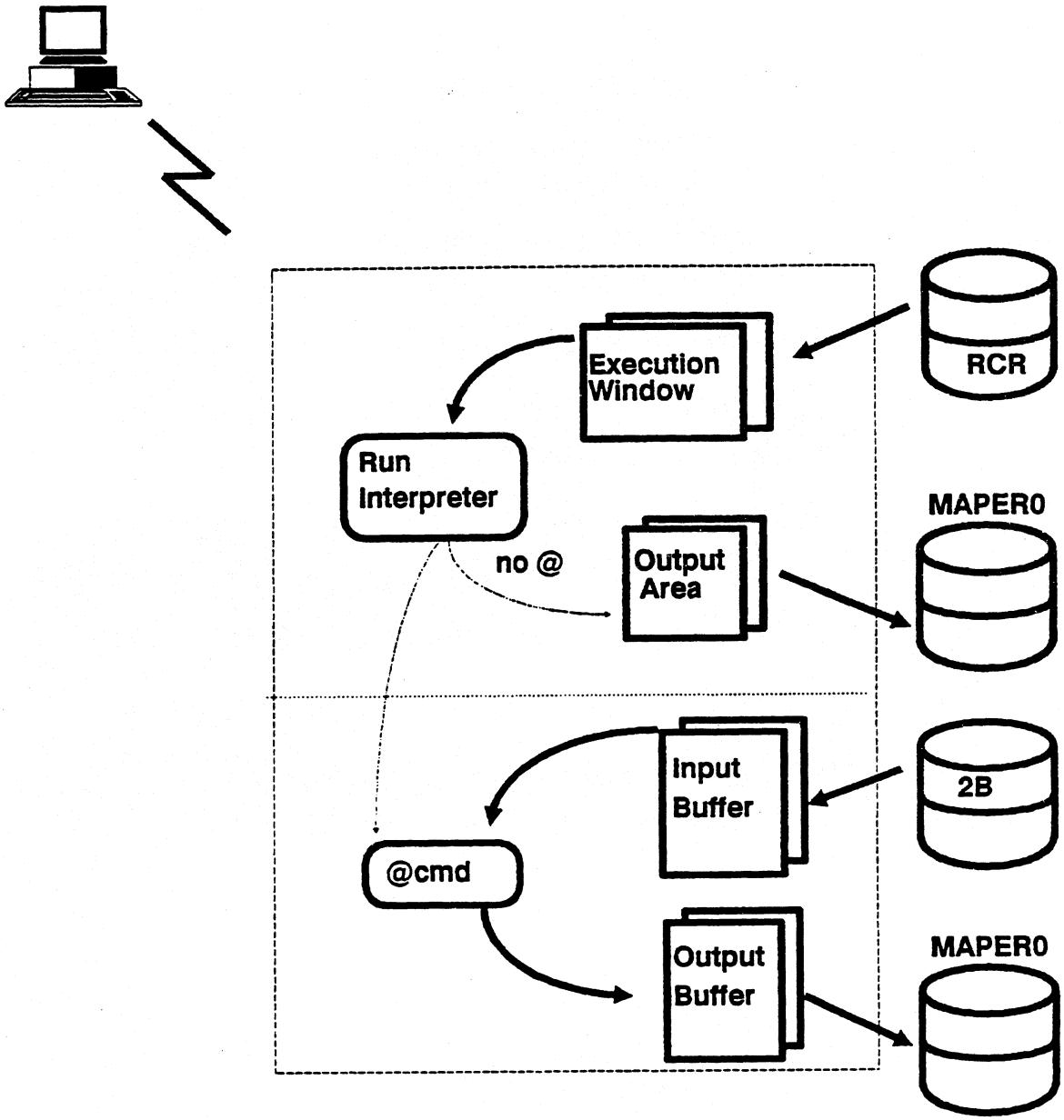
Function Processing



RCR vs Function Processing

- To process a report MAPPER must
  - Locate the report
    - Referencing the Type Table and the Rid Table
  - Read a "page" of data (I/O) into the Input Buffer.
  - Depending upon the run function that requested the report, lines of the report may be placed into its output buffer
  - Interprets each line, one "page" at a time, in the Execution Window until it encounters an End of Report marker.
  - Depending on what appears in column one
    - The line will get placed into the Output Area buffer and RCR processing will continue.
    - If an @ sign appears , Function processing will begin.

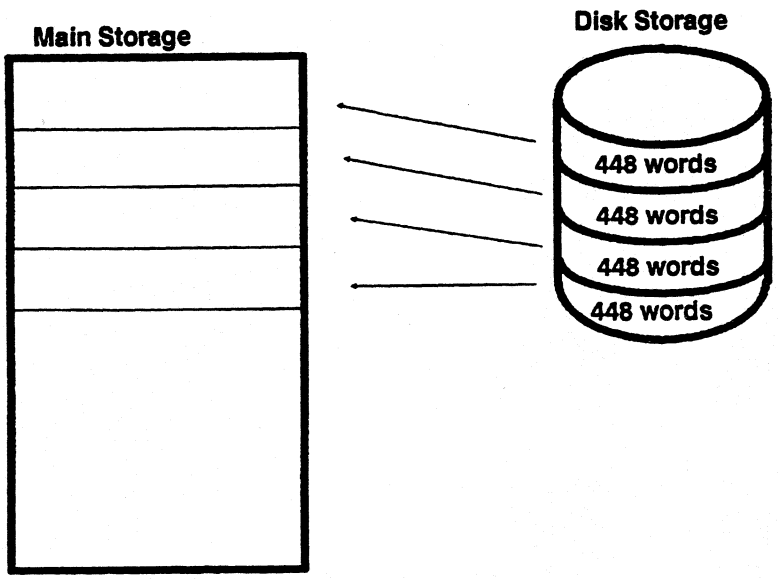
RCR vs Function Processing



Buffer Sizes

- Buffers are used to
  - Allocate space for the RCR report
  - Allocation of buffers for the RCR execution window, input and output area
  - Allocate space for the functions input and output buffers
- One buffer equals 448 words =1792 bytes
- To perform one I/O, a minimum of one 448-word buffer is required
- MAPPER's main storage is "formatted" in units of 448 words
- Larger buffers involve less I/O's
- Smaller buffers involve more I/O's
- Disk storage can be "formatted" or "prepped" in 112, 224, 448 or 1792 word buffers

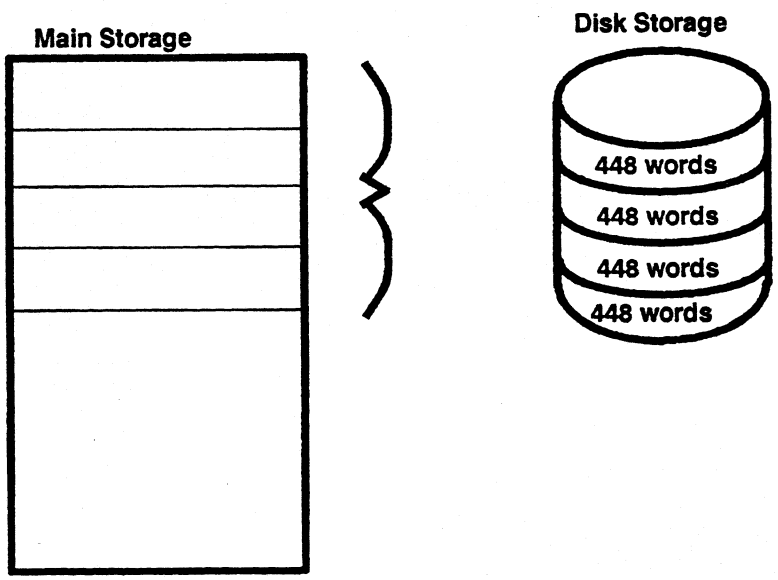
Buffer Sizes in Terms of Words



Seconds

4 I/O's =

.16 sec



Seconds

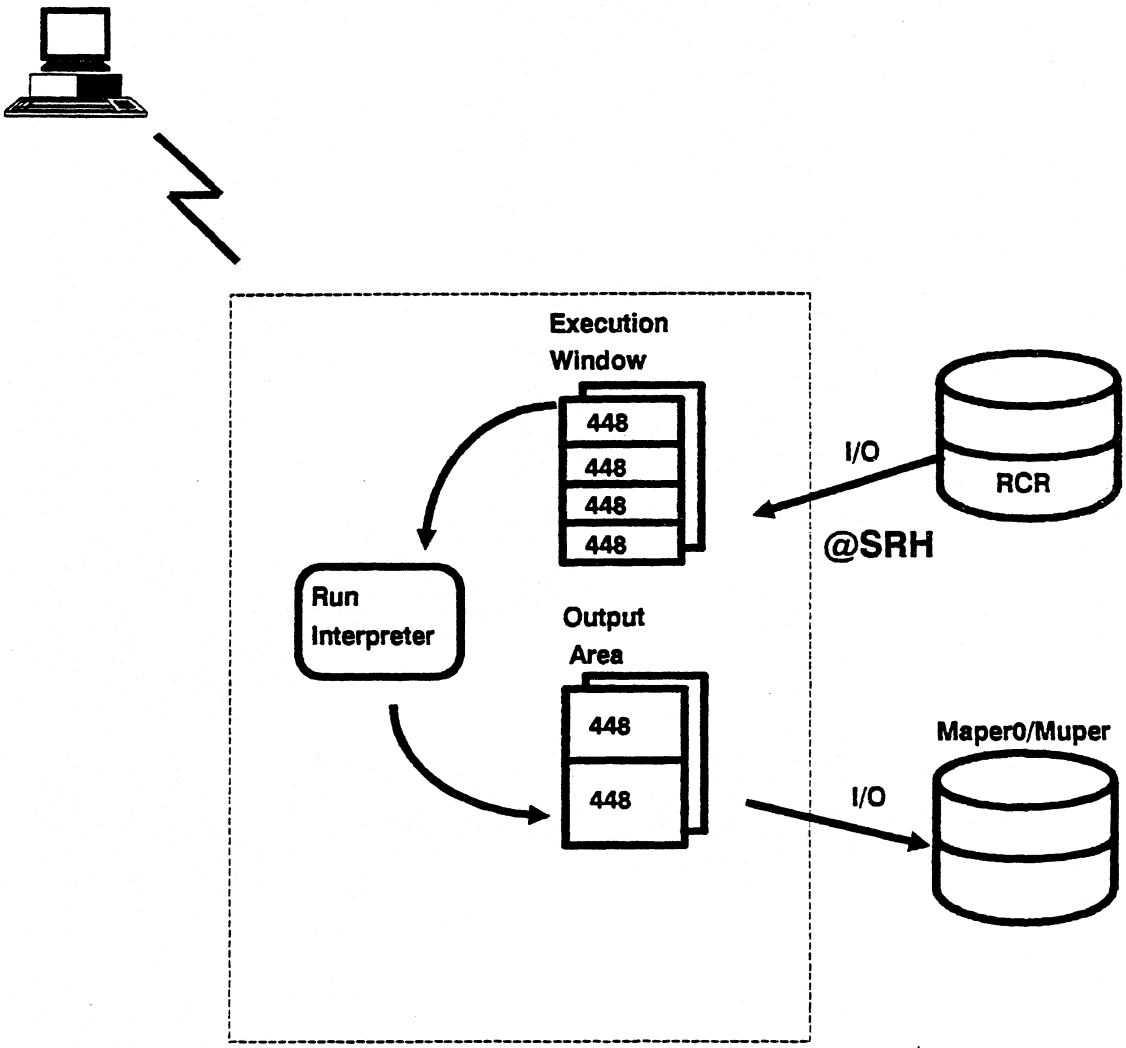
1 I/O =

.04 sec

Buffer Sizes

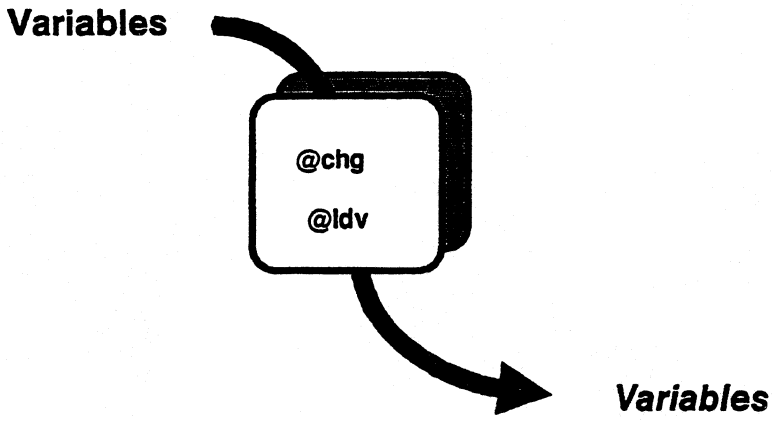
- Buffer sizes vary according to what the function is expected to do
- Output Area buffers less than or equal to 896 words remain in storage
- Data in output Area buffers more than 896 words are written to MAPER0/MUPERn by the Run Interpreter
- Results (-0 through -7) produced from run functions are allocated 1792 words and remain in main storage until their four unit buffers are full

Buffer Sizes



Functions Requiring No Buffers

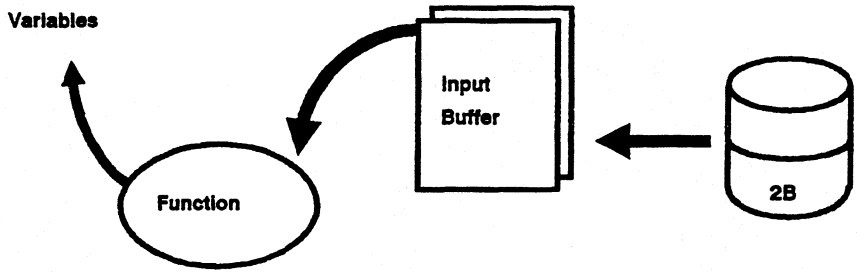
- **Functions that take information from variables, process the information, and return values in variables**




```
@chg v1 v2 + v3
@ldv,p v1,v2
```

Functions Requiring One Buffer

- **Functions that process reports and return values in variables**

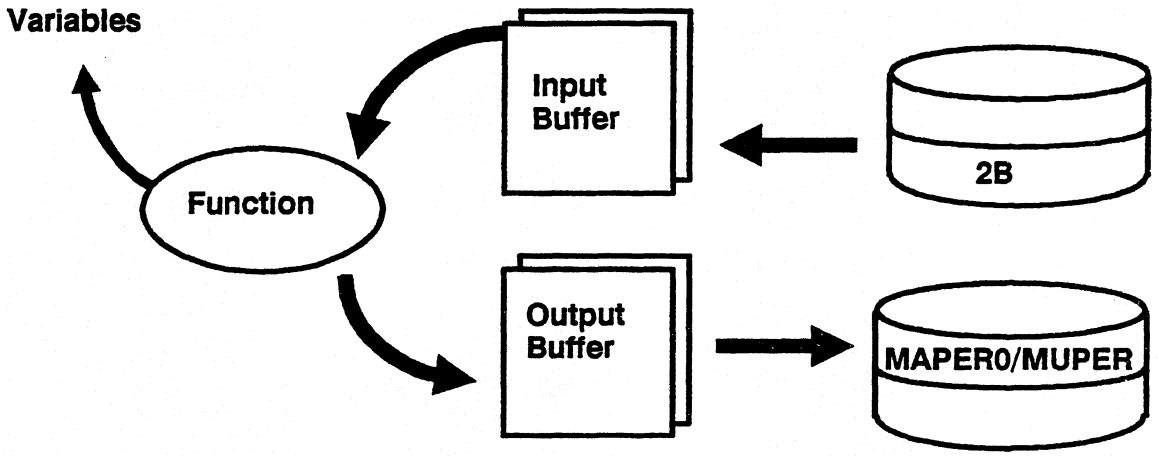



```
@lzr,0,b,2,10 <lines>i3,,,<char>a3,<upd>i3 .
        Lines in report - <lines>
        Character set - <char>
        Updates - <upd>
@gto end .
```

```
@brk .
@rdl,0,b,2,6  2-13 v1 i13
@1: if v1 (1-2) = xx gto end .
          v1
@rdl,0,b,2  2-13 v1
@gto 1 .
```

Functions Requiring Two Buffers

- **Functions that use both input and output buffers, as well as return values in variables**




```
@srh,0,b,10 d 'st cd'  ,or .
@dsp,-0 .
```

versus

```
@dup,0,b,2
@chg  <rid>i3 rid$ .
```

Lines Per I/O

- To calculate how many lines are transferred during an I/O you must consider
  - Character Set
  - Characters per Line
- Fieldata/Limited Character Set (LCS)
  - Upper case letters only, digits, and limited special characters
  - Fieldata reduces flexibility yet is more efficient.
  - One word holds six Fieldata characters.
- Ascii/Full Character Set (FCS)/Full Character Set Upper (FCSU)
  - Lower and/or upper case letters, digits, and all special characters
  - Ascii increases portability and flexibility yet is less efficient.
  - One word holds four Ascii characters.
- Fieldata allows more lines per I/O; Ascii allows less lines per I/O.
- Characters per line
  - Coordination has the capability to Gen from 40 through 256 character reports.
  - The number of characters per line will also affect the number of lines that are brought in with one I/O.

Lines Per I/O

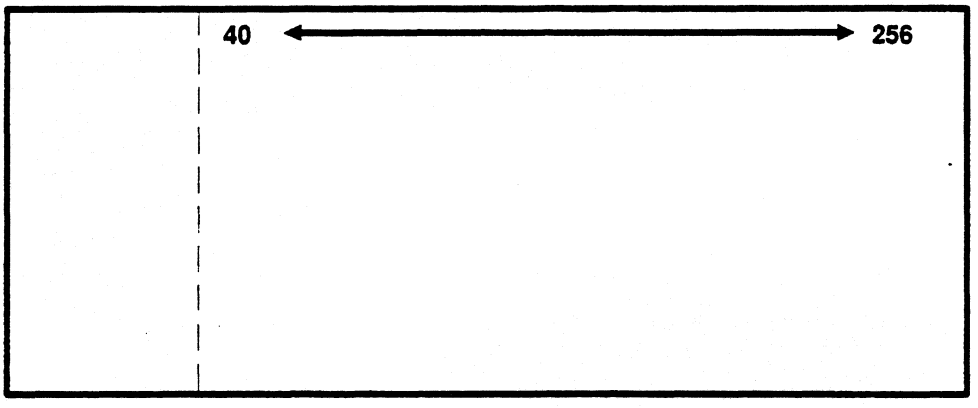
Six Fieldata Characters/Word - Reduces flexibility, yet is more efficient

A	B	C	D	E	F

Four Ascii Characters/Word - Increases portibility and flexibility, yet is less efficient

A	B	C	D

Characters/Line - The larger the report width, the fewer lines that may be transferred per I/O



Lines Per I/O

● Formula

Step One: $\dfrac{\text{(Characters per Line)}}{\text{(Characters per Word)}}$ = **Words per Line**

Step Two: **Words per Line + Control Words = Total Words per Line**

Step Three: $\dfrac{\text{(Words per Buffer)}}{\text{(Total Words per Line)}}$ = **Lines per I/O**

● Sample calculation

$$\frac{\text{(80 characters per line)}}{\text{(4 characters per word ASCII)}} = \text{20 words per line}$$

20 words per line + 3 control words = 23 total words per line

$$\frac{\text{1792 words per buffer}}{\text{23 words per line}} = \text{77 lines per I/O}$$

Note: Control words have a special function on the 1100. Generally there are three control words added: One control word for the 1100, one control word for internal line numbering and one control word for carriage return. In essence, these control words force fixed line length in 1100 Program Files. The only exception to the rule is LCS, 80 character reports. Only two control words are added.

I/O Buffer Requirements

multiple memory pools

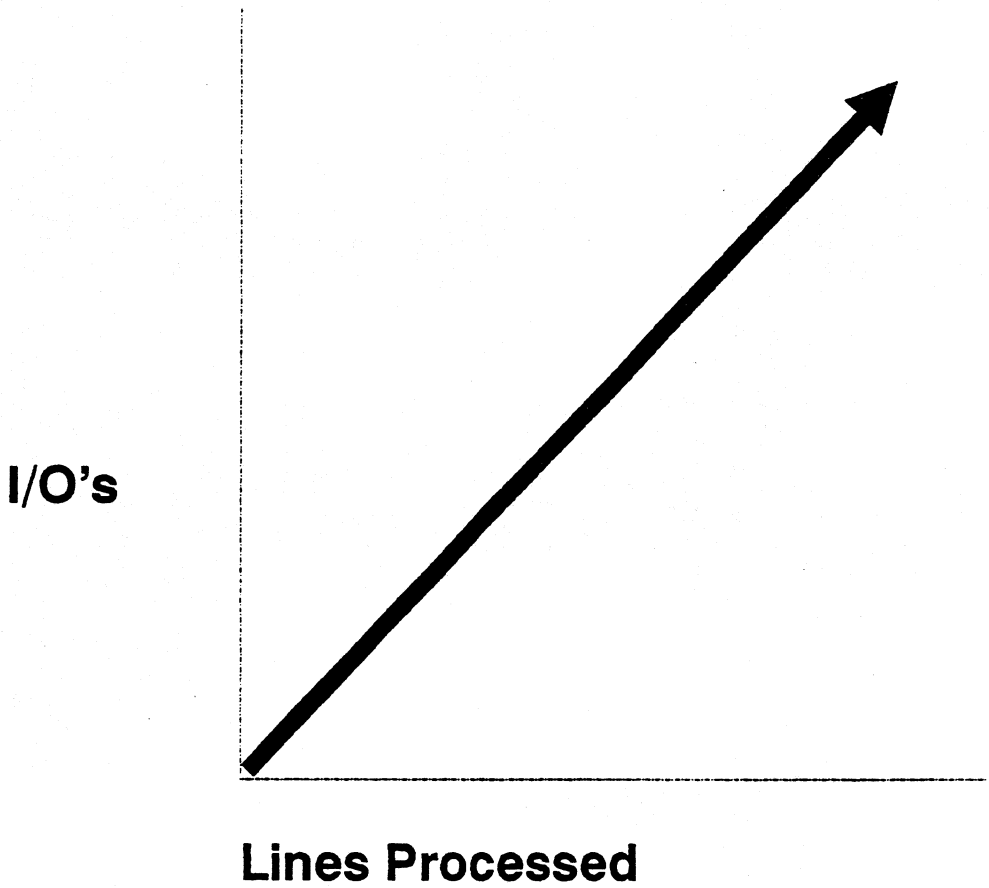
FUNCTION	READ	WRITE	ALT.	If MMPBNK > 0 READ	WRITE	ALT.
ADD	448*4	448*2		448*4	448*2	
APPEND	448*2	448*2		448*4	448*4	
ART	448*1	448*1		448*1	448*1	
AUX	448*1	448*2		448*4	448*4	
BATCH START	448*2	448*2		448*4	448*4	
BFN	448*4	448*2		448*4	448*2	
CALCULATE	448*1	448*1		448*4	448*4	
COMPARE	448*1	448*1		448*2	448*2	
CUT/PASTE	448*2	448*2		448*4	448*4	
DATE	448*2	448*2		448*2	448*2	
DELETE	448*1			448*1		
DISPLAY	448*2			448*2		
DUPLICATE	448*1	448*1		448*4	448*4	
ELT	448*2	448*2		448*4	448*4	
FIND	448*2			448*4		
INDEX		448*2			448*2	
LINE-CHANGE	448*2	448*2		448*4	448*4	
LINE ZERO	448*1			448*1		
LIST-MERGE	448*2	448*2		448*4	448*4	
LOCATE	448*2			448*4		
MATCH	448*1	448*1		448*4	448*4	
MESSAGE-STACKER	448*2	448*2		448*2	448*2	
PRINT	448*2			448*2		
REFORM	448*2	448*2	448*2	448*2	448*2	448*2
REPLACE REPORT	448*2	448*2		448*4	448*4	
RUN (RCR)	448*1	448*1		448*4	448*2	
(RDL)	448*4			448*4		
(RDC)	448*4			448*4		
(LSM)	448*1			448*1		
SEARCH	448*2	448*1		448*4	448*2	
SEARCH UPDATE	448*1	448*2	448*1	448*4	448*4	448*1
SHORT-SORT	448*2			448*4		
TOTAL	448*1	448*1		448*4	448*4	
TYPE GEN	448*1	448*1		448*1	448*1	
UPDATE	448*2	448*2		448*4	448*4	
WORD LOCATE/CHANGE	448*2	448*2		448*4	448*4	
WPS	448*2	448*2		448*4	448*4	
*COUNT	448*8	448*8		448*8	448*8	

***Count is unlike any other function, output I/Os will vary depending on how many unique key fields there are. Internal code is written different from all other commands.**

Function Power Curve

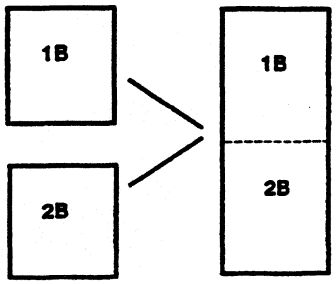
- Larger buffers reduce I/O's
- Buffer size considerations include Character Set and Characters per Line
- A relationship exists between I/O's and the number of lines processed

This relationship can be depicted by a graph. Typically, as the number of lines to process increases, the number of I/O's increases. This graph is called the Function Power Curve:



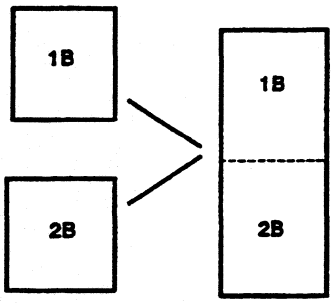
Append vs RDC

APPEND



	Lines	I/O's
Read	201	4
Write	197	3
		7 I/O's

RDC



	Lines	I/O's
Read	203	6
Write	199	5
		11 I/O's

Note: RCR included in Read I/O count

Point: Buffer sizes will affect the outcome of a command

```
LINE▶ 1      FMT▶    RL▶ -      SHFT▶      H
.DATE 30 JUL 90  10:08:09  RID       9E
.RUN FUNCTION DATA:
*=================================
@LOG .
@ADD,0,B,14,0,B,15 .
@DSP,-0 .
                        ..... END REPORT ..
```

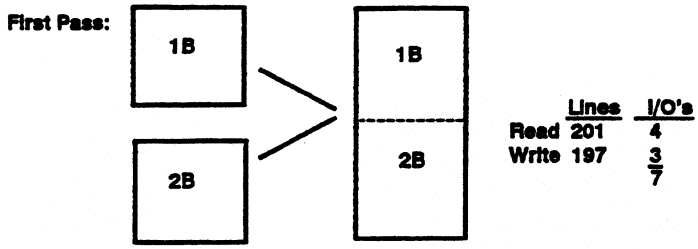
```
LINE▶ 1      FMT▶    RL▶ -      SHFT▶      H
.DATE 09 AUG 90  09:14:29  RID      14E
.RUN FUNCTION DATA:
*=================================
@LOG .
@BRK .
@RDC,0,B,14,2 1-80 U1S80 .
U1
@RDC,0,B,15,6 1-80 U1S80 .
U1
@GTO END .
                        ..... END REPORT ..
```

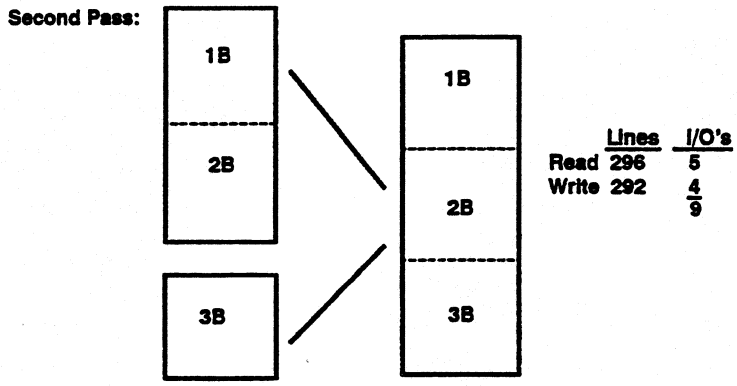
Append vs RDC

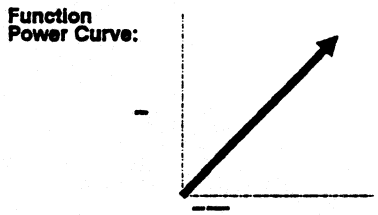
```
LINE▶ 2     FMT▶   RL▶ -     SHFT▶     HLD CHRS▶    HLD LN▶        ▶   61A      ▶
.LOG LIST: FROM 11:13:01  10 AUG 90   TO 11:15:11  10 AUG 90              H003356
*==========================================================================

                Functions        I/O          Logic Lines      Data Lines
               ============ ================ ================ ================
Part 1     =            3               10                5              409
Part 2     =            2                5                0               25
               ------------ ---------------- ---------------- ----------------
    Total =             5               15                5              434

Approximate charge for I/O processing  $    0.02

*   USER     .  FUNCTION  .   ACTIVE   . TYPE .RID .LINES-IN.LINE-OUT.READS.WRITE
*==========.============.============.======.====.========.========.=====.=====
 WEBMJG      PRE-RUN*            0.010 003330  100        4        0     1     0
 WEBMJG      APPEND*             0.141 000002   15      201      197     4     3
 WEBMJG      RUN*                0.011 000010    9        5        2     1     1
Part 1     : Funcs = 3      LLP = 5          Lines = 409       I/O = 10
 WEBMJG      DISPLAY*            0.021 000002               23        0     1     0
 WEBMJG      RUN*                0.015 000010    9        0        2     3     1
Part 2     : Funcs = 2      LLP = 0          Lines = 25        I/O = 5
 WEBMJG      DISPLAY*            0.031 000010                2        0     0     1
                        ..... END REPORT .....
1 Row:1  Col:7    90/AUG/10 14:35                  --
```

```
LINE▶ 1     FMT▶   RL▶ -     SHFT▶     HLD CHRS▶    HLD LN▶        ▶   63A      ▶
.DATE            11:21:43  RID     63A   10 AUG 90  WEBMJG
.LOG LIST: FROM 11:18:54  10 AUG 90   TO 11:21:21  10 AUG 90              H003356
*==========================================================================

                Functions        I/O          Logic Lines      Data Lines
               ============ ================ ================ ================
Part 1     =            2               12              203              406
               ------------ ---------------- ---------------- ----------------
    Total =             2               12              203              406

Approximate charge for I/O processing  $    0.01

*   USER     .  FUNCTION  .   ACTIVE   . TYPE .RID .LINES-IN.LINE-OUT.READS.WRITE
*==========.============.============.======.====.========.========.=====.=====
 WEBMJG      PRE-RUN*            0.009 003330  100        4        0     1     0
 WEBMJG      RUN*                0.354 000010   14      203      199     6     5
Part 1     : Funcs = 2      LLP = 203        Lines = 406       I/O = 12
 WEBMJG      DISPLAY*            0.023 000010               23        0     1     0
                        ..... END REPORT .....

1 Row:1  Col:7    90/AUG/10 14:37                  --                        Poll
```

Append





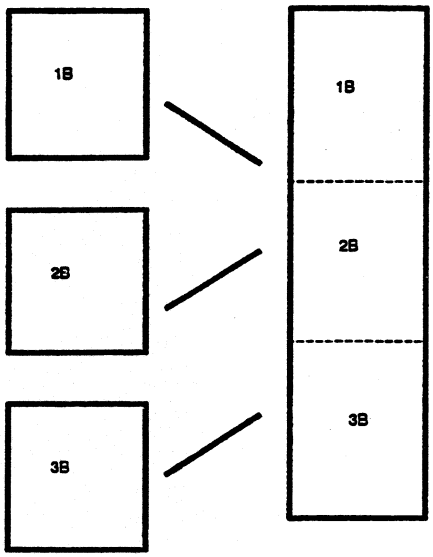



```
LINE▶ 1      FMT▶    RL▶ -      SHFT▶      HLD CHRS▶    HLD LN▶        ▶   13E0      ▶
.DATE 27 JUL 90  13:39:13  RID      13E   27 JUL 90  WEBMJG
.RUN FUNCTION DATA:                                   BY:                      E000010
*=================================================================================
@LOG .
@ADD,0,B,14,0,B,15 .
@ADD,-0,0,B,16 .
@DSP,-0 .
                    ..... END REPORT .....
```

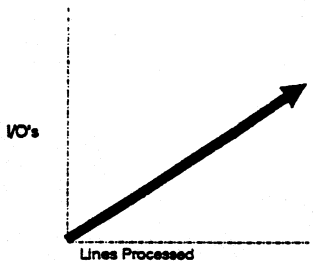
Append

```
LINE▶ 2     FMT▶   RL▶       SHFT▶     HLD CHRS▶    HLD LN▶       ▶   62A      ▶
.LOG LIST: FROM 11:16:02  10 AUG 90   TO 11:17:51  10 AUG 90                H003356
*==========================================================================

                Functions        I/O          Logic Lines      Data Lines
               =========== ================ ================ ================
Part 1     =            4               19                6              999
Part 2     =            2                5                0               25
               ----------- ---------------- ---------------- ----------------
    Total =             6               24                6            1,024

Approximate charge for I/O processing  $    0.03

*   USER     .  FUNCTION  .   ACTIVE   . TYPE .RID .LINES-IN.LINE-OUT.READS.WRITE
*==========.============.============.======.====.========.========.=====.=====
 WEBMJG       PRE-RUN*            0.009 003330  100        5        0     1     0
 WEBMJG       APPEND*             0.134 000002   15      201      197     4     3
 WEBMJG       APPEND*             0.174 000002   16      296      292     5     4
 WEBMJG       RUN*                0.040 000010   13        6        2     1     1
Part 1     : Funcs = 4       LLP = 6          Lines = 999       I/O = 19
 WEBMJG       DISPLAY*            0.017 000002              23        0     1     0
 WEBMJG       RUN*                0.017 000010   13         0        2     3     1
Part 2     : Funcs = 2       LLP = 0          Lines = 25        I/O = 5
 WEBMJG       DISPLAY*            0.009 000010               2        0     0     1
1 Row:1  Col:7    90/AUG/10 14:37                                                Poll
```

RDC



	Lines	I/Os
Read	300	8
Write	294	7
		15




```
LINE▶ 1      FMT▶    RL▶ -      SHFT▶      HLD CHRS▶     HLD LN▶          ▶   16E0      ▶
.DATE 10 AUG 90  10:00:06  RID      16E    27 JUL 90  WEBMJG
.RUN FUNCTION DATA:                                   BY:                        E000010
*=================================================================================
@LOG .
@BRK .
@RDC,0,B,14,2 1-80 V1S80 .
V1
@RDC,0,B,15,6 1-80 V1S80 .
V1
@RDC,0,B,16,6 1-80 V1S80 .
V1
@GTO END .
                    ..... END REPORT .....
```

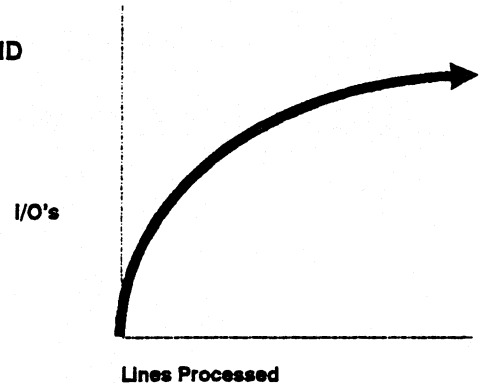
RDC

```
LINE▶ 1      FMT▶   RL▶ -     SHFT▶     HLD CHRS▶    HLD LN▶        ▶   64A      ▶
.DATE            11:27:08  RID      64A    10 AUG 90  WEBMJG
.LOG LIST: FROM 11:24:32  10 AUG 90   TO 11:26:52  10 AUG 90              H003356
*==============================================================================

              Functions        I/O          Logic Lines      Data Lines
             ============ ================ ================ ================
Part 1    =            2               16              300              599
             ------------ ---------------- ---------------- ----------------
    Total =            2               16              300              599

Approximate charge for I/O processing  $    0.02
.
*   USER     .  FUNCTION  .   ACTIVE   . TYPE .RID .LINES-IN.LINE-OUT.READS.WRITE
*==========.============.============.======.====.========.========.=====.=====
 WEBMJG      PRE-RUN*            0.009 003330  100        5        0     1     0
 WEBMJG      RUN*                0.447 000010   16      300      294     8     7
Part 1    ! Funcs = 2      LLP = 300       Lines = 599        I/O = 16
 WEBMJG      DISPLAY*            0.031 000010              23        0     1     0
                      ..... END REPORT .....


1 Row:1  Col:7    90/AUG/10 14:38
```

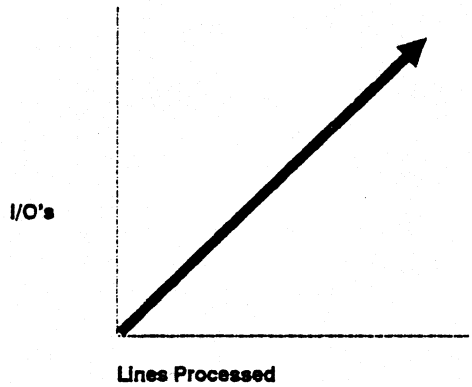
Function Power Curve:BFN versus FND

- **Use FND function if processing requirement involves 400 lines or less**

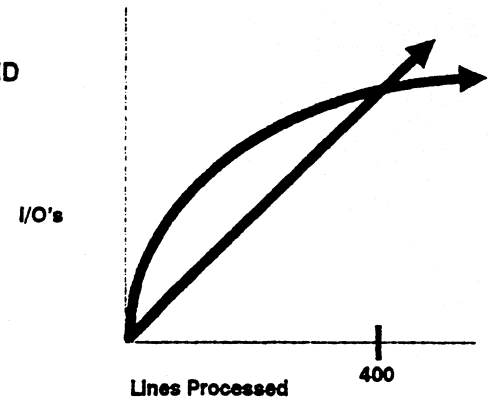
BINARY FIND



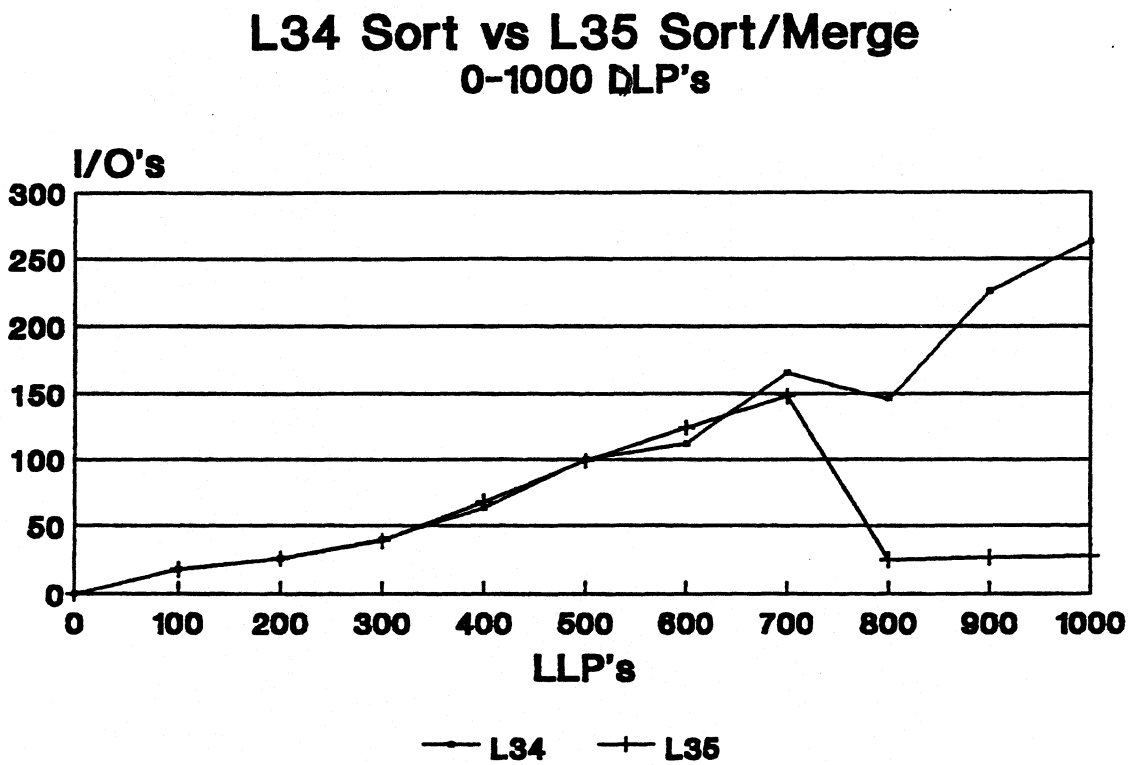
FIND



BFN VS FND



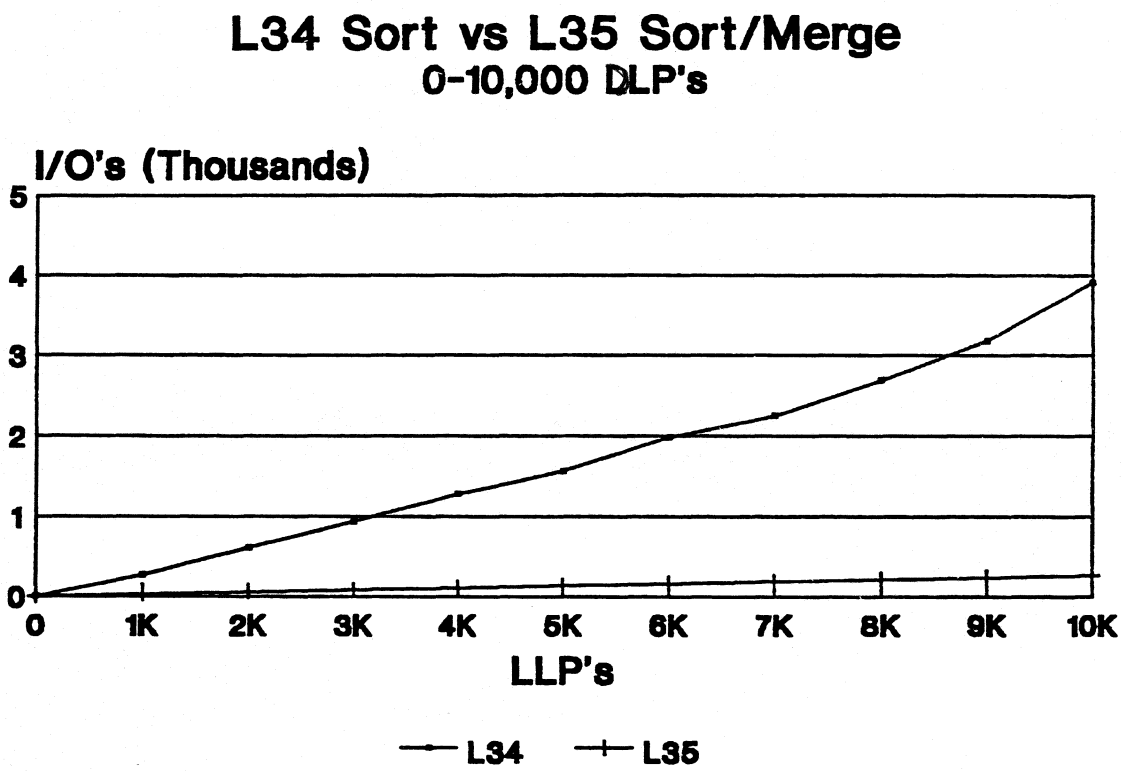
Function Power Curve: SOR



Level 35 External Sort
Threshold parameter (EXTSRT)
Default = 750

Old 'sort' used Bubble algorithm

Function Power Curve: SOR



Level 35 External Sort
Threshold parameter (EXTSRT)
Default = 750

There is an option to force MAPPER to use either bubble or new 'sort'.

Exercise

1. Match the following items:

Control Table	a. RCRs input buffer
D Bank	b. In main storage
a Execution Window	c. RCR D Bank
h Output Area	d. FRP, pointers, variable and label tables
b Resident Code	e. Units of 448
g Nonresident Code	f. I/Os to lines processed
e Buffers	g. Disk storage
f Function Power Curve	h. RCRs output buffer

2. What is Run Interpreter's purpose and how does it work?

3. Buffer requirements vary according to the function's goal. A function may require _____ buffer(s), _____ buffer(s), or _____ buffer(s), depending on the tools the function needs in its execution.

4. Calculate the following chart for the number of lines that fit in the Execution Window based on the character set and number of characters per line:

	40	80	132	256
FCS/ FCSU		77 lines		
LCS				

3

Efficiency and Analysis

Module 3
Efficiency and Analysis

Objectives

Upon completion of this module you should be able to:

1. Define and distinguish the difference between I/Os, LLPs ,DLPs.
2. Interpret LOG, LOGLA and RUNA results.

The Term "Efficiency"

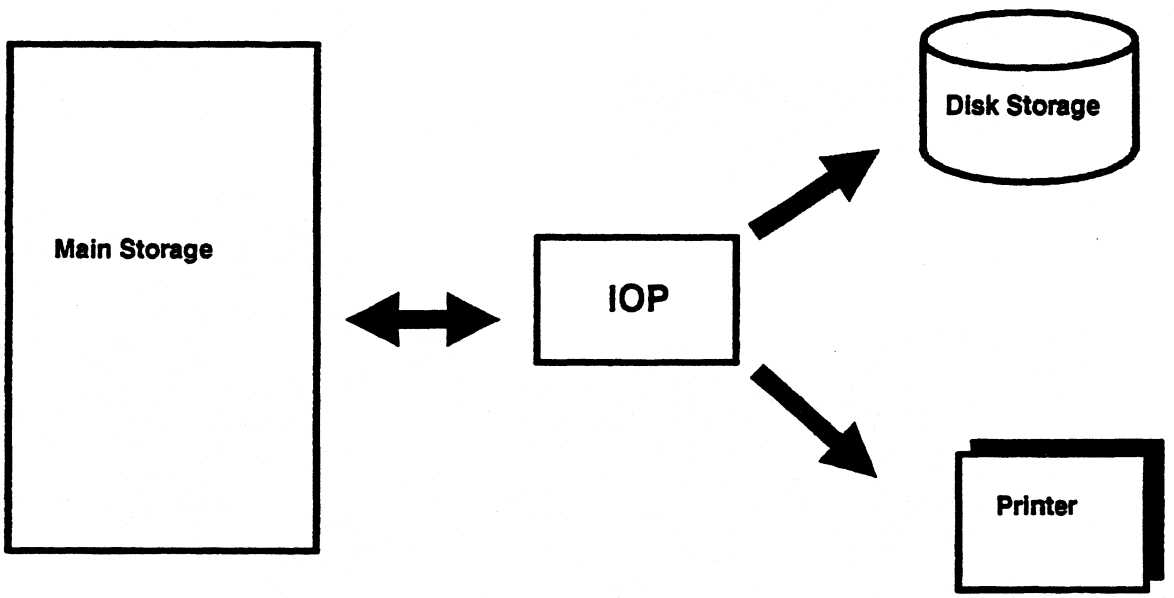
- Run Designer- Maintainability and portability
- MAPPER-Low I/O's and low lines processed

E = Maintainability

E = I/Os + LLPs

I/Os

- Each action of retrieval or transfer creates an I/O.
- Reserved word IO$ captures the I/O count.



I/O's

- Each run must have an I/O limit registered
- The I/O limit should not exceed 1000 to prevent the run from falling into an infinite loop

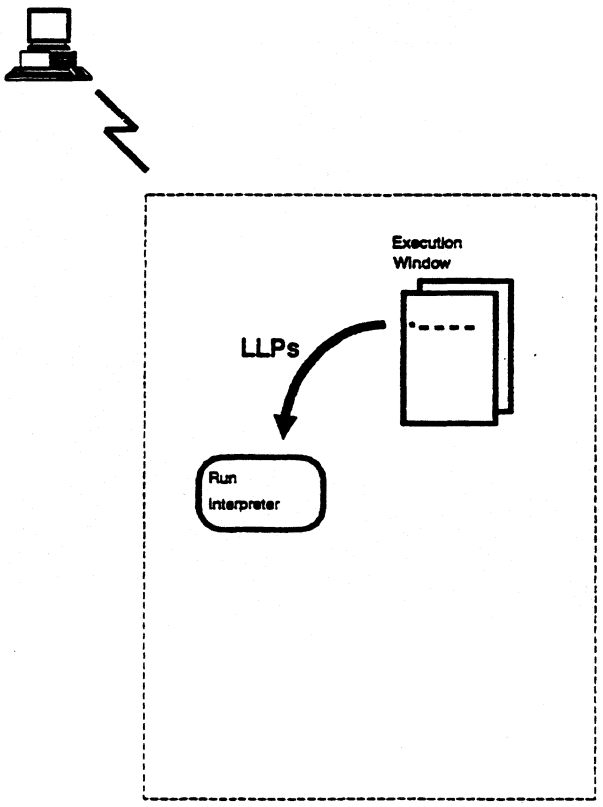
```
LINE▶ 1     FMT▶   RL▶ -     SHFT▶     HLD CHRS▶    HLD LN▶       ▶   45E218   ▶
.DATE 20 JUL 90  09:47:43  RID     45E   20 JUL 90  WEBMJG
.RUN CONTROL FOR :                                                      E003330
.                     *****KEEP RID SORTED BY RUN NAME*****
* RUNID    B.   USER    P. UNIT. TYPE . RID.F. B TIME . E TIME . I/O .LINES.MOD
*============.============.=====.======.====.=.========.========.=====.=====.===
 XYZ          JLD                                                 1000  2000
```

..... END REPORT .....

```
1 Row:1  Col:7    90/JUL/24 13:17
```

LLPs

- Logic Lines Processed are any RCR lines that are read/processed by the Run Interpreter, including blank lines, comment lines, and continuation lines
- Processing is between line 3 and end of report
- LLP$ captures the Logic Lines Processed.



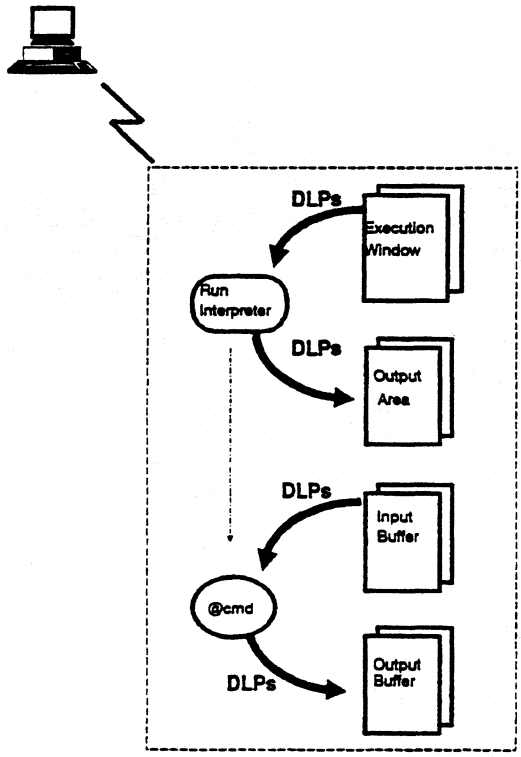
LLPs

- Each run must have an LLP limit registered
- The LLP count should not exceed 2000 during design phase

```
LINE▶ 1     FMT▶   RL▶ -     SHFT▶     HLD CHRS▶    HLD LN▶       ▶   45E218   ▶
.DATE 20 JUL 90  09:47:43  RID     45E   20 JUL 90  WEBMJG
.RUN CONTROL FOR :                                                      E003330
.                     *****KEEP RID SORTED BY RUN NAME*****
*  RUNID    B.   USER    P. UNIT. TYPE . RID.F. B TIME . E TIME . I/O .LINES.MOD
*===========,===========,=====,======,====,=,=======,=======,=====,======,===
 XYZ         JLD                                                  1000  2000


                    ..... END REPORT .....


1 Row:1  Col:7    90/JUL/24 13:20                                          Poll
```

DLPs

- Data Lines Processed are any lines read/processed by the Run Interpreter, including results, reports processed and the run control report itself
- The system charges twice on LLPs, once as LLPs and again as DLPs
- DLP$ captures the number of Data Lines Processed



I/Os, LLPs, and DLPs

The following example illustrates the differences between I/Os, LLPs, and DLPs through usage of reserved words.
Note: When using reserved words to capture LLP and DLP values, the values are cumulative until a function resets them to 0. Reserved words take only a snapshot of what the value is at the time value us requested.

RCR

```
LINE▶ 1      FMT▶   RL▶ -      SHFT▶      HLD CHRS▶    HLD LN▶        ▶   8E0      ▶
.DATE 20 JUL 90  10:02:06  RID       8E   20 JUL 90  WEBMJG
.RUN FUNCTION DATA:                                   BY:                     E000010
*==========================================================================
@SRH,0,B,12 D 2-2   ,SH .
@LDV,W  V1I3=IO$,V2I3=LLP$,V3I3=DLP$.
     IO$=V1      LLP$=V2      DLP$=V3
@GTO END.
                        ..... END REPORT .....
```

Source Report

```
LINE▶ 1      FMT▶   RL▶ -      SHFT▶      HLD CHRS▶    HLD LN▶        ▶   12B0     ▶
.DATE 20 JUL 90  10:01:48  RID      12B   20 JUL 90  WEBMJG
.abc box factory                                      Corporate Production     B000002
*St.Status.By. Product .Serial.Produc.Order.Cust.Produc.Produc. Ship .Ship .Spc.
*Cd. Date .In.  Type   .Number. Cost .Numbr.Code. Plan .Actual. Date .Order.Cod.
*==.======.==.========.======.======.=====.====.======.======.======.=====.===.
 IP 831224 LS BLACKBOX1 436767        84389 AMCO 831223 831224
 IP 831225 LS BLACKBOX1 436768        84390 AMCO 831223 831225
 IP 831219 LS BLACKBOX2 637071        84353 INTR 831218 831219
 OR 840110 LS BLACKBOX4               94754 ARCO
 SH 840110 LS BLACKBOX5 675281        97441 FEDS 840131
                        ..... END REPORT .....
```

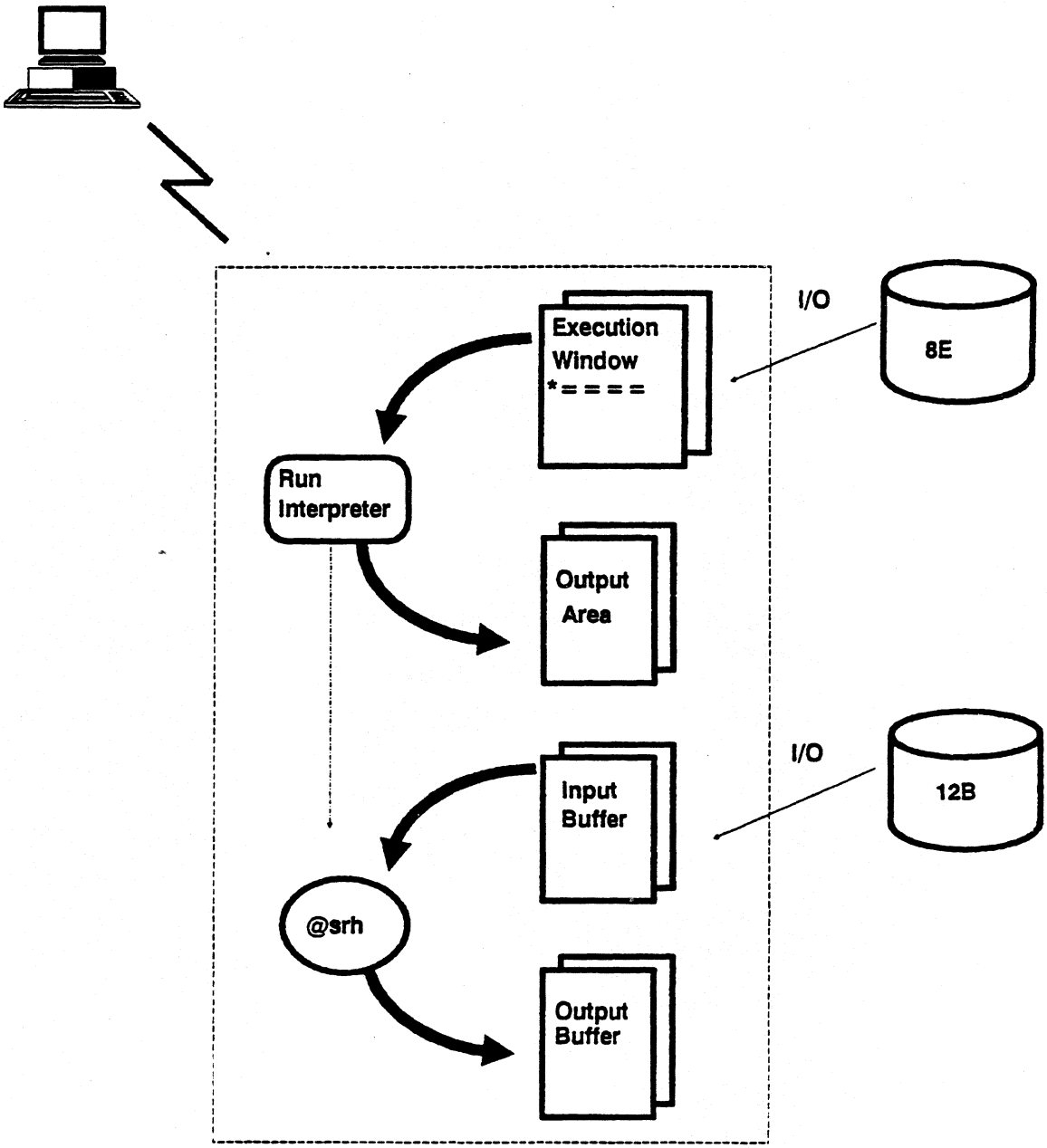
Search Result

```
line▶ 1      fmt▶   rl▶ -      shft▶      hld chrs▶    hld ln▶        ▶   RESULT   ▶
.DATE 20 JUL 90  10:01:48  RID      12B   20 JUL 90  WEBMJG
.abc box factory                                      Corporate Production     B000002
*St.Status.By. Product .Serial.Produc.Order.Cust.Produc.Produc. Ship .Ship .Spc.
*Cd. Date .In.  Type   .Number. Cost .Numbr.Code. Plan .Actual. Date .Order.Cod.
*==.======.==.========.======.======.=====.====.======.======.======.=====.===.
 SH 840110 LS BLACKBOX5 675281        97441 FEDS 840131
                        ..... END REPORT .....
```

Output Area

```
line▶ 1      fmt▶   rl▶ -      shft▶      hld chrs▶    hld ln▶        ▶   RESULT   ▶
.DATE 24 JUL 90  13:34:10      REPORT GENERATION      WEBMJG
*==========================================================================
     IO$=2       LLP$=3       DLP$=23.
                        ..... END REPORT .....
```

I/Os,LLPs, and DLPs



Analysis Tools

- Used to analyze I/O's, LLP's, and DLP's
- Used to examine
  - Manual functions
  - Run functions
  - Summary run information
  - Performance data
- ACCTNG-LOG file
  - Logs information on all run activities and transactions
    - Kept in chronological order
  - Aids in the recovery or restart of the RCR if necessary
  - Is set by Auto Parameter/Start Parameters
  - Assigned certain values on the release tape
    - If LOGFRC = 1, then each RCR function will be separately logged
    - If LOGFRC = 0, only summary statistical information will be logged
    - Affects the way the MAPPER system functions

Note: Some Auto Parameters can be dynamically changed by operator key-in.

Accessing the Acctng-Log File

max of 32 cycles
1 cycle per day

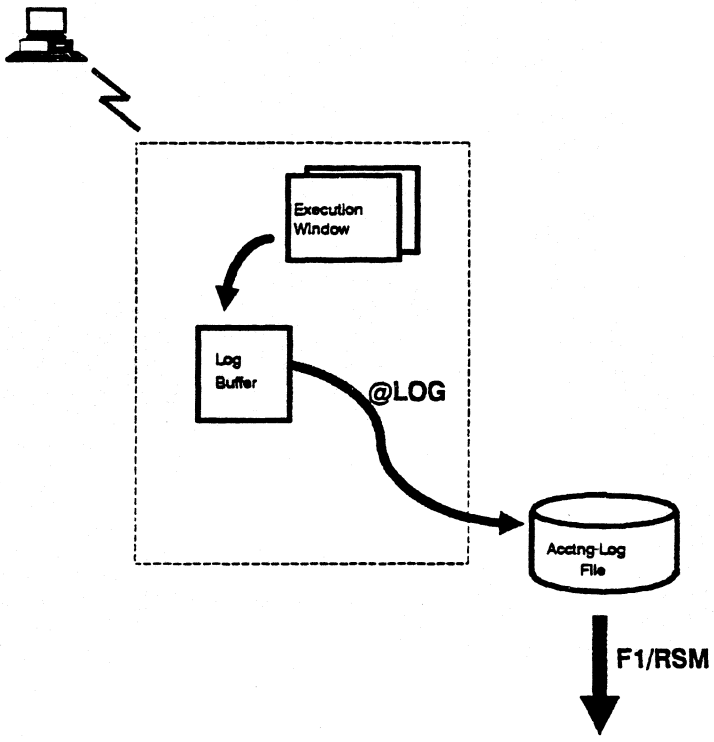
Step 1: Enter @LOG as the first statement of the RCR

Step 2: Start the run

Step 3: Upon termination of the RCR's execution

- Wait a few seconds
  (MAPPER needs time to write from the Log Buffer to the ACCTNG-LOG file)

Step 4: Enter RSM and transmit (or F1)




```
line▶ 1     fmt▶ 3 rl▶       shft▶     hld chrs▶    hld ln▶   undo▶   RESULT   ▶
.DATE 24 JUL 90  13:45:16                            WEBMJG
.LOG LIST: FROM 13:40:59  24 JUL 90   TO 13:42:53  24 JUL 90            H003356
. STA=11637, CYCLE=0
*   USER     . FUNCTION    .  TIME  . TYPE . RID.LINES-IN.LINE-OUT.READS.WRITE.BKP
*==========.============.========.======.====.========.========.=====.=====.===
 WEBMJG      PRE-RUN*     13:41:01 003330  100        6        0     3     0
```

LOG

The MARK run demonstrates use of the @LOG function. The Mark run is available to everyone and can be found in the Coordinator's database - Cabinet 0. The Mark run accesses the data reports in the demonstration database, which is found in Cabinet 0.

Mark Run

RCR

```
LINE▶ 1      FMT▶    RL▶        SHFT▶      HLD CHRS▶     HLD LN▶        ▶   3E0       ▶
.DATE 23 JUL 90  10:36:02  RID       3E   27 MAR 90  WEBMJG
.@991231 REV 01.003 'MARK' EXAMPLE RUN                BY: J.R.THALHUBER     E000010
*=============================================================================
@LOG .
@MCH,0,C,1,0,D,1 '' 'PRODUCT-TYPE','RETAIL'  ,1,A \            MATCH & EXTRACT $
 'PRODUCT-TYPE','UNIT-RETAIL'  ,1,A .
@CAL,-0 '', 'ORD-QTY','UNIT-RETAIL','EXTENDED-RETAIL' \         QTY * COST = EXT$
 ,A,B,C C=A*B;QTY=VSUM(A);RETAIL=VSUM(C) <QTY>I3,<RETAIL>I8 .    SUM QTY & EXT$
@RNM -1 .
@JUV,C <RETAIL> .                                               INSERT COMMAS
@BRK .

 CORPORATE ORDER STATUS: RETAIL VALUE AND ORDER QUANTITIES

 GRAND TOTAL ORDER QTY: <QTY>, RETAIL VALUE: $<RETAIL>

 CUSTOMER   ORDER QTY    RETAIL VALUE
*========  .=========  .============
@SUB,-1 J(L) 'ORD-QTY','EXTENDED-RETAIL','CUSTOMER(1-8)' \      SUBTOTAL QTY &
 ,+,+,$ <QTY>I,<RETAIL>I,<CUST>H .                               EXT$ BY CUSTOMER
 <CUST>        <QTY>          <RETAIL>
@BRK OUT,-0,2,23,1,0,Y,,,P .
```

```
LINE▶ 24    FMT▶    RL▶ -     SHFT▶     HLD CHRS▶    HLD LN▶        ▶   3E0       ▶

.APT TABLE
*    .D.
*CAB .W.REPT.RANG.
*====.=.====.====.
    0 C    1
.END APT TABLE

.VARIABLE TABLE
*    NAME     .VNUM.SQ.            LINE  NUMBERS                .      COMMENT
*============.====.==.=========================================.====================
 CUST         V001 01 19*,20                                    CUSTOMER NAME
 QTY          V002 01 7*,14,19*,20                              ORDER QUANTITY
 RETAIL       V003 01 7*,9,14,19*,20                            RETAIL $$$$
                          ..... END REPORT .....
```

RCR execution result

```
 CORPORATE ORDER STATUS: RETAIL VALUE AND ORDER QUANTITIES

 GRAND TOTAL ORDER QTY:  16, RETAIL VALUE: $ 380,800

 CUSTOMER   ORDER QTY    RETAIL VALUE
*========  .=========  .============
 AMERICAN      4           104650
 ARGENTIN      4           99575
 DIGITAL       1           25375
 FED SYST      4           75250
 INTERNAT      1           24150
 UNION ST      2           51800
```

LOG

Depicted on the following page is the LOG Analysis in result form. It is shown in the default format -- 3 and also in Format 0.

In Format 0, we see the complete summary listing of the Mark run. The fields and their meanings are as follows:

- **Ltname** = Communications Line Terminal (CLT) used
- **Stan** = Terminal station number
- **Department** = Department name of user
- **User** = User who executed the run
- **Function** = Run function executed
- **Time** = Time of day function invoked
- **Active** = Time active in the function
- **Type** = Drawer processed
- **Rid** = Report processed (Blank means result or all of a type)
- **Lines-In** = Number of lines read from main storage (LLPs,DLPs)
- **Lines-Out** = Number of lines written from main storage (DLPs)
- **Reads** = Number of buffers read from main storage (I/Os)
- **Write** = Number of buffers written from main storage (I/Os)
- **Bkpt** = Number of times breakpoint occurred
- **Bkpt-Time** = Time spent in Breakpoint

Bkpt and Bkpt-Time stands for system breakpoint. Prior to multiple banking/multiple memory pools, the system invoked breakpointing to spread resources between users in peak load situations. Its purpose was to avoid memory contention between functions.

*Note: An * following the function name in the LOG listing indicates it is a run statement vs. a manual function.*

Log Analysis of the Mark Run

Format 3

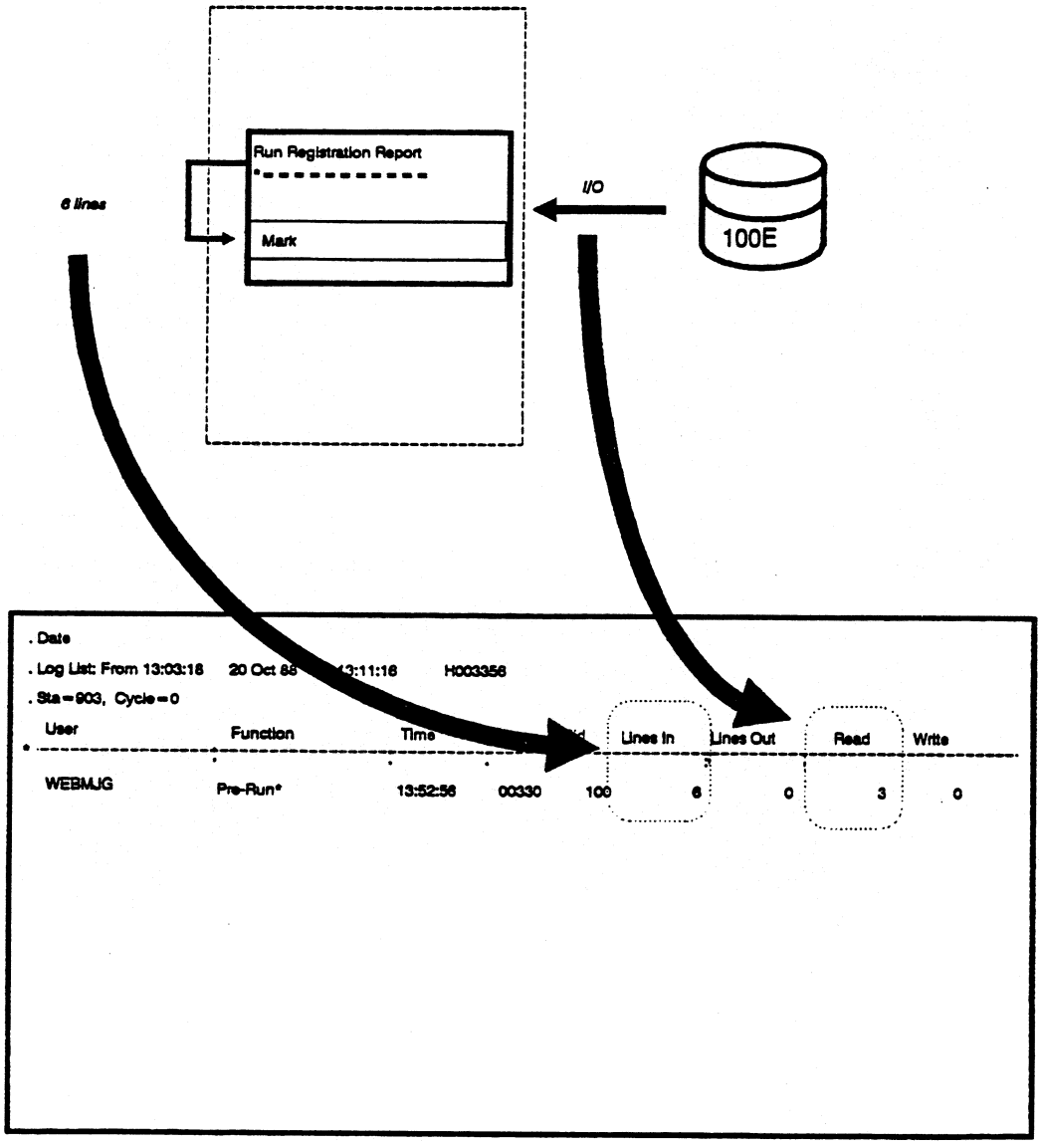
```
line▶ 1     fmt▶ 3 rl▶ -     shft▶     hld chrs▶    hld ln▶   undo▶   RESULT   ▶
.DATE
.LOG LIST: FROM 13:52:54  24 JUL 90   TO 13:54:41  24 JUL 90              H003356
. STA=11637, CYCLE=0
*   USER     . FUNCTION    .  TIME   . TYPE . RID.LINES-IN.LINE-OUT.READS.WRITE.BKP
*============.============.========.======.====.========.========.=====.=====.===
 WEBMJG      PRE-RUN*     13:52:56 003330  100        6        0     3     0
 WEBMJG      SORT*        13:52:57 000006            81       38     2     7
 WEBMJG      SORT*        13:52:57 000006            45       20     2     5
 WEBMJG      MATCH*       13:52:57 000006     1      78       58     2     0
 WEBMJG      CALCULATE*   13:52:57 000006            19       20     0     0
 WEBMJG      TOTALS*      13:52:57 000006            19        6     0     0
 WEBMJG      RUN*         13:52:57 000010     3      21       13     3     2
 WEBMJG      DISPLAY*     13:52:57 000010            13        0     0     1
 WEBMJG      RUN*         13:53:39 000010     3      15       17     3     1
 WEBMJG      DISPLAY*     13:53:39 000010            17        0     0     1
                       ..... END REPORT .....

1 Row:1  Col:18   90/JUL/24 13:48                  --
```

Format 0

```
line▶ 1     fmt▶   rl▶ -     shft▶     hld chrs▶    hld ln▶   undo▶   RESULT   ▶
.DATE
.LOG LIST: FROM 14:46:29  24 AUG 90   TO 14:48:14  24 AUG 90              H003356
. STA=11637, CYCLE=0
*LINAME.STAN .DEPARTMENT .   USER     . FUNCTION    .  TIME  .   ACTIVE   . TYPE .
*======.=====.===========.===========.============.========.============.======.
 MCB    11637 MSCSSS--100 WEBMJG      PRE-RUN*     14:46:31        0.010 003330
 MCB    11637 MSCSSS--100 WEBMJG      MATCH*       14:46:32        0.030 000006
 MCB    11637 MSCSSS--100 WEBMJG      CALCULATE*   14:46:32        0.032 000006
 MCB    11637 MSCSSS--100 WEBMJG      TOTALS*      14:46:32        0.018 000006
 MCB    11637 MSCSSS--100 WEBMJG      RUN*         14:46:32        0.144 000010
 MCB    11637 MSCSSS--100 WEBMJG      DISPLAY*     14:46:32        0.019 000010
 MCB    11637 MSCSSS--100 WEBMJG      RUN*         14:46:36        0.017 000010
 MCB    11637 MSCSSS--100 WEBMJG      DISPLAY*     14:46:36        0.018 000010
                       ..... END REPORT .....

1 Row:1  Col:18   90/AUG/24 14:38                  --                        Poll
```

PreRun Logging

- First line of the log listing shows the execution of PreRun
- PreRun looks for the mark run
  - 1 I/O Run Registration Report - 100E (octal TYPE 03330).
  - 1 I/O Global Run Registration Report
  - 1 I/O PreRun read in 6 lines before it found the Mark run.



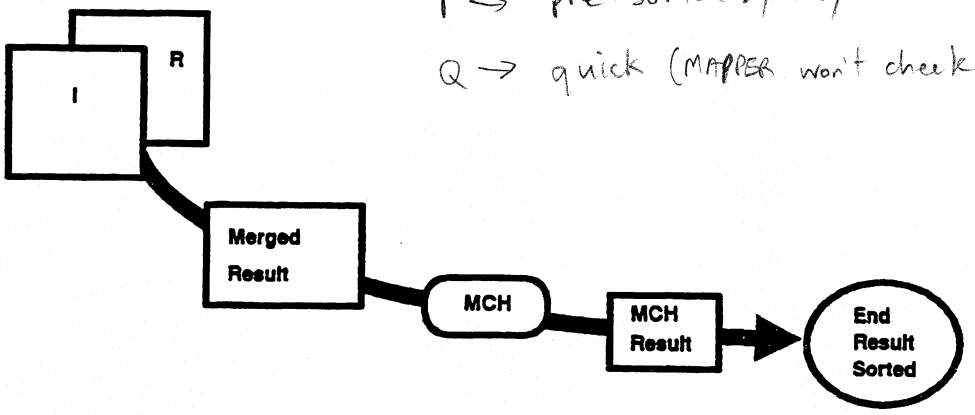
Further Analysis of the MARK Run Log Result

Using the P option with MCH:

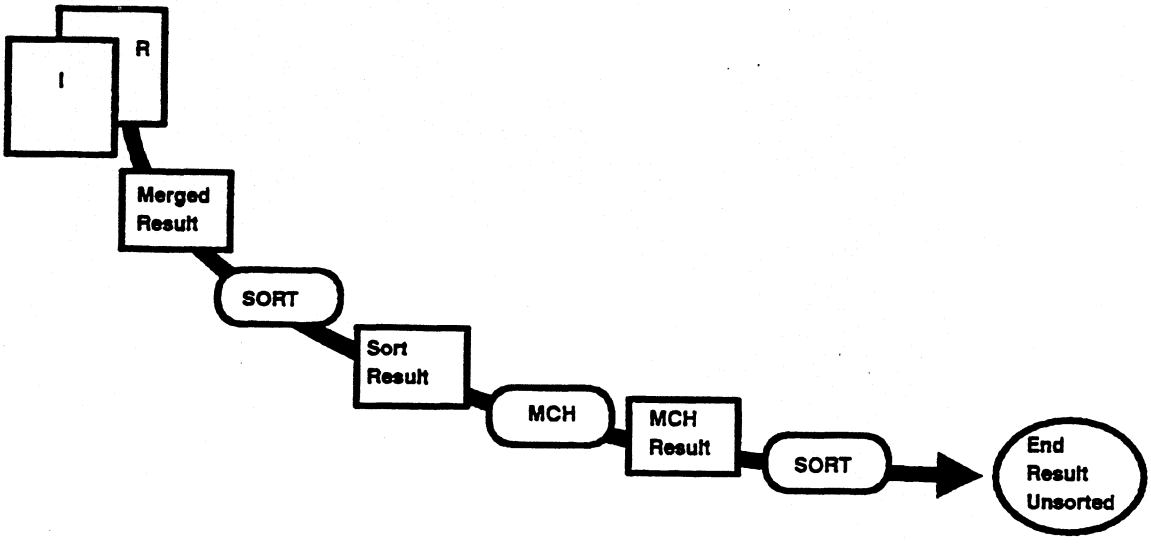
or Q

P → pre-sorted by key

Q → quick (MAPPER won't check for sorted)



When the P option is not used:



LOG of MARK Run

The LOG list does not coincide with the run's code.

@CAL

@RNM

@JUV

@BRK

@SUB

The @CAL has no I/O count because the CAL function processed the result from the Match. The result (less than 1792 words) remained in main storage, thus no I/Os.

@JUV and @BRK explained on following pages

The @TOT (@SUB) has no I/O count because variables were used. Whenever the TOT function encounters any variable, no result is produced.

LOG Analysis of the Mark Run

```
LINE▶ 1      FMT▶    RL▶        SHFT▶      HLD CHRS▶     HLD LN▶        ▶   3E0       ▶
.DATE 23 JUL 90  10:36:02  RID        3E    27 MAR 90  WEBMJG
.@991231 REV 01.003 'MARK' EXAMPLE RUN                  BY: J.R.THALHUBER     E000010
*=================================================================================
@LOG .
@MCH,0,C,1,0,D,1 '' 'PRODUCT-TYPE','RETAIL'  ,1,A \                MATCH & EXTRACT $
 'PRODUCT-TYPE','UNIT-RETAIL'   ,1,A .
@CAL,-0 '' 'ORD-QTY' 'UNIT-RETAIL' 'EXTENDED-RETAIL' \              QTY * COST = EXT$
 ,A,B,C C=A*B;QTY=USUM(A);RETAIL=USUM(C) <QTY>I3,<RETAIL>I8 .        SUM QTY & EXT$
@RNM -1 .
@JUV,C <RETAIL> .                                                   INSERT COMMAS
@BRK .

 CORPORATE ORDER STATUS: RETAIL VALUE AND ORDER QUANTITIES

 GRAND TOTAL ORDER QTY: <QTY>, RETAIL VALUE: $<RETAIL>

 CUSTOMER   ORDER QTY    RETAIL VALUE
*========  ==========   ============
@SUB,-1 J(L) 'ORD-QTY','EXTENDED-RETAIL','CUSTOMER(1-8)' \          SUBTOTAL QTY &
 ,1,1,$ <QTY>I,<RETAIL>I,<CUST>H .                                   EXT$ BY CUSTOMER
 <CUST>       <QTY>          <RETAIL>
@BRK OUT,-0,2,23,1,0,Y,,,P .

1 Row:1  Col:7    90/JUL/24 13:45                 ..                                Poll
```

```
line▶ 1     fmt▶ 3 rl▶ -     shft▶     hld chrs▶    hld ln▶   undo▶   RESULT    ▶
.DATE
.LOG LIST: FROM 13:52:54  24 JUL 90   TO 13:54:41  24 JUL 90            H003356
. STA=11637, CYCLE=0
*   USER     . FUNCTION    .  TIME   . TYPE . RID.LINES-IN.LINE-OUT.READS.WRITE.BKP
*===========.============.========.======.====.========.========.=====.=====.===
 WEBMJG      PRE-RUN*     13:52:56 003330  100        6        0     3     0
 WEBMJG      SORT*        13:52:57 000006            81       38     2     7
 WEBMJG      SORT*        13:52:57 000006            45       20     2     5
 WEBMJG      MATCH*       13:52:57 000006    1       78       58     2     0
 WEBMJG      CALCULATE*   13:52:57 000006            19       20     0     0
 WEBMJG      TOTALS*      13:52:57 000006            19        6     0     0
 WEBMJG      RUN*         13:52:57 000010    3       21       13     3     2
 WEBMJG      DISPLAY*     13:52:57 000010            13        0     0     1
 WEBMJG      RUN*         13:53:39 000010    3       15       17     3     1
 WEBMJG      DISPLAY*     13:53:39 000010            17        0     0     1
                       ..... END REPORT .....

1 Row:1  Col:18   90/JUL/24 13:48                 ..
```

LOG

The last line of the run does not match the LOG listing.

@BRK OUT

VERSUS

RUN*

DISPLAY*

RUN*

DISPLAY*

- After any kind of display statement, the Log listing shows
  - RUN* entry:
    - Includes functions that do not appear in the LOG listing (RNM,JUV,BRK)
    - Accounts for number of lines processed in RCR beginning at line three up to this point of the run
  - DISPLAY entry (BRK OUT run statement)

BRK	ESR	LNK	RDC
CAR	FDR	LOG	RDL
CER	FMT	LOK	RLN
CHD	GT	LSM	RMV
CHG	IF	MSG	RNM
CLK	INC	OK	RPW
CLV	INS	PEK	RSR
CMU	JUV	POK	SFC
CSR	KEY	PSH	TYP
DEC	LDV	RAR	ULK
DEF	LCV	REL	WAT
DFU	LFC	RER	XCH
DVS	LFN		XQT

do appear in Log listing

Further Analysis of the MARK Run Result

```
LINE▶ 1      FMT▶    RL▶        SHFT▶      HLD CHRS▶    HLD LN▶        ▶   3E0      ▶
.DATE 23 JUL 90  10:36:02  RID       3E   27 MAR 90  WEBMJG
.@991231 REV 01.003 'MARK' EXAMPLE RUN                BY: J.R.THALHUBER    E000010
*=========================================================================
@LOG .
@MCH,0,C,1,0,D,1 '' 'PRODUCT-TYPE','RETAIL'  ,1,A \            MATCH & EXTRACT $
 'PRODUCT-TYPE','UNIT-RETAIL'  ,1,A .
@CAL,-0 '' 'ORD-QTY','UNIT-RETAIL','EXTENDED-RETAIL' \          QTY * COST = EXT$
 ,A,B,C C=A*B;QTY=VSUM(A);RETAIL=VSUM(C) <QTY>I3,<RETAIL>I8 .    SUM QTY & EXT$
@RNM -1 .
@JUV,C <RETAIL> .                                               INSERT COMMAS
@BRK .

 CORPORATE ORDER STATUS: RETAIL VALUE AND ORDER QUANTITIES

 GRAND TOTAL ORDER QTY: <QTY>, RETAIL VALUE: $<RETAIL>

 CUSTOMER   ORDER QTY    RETAIL VALUE
*========  ,========   ,============
@SUB,-1 J(L) 'ORD-QTY','EXTENDED-RETAIL','CUSTOMER(1-8)' \      SUBTOTAL QTY &
 ,+,+,$ <QTY>I,<RETAIL>I,<CUST>H .                               EXT$ BY CUSTOMER
 <CUST>     <QTY>        <RETAIL>
@BRK OUT,-0,2,23,1,0,Y,,,P .
1 Row:1  Col:7    90/JUL/24 13:45                ..                          Poll
```

```
line▶ 1     fmt▶ 3 rl▶ -     shft▶     hld chrs▶    hld ln▶   undo▶   RESULT   ▶
.DATE
.LOG LIST: FROM 13:52:54  24 JUL 90   TO 13:54:41  24 JUL 90            H003356
. STA=11637, CYCLE=0
*   USER    . FUNCTION    .  TIME   . TYPE . RID.LINES-IN.LINE-OUT.READS.WRITE.BKP
*==========.============.========.======.====.========.========.=====.=====.===
 WEBMJG      PRE-RUN*     13:52:56 003330  100        6        0     3     0
 WEBMJG      SORT*        13:52:57 000006            81       38     2     7
 WEBMJG      SORT*        13:52:57 000006            45       20     2     5
 WEBMJG      MATCH*       13:52:57 000006    1       78       58     2     0
 WEBMJG      CALCULATE*   13:52:57 000006            19       20     0     0
 WEBMJG      TOTALS*      13:52:57 000006            19        6     0     0
 WEBMJG      RUN*         13:52:57 000010    3       21       13     3     2
 WEBMJG      DISPLAY*     13:52:57 000010            13        0     0     1
 WEBMJG      RUN*         13:53:39 000010    3       15       17     3     1
 WEBMJG      DISPLAY*     13:53:39 000010            17        0     0     1
                     ..... END REPORT .....

1 Row:1  Col:18   90/JUL/24 13:48                ..
```

LOG

The DISPLAY entry means that the Run Interpreter has encountered either an @BRK,@DSP,@OUT or @GTO END. A result with processed data/information was sent to the screen. The effect of DSP and OUT statements on the reserved words IO$,LLP$,DLP$ should be considered.

A noninterim @DSP or @OUT resets the counters of the reserved words to zero. In a noninterim display, the RCR's D Bank is swapped out to Maper0. The idea behind this is the same as BRKPTing, it is a user initiated swap to maximize memory usage whereas BRKPTing is initiated by the system.

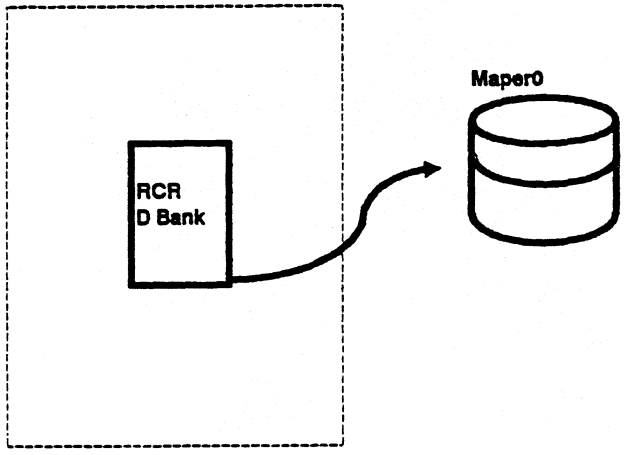
An interim display does not reset the reserved words to zero.

Caution: If using reserved words in combination with noninterim DSPs, OUTs be aware that the counts in IO$,LLP$,DLP$ are not cumulative.

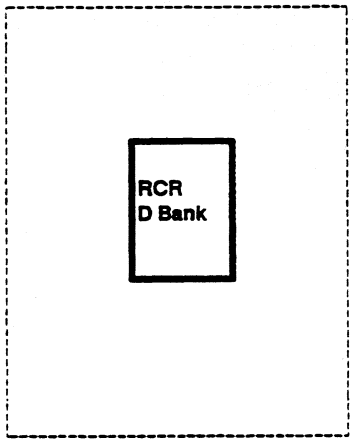
Note: A forced transmit on the OUT also resets the counters to zero. However, this was not the original intention of the forced transmit

Noninterim vs Interim

Noninterim IO$ LLP$ DLP$ = 0



Interim IO$ LLP$ DLP$ = No change



LOGLA

LOGLA is another analysis tool that provides more detailed information for the run designer. LOGLA performs further analysis of the @LOG listing.

Procedure:

Step 1: Enter @LOG as the first statement of the RCR.

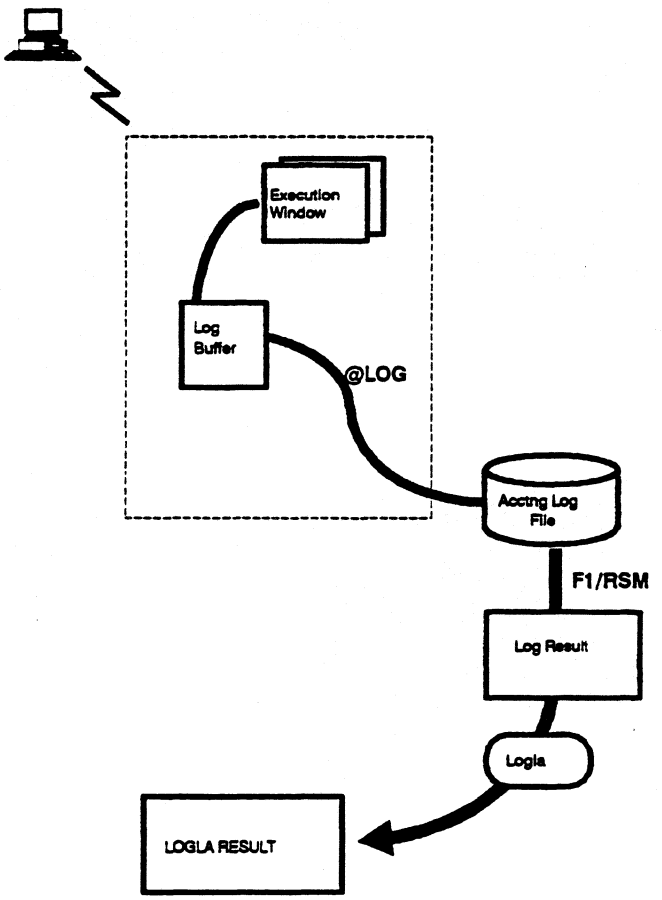
Step 2: Start run

Step 3: Upon termination of the RCRs execution

- Wait a few seconds
- Enter RSM and transmit (F1 key)

Step 4: Home cursor

- Enter LOGLA on line 0/control line and transmit



LOGLA - Parts

The LOGLA listing looks much like the LOG listing. At the top of the LOGLA result there is a summary of the run. In this summary information we see a new term.

- PART

PART is a division of the RCR by noninterim displays. Each time the LOGLA encounters a noninterim DSP,OUT,SC or GTO END, a **PART** entry appears along with the summary information up to that point. This occurs due to the swapping of the D Bank on a noninterim display.

The **LOGLA** breaks a run into logical units and presents a summary of each of the logical units. It is exactly for this reason that the LOGLA is preferred over the LOG for long runs. The summaries provide a means to fine tune a run.

```
line▶ 4     fmt▶   rl▶ -     shft▶     hld chrs▶    hld ln▶       ▶   RESULT    ▶
              Functions        I/O          Logic Lines      Data Lines
             ============ ================ ================ ================
Part 1     =           5                8               21              170
Part 2     =           2                4               15               52
             ------------ ---------------- ---------------- ----------------
    Total =            7               12               36              222

Approximate charge for I/O processing  $    0.01
.
*   USER     .  FUNCTION  .   ACTIVE   . TYPE .RID .LINES-IN.LINE-OUT.READS.WRITE
*============.============.============.======.====.========.========.=====.=====
 WEBMJG       PRE-RUN*          0.073 003330  100        4        0     1     0
 WEBMJG       MATCH*            0.046 000006    4       41       20     2     0
 WEBMJG       CALCULATE*        0.029 000006            19       20     0     0
 WEBMJG       TOTALS*           0.029 000006            19       13     0     0
 WEBMJG       RUN*              0.149 000010   38       21       13     3     2
Part 1     : Funcs = 5      LLP = 21        Lines = 170        I/O = 8
 WEBMJG       DISPLAY*          0.134 000010            20        0     0     1
 WEBMJG       RUN*              0.084 000010   38       15       17     2     1
Part 2     : Funcs = 2      LLP = 15        Lines = 52         I/O = 4
 WEBMJG       DISPLAY*          0.021 000010            17        0     0     1
                        ..... END REPORT .....
1 Row:1  Col:24   90/AUG/24 14:43
```

RUNA

- RUNA provides extremely detailed information about:
  - I/O distribution
  - Line usage
  - Efficiency guidelines
  - Registration data

Procedure:

Step 1: Enter @LOG as the first statement of the RCR

Step 2: Start run

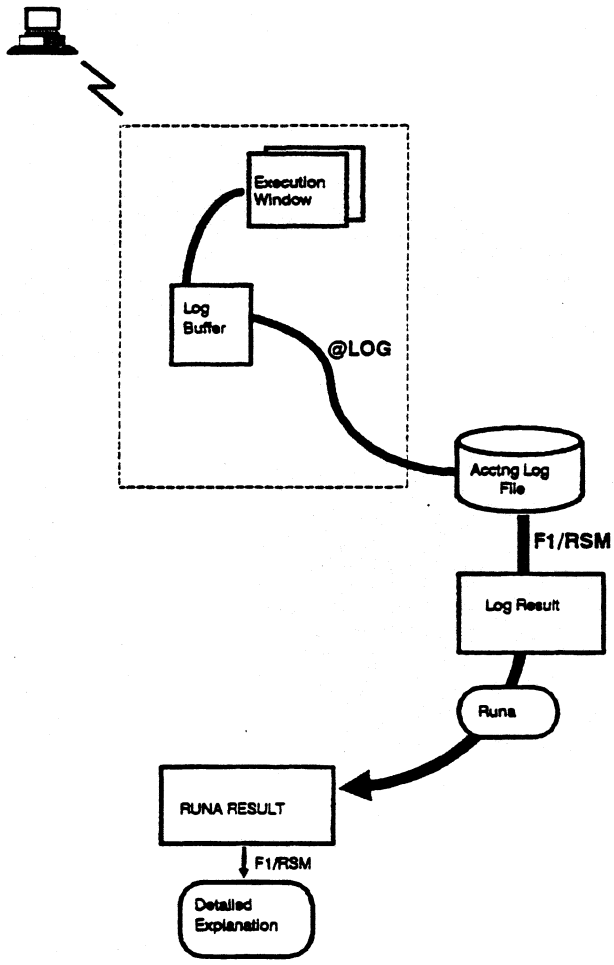
Step 3: Upon termination of the RCR's execution

- Enter RSM and transmit (or F1)

Step 4: Home cursor

- Enter RUNA on line 0/control line and transmit

Step 5: For further detailed explanations, enter RSM and transmit



RUNA

```
line▶ 1      fmt▶   rl▶        shft▶     hld chrs▶    hld ln▶   undo▶   RESULT   ▶
.DATE 24 AUG 90  15:01:03      REPORT GENERATION      WEBMJG

 * * * * * *   M A P P E R   1 1 0 0   R U N   A N A L Y Z E R   * * * * * *

                        ANALYSIS OF RUN MARK
                 RUN CONTROL RID 38E IN MODE PAIR 0/1

                          *** LOG LIST ***
  FUNCTION    .  TIME   .   ACTIVE   . TYPE . RID.LINES-IN.LINE-OUT.READS.WRITE.
============.========.============.======.====.========.========.=====.=====.
PRE-RUN*     14:54:15        0.013 003330  100        4        0     1     0
MATCH*       14:54:15        0.029 000006    4       41       20     2     0
CALCULATE*   14:54:15        0.027 000006            19       20     0     0
TOTALS*      14:54:15        0.022 000006            19       13     0     0
RUN*         14:54:15        0.169 000010   38       21       13     3     2
DISPLAY*     14:54:15        0.021 000010            20        0     0     1
RUN*         14:54:23        0.017 000010   38       15       17     2     1
DISPLAY*     14:54:23        0.019 000010            17        0     0     1
```

```
line▶ 25     fmt▶   rl▶ -     shft▶     hld chrs▶    hld ln▶   undo▶   RESULT   ▶
                 *** DATA I/O DISTRIBUTION BY FUNCTION ***
       FUNCTION        NUMBER  DATA LINES     I/O'S        % OF I/O'S
      ------------     ------  ----------    ------       ----------
      DISPLAY*           2        37           2            40.00
      MATCH*             1        61           2            40.00
      PRE-RUN*           1         4           1            20.00
      CALCULATE*         1        39           0             0.00
      TOTALS*            1        32           0             0.00

                      *** LINE USE SUMMARY ***
                                 LINES              I/O'S
                                -------            -------
    LOGIC LINES                   36                 5
    DATA LINES                   203                 8

    TOTALS                       239                13

              THE APPROXIMATE COST FOR PROCESSING WAS $0.014

*******  SOME GENERAL GUIDELINES FOR GOOD RUN DESIGN HAVE NOT BEEN MET  *******

NUMBER
------
```

```
line▶ 48     fmt▶   rl▶ -     shft▶     hld chrs▶    hld ln▶   undo▶   RESULT   ▶

*01 THE RATIO OF DATA LINES TO LOGIC LINES IS  5.64 TO 1.
    (TRY FOR 10 OR GREATER)

*01 THE RATIO OF DATA I/O'S TO LOGIC I/O'S IS  1.60 TO 1.
    (TRY FOR 10 OR GREATER)

 ******                   RUN REGISTRATION DATA                          ******
                 IO'S ALLOWED -  1000  LLP ALLOWED -  2000
                        RUN DESIGNER -
                          - MODES REGISTERED -
                                 0

    MULTI REGISTRATION -    RUN STATUS -     BATCH LEVEL - NO

 ******                  USER REGISTRATION DATA                          ******
 USER SIGN ON - WEBMJG      NAME - CARPENTER,R        DEPT -  100
 EXT-LOCATION - 4292/BLUE BELL              STATION NUMBER - 11637
 THIS USER IS A RUN DESIGNER

       IF YOU WANT EXPLANATIONS OF THE GUIDELINES PLEASE RESUME (F1)
                      ..... END REPORT .....
```

RUNA

I/O Distribution

The top section of the RUNA listing repeats the same information seen in the LOG, LOGLA listings. However, immediately below the function entries appears the I/O distribution data. This data provides overhead information for each individual function executed in the RCR. The I/O distribution data is broken down for each function. The information includes:

Function	Name of the executed function
Number	The number of times the function was requested
Data Lines	The number of data lines processed for that function (Lines-In + Lines-Out)
I/Os	The number of I/Os created by that function (Reads + Writes)
% of I/Os	The percentage of I/Os this function used in relationship with the total I/Os

Note: This section is listed not by function execution order but by the % of I/Os column. The highest I/O user is listed first. The noninterim displays are not included in the distribution breakdown.

Line Use Summary

The Line Use Summary of the RUNA provides information concerning the overall impact of the run on LLPs, DLPs, I/Os. Recall the RUN entry accumulates the total number of logic lines processed in the run; therefore, they are not included in the Data Line count. The Line Use Summary information includes:

Lines	Total numbers of the LLPs and DLPs
I/Os	The number of I/Os created to retrieve and transfer the lines of data
Approx Cost	How much the run cost

RUNA

```
line▶ 25    fmt▶   rl▶ -    shft▶    hld chrs▶   hld ln▶   undo▶   RESULT   ▶
                 *** DATA I/O DISTRIBUTION BY FUNCTION ***
   FUNCTION      NUMBER  DATA LINES    I/O'S      % OF I/O'S
   ----------    ------  ----------   ------     ----------
   DISPLAY*        2        37          2          40.00
   MATCH*          1        61          2          40.00
   PRE-RUN*        1         4          1          20.00
   CALCULATE*      1        39          0           0.00
   TOTALS*         1        32          0           0.00

                 *** LINE USE SUMMARY ***
                          LINES        I/O'S
                          ------       ------
   LOGIC LINES              36           5
   DATA LINES              203           8

   TOTALS                  239          13

          THE APPROXIMATE COST FOR PROCESSING WAS $0.014
******* SOME GENERAL GUIDELINES FOR GOOD RUN DESIGN HAVE NOT BEEN MET *******

NUMBER
======
1 Row:1  Col:24   90/AUG/24 14:53
```

RUNA

Guidelines

This section lists suggestions pertaining to efficient run design techniques. The guidelines attempt to detect potential problem areas. It is up to the run designer to implement these recommendations and find the correct solutions.

Registration Data

The last two sections of the RUNA result show information from:

- Run Registration Report
- User Registration Report

This information has been entered by the coordinator.

The Run Registration Data reflects information from the Run Registration Report:

I/O limit

LLP limit

Cabinets registered

Run status

Batch level

The User Registration Data reflects information from the User Registration Report

User Sign-On

Name

Department

Extension

Station Number

RUNA

```
line▶ 48    fmt▶   rl▶ -    shft▶    hld chrs▶    hld ln▶   undo▶   RESULT   ▶
*01 THE RATIO OF DATA LINES TO LOGIC LINES IS  5.64 TO 1.
    (TRY FOR 10 OR GREATER)

*01 THE RATIO OF DATA I/O'S TO LOGIC I/O'S IS  1.60 TO 1.
    (TRY FOR 10 OR GREATER)

******                    RUN REGISTRATION DATA                          ******
              IO'S ALLOWED -  1000  LLP ALLOWED -  2000
                      RUN DESIGNER -
                        - MODES REGISTERED -
                                0


   MULTI REGISTRATION -    RUN STATUS -     BATCH LEVEL - NO

******                    USER REGISTRATION DATA                         ******
USER SIGN ON - WEBMJG      NAME - CARPENTER,R        DEPT -  100
EXT-LOCATION - 4292/BLUE BELL            STATION NUMBER - 11637
THIS USER IS A RUN DESIGNER

       IF YOU WANT EXPLANATIONS OF THE GUIDELINES PLEASE RESUME (F1)
                      ..... END REPORT .....
1 Row:1  Col:24   90/AUG/24 14:54                                          Poll
```

Exercises

1. Define I/O.
2. Define LLP.
3. Define DLP.
4. Write a run that will match on Product Type and move Product Cost field. (Use a report from type B of JDOE database and type C. You will execute the run 3 times.
   1 - Data **not** sorted without the P option.
   2 - Data **sorted, no** P option.
   3 - Data **sorted**, with the **P option**

   a. Execute the Run.

   b. Wait a few seconds and resume the run.

   c. Append the LOG listing to the RCR.

   d. Print it (PR).

   e. Repeat the same step using RUNA
5. All _____ are written to the recovery tape. If these _____ change the size of a report, the _____ is written to the recovery tape.

4

I/O Considerations

Module 4
I/O Considerations

Objectives

Upon completion of this module,you should be able to:

1. Apply RUNA recommendations.
2. List 2 ways a run may be registered to lower the impact on system resources.
3. Explain how characters per line and number of lines per report affect efficiency.

Point Counterpoint

The guidelines found in the RUNA results gear toward improvement of run design efficiency. As mentioned earlier, it is up to the Run Designer to implement the recommendations. The problem with implementation, however, is twofold:

First, no solutions are provided.

Second, efficiency almost always involves a trade-off between maintainability and system resources. Trade-offs occur between resources themselves - thus point, counterpoint.

I/O Considerations

A major concern of efficient run design is the maintenance of low I/Os. Thus many of the guidelines found in the RUNA listing concern themselves with I/Os. Reviewing the items that impact the I/O count:

- Resident vs. Nonresident code
- Buffer Sizes
- Size of result
- Character set
- Characters per line
- Number of lines processed (Function Power Curve)

I/O Considerations : Time Slotting

A frequently seen guideline in RUNA is Time Slotting Due to I/O Quantity. This means the run is exceeding 1000 I/Os during execution. The assumption being noninterim displays do not exist in the RCR (no resetting of the I/O counters occurs).

An average I/O can process 77 lines of an 80 character report in FCS. Thus 1000 I/Os can process 77,000 lines. This is sufficient for a single transaction. Executing larger runs at prime time can have adverse effects on the system.

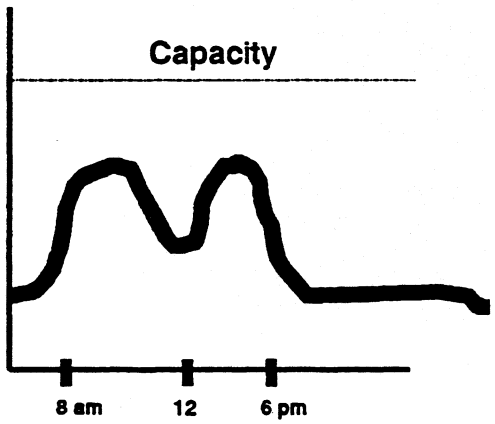
Solution One: Time slot the RCR to execute at a low-use period.

Point: Time slotting lessens the impact on the system during peak usage.

Counterpoint: It is more than likely that low-usage periods are during off work hours.

Run Registration Report

```
. Date 20 Oct 88  11:55:46  Rid 8E  20 Oct 88
. Run Control For                                   E003330
* Runid        B        User        P        BTime        ETime
*
  XYZ                   JLD                  081000       140800
```



I/O Considerations : Priorities

Solution Two: Lower priority of entire MAPPER system and of individual runs using TIP (Transaction Interface Processing).

TIP:

- Priority falls between real time and batch.
- Permits fine tuning of a MAPPER system.
- Has seven levels indicated by a numeric (1-7)
  - 1 = Highest, 7 = Lowest
- Auto Parameter TIPLVL set to Tip level
- Allows individual run prioritizing A-G within TIP level
  - A = Highest priority and G = Lowest

If Auto Parameter, TIPLVL = 2, means the entire system processes at the second highest TIP level. Then using the priority field of the Run Registration Report, individual runs are given a priority of A-G. So that, if the P field of the Run Registration Report contains a G, then that run executes at the lowest TIP priority of seven regardless of the TIPLVL setting .

Point: Executing at TIP priority permits fine tuning and lessens impact on lower priority levels.

Counterpoint: Demand and Batch users still feel the impact.

TIP

- **Can lower priority of entire MAPPER system and of individual runs**

TIP LEVELS

Highest	1
	2
	3
	4
	5
	6
Lowest	7

RUN REGISTRATION REPORT

```
LINE▶ 1     FMT▶   RL▶ -     SHFT▶     HLD CHRS▶    HLD LN▶   UNDO▶   3E218    ▶
.DATE 30 AUG 90  11:21:55  RID      3E   24 AUG 90  WEBMJG
.RUN REGISTRATION RID DEP'T. 3                                          E003330
.                          *****KEEP RID SORTED BY RUN NAME*****
*  RUNID     B.    USER    P. UNIT. TYPE . RID.F. B TIME . E TIME . I/O .LINES.MOD
*============.============.=====.======.====.=.========.========.=====.=====.===
 RUN NAME                          000000    0                      1000  2000 0
 RUN10                             000010   10                      2000  2000 0
 RUN11                             000010   11                      2000  2000 0
 RUN12                             000010   12                      2000  2000 0
 RUN100       JLD        G         000010   22 MAPPER EXECUTING AT TIP PRIORITY G
 RUN200       JLD        A         000010  200 MAPPER EXECUTING AT TIP PRIORITY A
                      ..... END REPORT .....

1 Row:11 Col:80   90/AUG/30 11:14       ↑      ..
```

I/O Considerations : Background Runs

Solution Three: Make the RCR a background run

A background run:

- Is a Noninteractive run
- Frees the terminal
- Executes in the MAPPER environment in the background
  - MAPPER still looks at it as an activity executing at the current priority level
- May not contain DSP, OUM, REL, RSI, or XIT statement
- May contain DSG, SC, or OUT statement:
  - Only if the output is sent to another terminal
- May use reserved word ORSTAN$
  - Contains the station number of the calling run.
- Cannot be executed from a background run
- Allows current result to be passed to the Background run
- Auto parameter MAXBGR, controls the maximum number of background runs allowed
  - Default is four
- @RS (Run Status) statement produces a result that shows the status of runs being executed with the user's user-id

Point: A background run frees the terminal for other activities and permits processing during work hours.

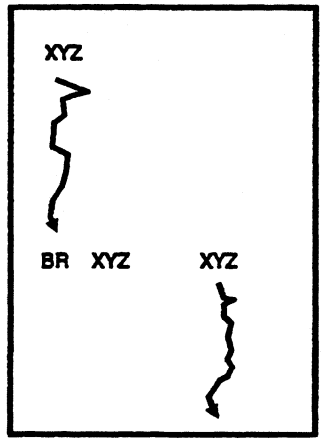
Counterpoint: Loss of interaction with a background run and it still impacts lower priorities.

Background Run

- **Background runs are noninteractive runs that free the terminal for other activities**

Run Registration Report

```
. Date 20 Oct 88  11:55:46   Rid 8E   20 Oct 88
. Run Control For                                          E003330
* Runid        B        User
* ---------------------------------------------------------------------------
  XYZ                   JLD      Mapper activity executing at real time in the foreground
  XYZ          *        JLD      Mapper activity executing at real time in the backgound
```



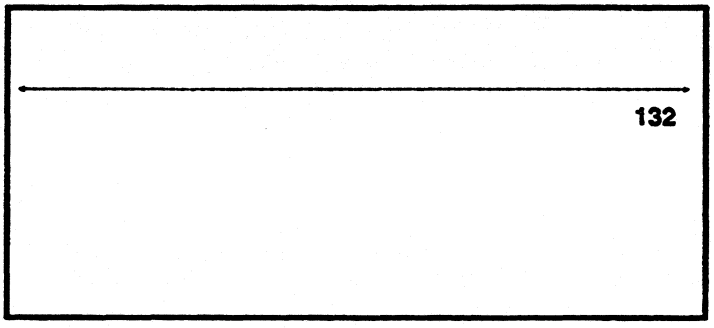
Line Length

- Directly affects I/Os
- Eighty character reports save up to 39% in I/Os over 132 character reports
- Savings directly relates to processing of logic lines

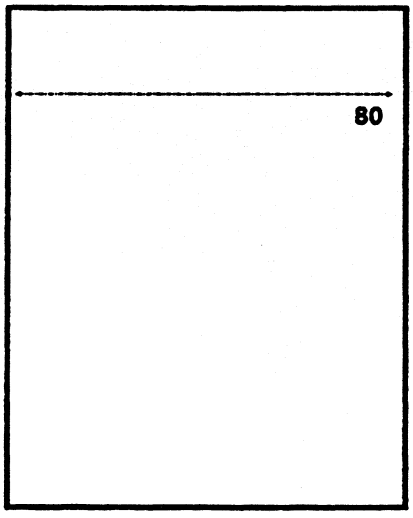
Solution: Wherever there are many logic lines to process, an 80-character report should be considered.

Point: Savings in I/Os.

Counterpoint: Before an existing run is moved to an 80-character report, coordination must weigh cost and time needed to redesign the run.



versus



Lines Processed : Single vs Multiple

One frequently seen guideline is 'The Ratio of data I/Os to logic I/Os is 8.67 to 1. (Try for 10 or greater)'

- The greater the number of lines to process, the more I/Os are created
- If ratio of data lines (data I/Os) to logic lines (logic I/Os) is less than ten, the run is not taking full advantage of run function capabilities.
- For one logic line processed, ten data lines should be processed

Solutions: Compress the number of logic lines by using multiple statements per line. Shorten logic loops to fit in one Execution Window. Process more data.

Point: Any of the above solutions improves the ratio.

Counterpoint: Compression of LLPs and shortening logic loops may hinder maintainability.

Single statements

```
*-------------------------
@Ldv v1l3 = 123 .
@Ldv v2a4 = xyzk .
@1 .
@if v1 = 123 gto end .
@dup,0,b,2 .
@chg < rid > l3 rid$ .
        .
        .
        .
```

Multiple statements

```
*-------------------------
@Ldv v1l3 = 123 ,v2a4 = xyzk .
@1:if v1 = 123 gto end .
@dup,0,b,2 chg < rid > l3 rid$ .
        .
        .
        .
```

Lines Processed : Limit Report Length

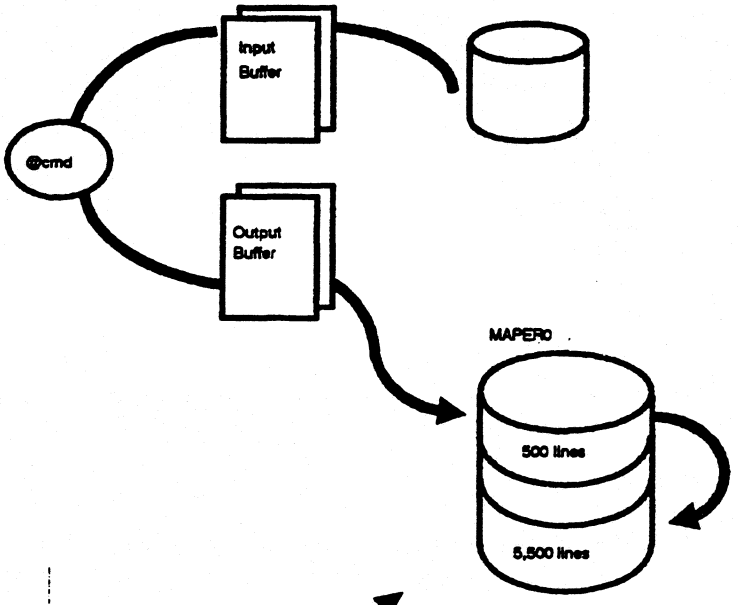
MAPPER requires that all results/reports are allocated with contiguous disk space. As MAPPER processes reports/results, the Output Buffer fills. When the Output Buffer is full, it is written out to MPOS of MAPER0. Initially, MAPPER allocates a space on MAPER0 that holds up to 500 contiguous lines of processed data for each report/result.
If the 500 line buffer space isn't large enough:

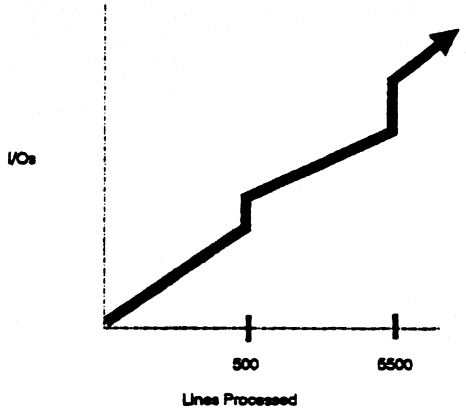
- MAPPER allocates an additional 5,000 lines of space
- Then copies the previous 500 lines to a contiguous space of 5,500

Solutions One and Two: Reduce reports to 500 lines or less. Lower priority to batch.

Point: Either solution lessens impact on the system.

Counterpoint: It isn't always possible or feasible to reduce reports/results to 500 lines and batch processing takes longer and creates additional I/Os.





Lines Processed : E Option

Solution 3: Use the E option with functions producing results or the @BRK statement for estimating the size of the output area.

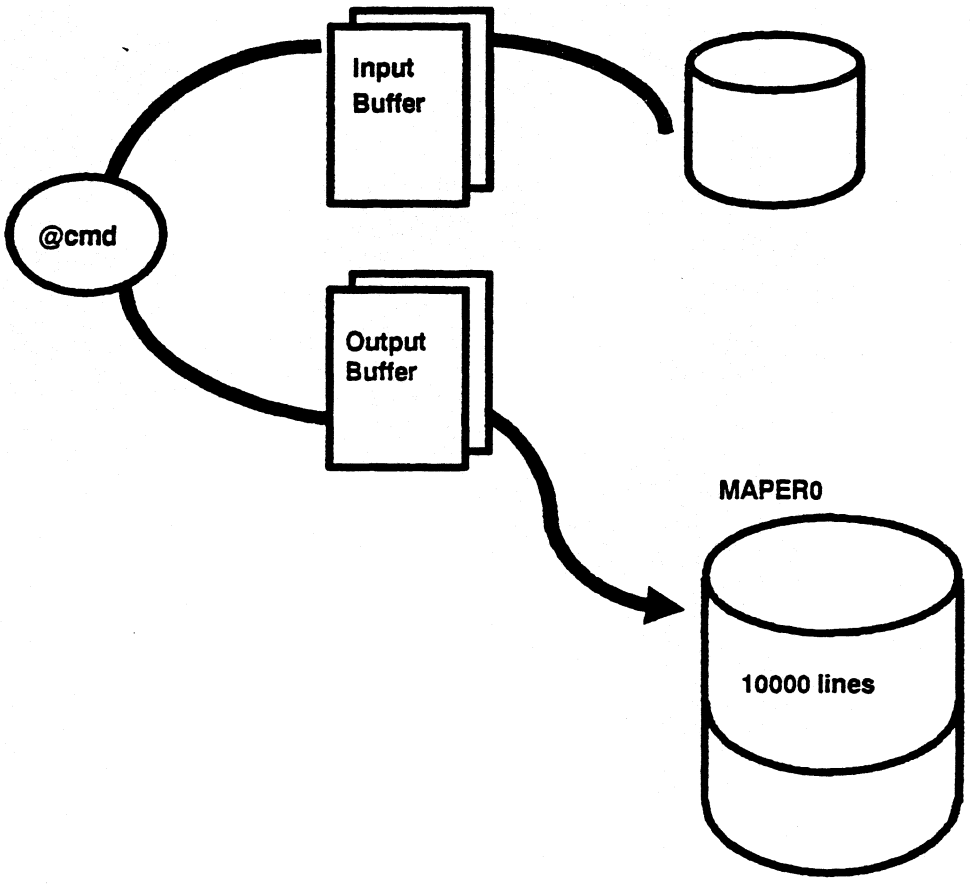
The E option:

- Stands for estimate - Estimate the size of the result
- Permits us to allocate contiguous MAPER0 space in increments greater or less than 500 lines

Rule of Thumb: Always estimate high when using the E option or @BRK,,#.

Point: Using the E option or @BRK saves the system overhead in terms of I/Os and processing time.

Counterpoint: An underestimation in either case will defeat the purposes of the option or @BRK.



Comment Lines

RUNA will detect run control statements that have not terminated. Each @ statement must be terminated by a space-period-space combination to terminate scanning and to save processing of unused characters. The Run Interpreter continues to read the remaining Fieldata or Ascii characters of that line even though they are spaces. The Run Interpreter continues scanning until it encounters a space-period-space combination.

What constitutes a comment line?

- A space-period-space starting in column one is placed into the output area and is counted as an LLP and DLP
- An @-space-period-space combination constitutes a true comment line. Only the LLP counter is incremented

Solution One: Use an @GTO to skip around the comment lines. Must have label table built in beginning of run.

Point: No cost either as DLPs or LLPs.

Counterpoint: They are still part of the RCR and in production those comment lines take up main storage and disk space.

Solution Two: The ideal solution is to place the comment lines outside the production RCR altogether. Maintain a corresponding report and type for documentation purposes only and submit the stripped version to production.

Point: Savings on all aspects - No LLPs, no DLPs , no extra space taken.

Counterpoint: A single cost remains in maintaining a corresponding report and type with documentation.

Comment Lines

```
*==========================
. These are not comment lines - They are counted
. as LLPs and DLPs
@ . These are comment lines - But they are still
@ . counted as LLPs
```

versus

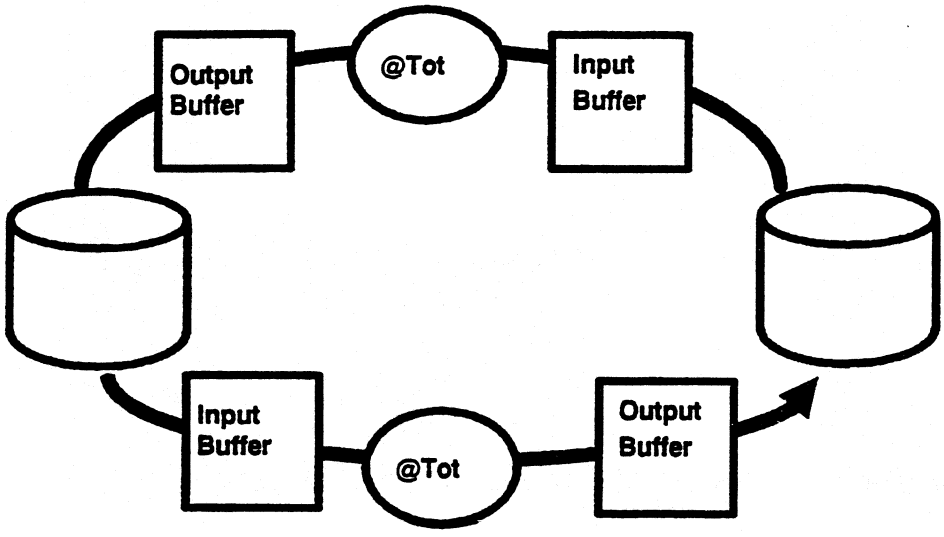
```
*==========================
:L1=10
@GTO 1 .
. Using a label to circumvent the comment lines is
. one way of resolving the problem
. The other way would be to keep a separate
. document run
@ 1 : Ldv v1i3=123 .
```

Function Logic : TOT vs CAL

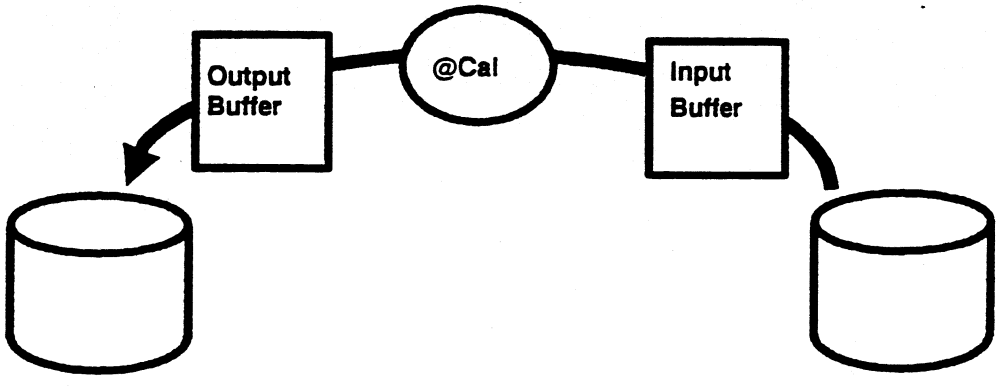
Solution: Replace multiple @TOT statements with an @CAL statement.

Point: One CAL performs the same calculation(s) as two or more TOTs in a single pass of the data. One CAL statement equals one LLP whereas two TOTs may equal two LLPs.

Counterpoint: Calculate may not be a resident function because of its size (I/Os). The CAL statement is more difficult to decipher than the @TOT.



versus



CHG

Another function RUNA keeps a look out for is the @CHG. When RUNA encounters five @CHG statements, a guideline appears suggesting usage of @LDV.

The @CHG should only be used when:

- Arithmetic is involved
- Data input reserved words are needed (INPUT$,INVAR$,...)

Solution: Replace pertaining @CHG statements with @LDV or @LDV,W statements or with @INC, @DEC statements.

Point: @LDV permits multiple variable initialization with less column usage and perhaps less LLPs. Furthermore, only one pass is necessary through the LDV code and several functions are performed.

Counterpoint: There are none!

```
*---------------------------
@Chg v1l3 100 chg v2l1 1 .
@Chg v3h6 date1$ .
@Chg v5 v5 + 1 .
@Chg v6 v6 - 1 .
.
.
.
```

versus

```
*---------------------------
@Ldv v1l3 = 100, v2a1 = 1 .
@Ldv,w v3h6 = Date1$ .
@Inc v5 .
@Dec v6 .
.
.
.
```

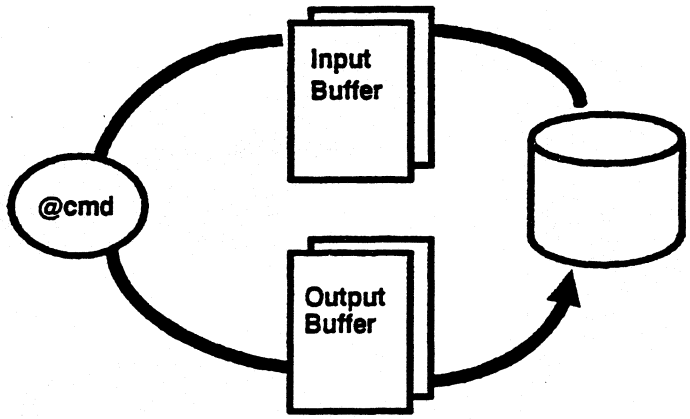
Multiple Report Accesses

RUNA also reveals patterns of repetitive report or result processing by the same function. This indicates a single pass through the data may give identical results.

Solution: Test various techniques with the LOG and RUNA analysis tools until a better method is found. If a single report is constantly searched with different parameters perhaps the solution is to read the individual lines (one pass) and use the @IF statement to make decisions.

Point: Cutting down to a single pass saves in I/Os,LLPs,DLPs.

Counterpoint: Finding methods and logging takes time and work.




```
*=================
@Srh,0,b,2 ...   .
@Srh,0,b,2 ...   .
@Srh,0,b,2 ...   .
                 versus

@Rdl,0,b,2 ...   .
@Rdl,0,b,2 ...   .
@If this then that .
```

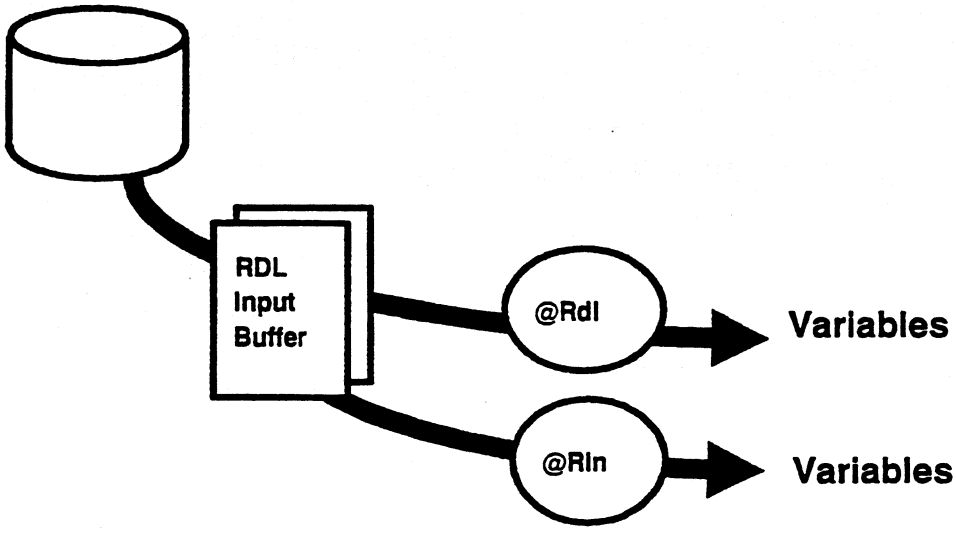
Reading Lines : RDL (Read Line)

- Reads a line or part of a line from a report
- MAPPER sets aside buffer space in main storage and pulls in a buffer starting at the indicated line number
- If multiple lines are read, a loop is necessary
  - To accomplish this successfully, we were given RLN (Read Line Next).
  - RLN reads data from the current read buffer
- No run statements are permitted between the RDL, RLN combination
  - Because run statements change buffers
  - RLN works on the pretext of a currently available RDL buffer

Point: Permits reading of single/multiple lines without cost if within buffer boundaries.

Counterpoint: The RDL does not permit specification of a line type.

Note: Lock (@LOK) a report prior to any reads to ensure integrity of the data. Any I/Os incurred from the above read statements are logged under the RUN entry.



Reading Lines : RDC (Read Continuous)

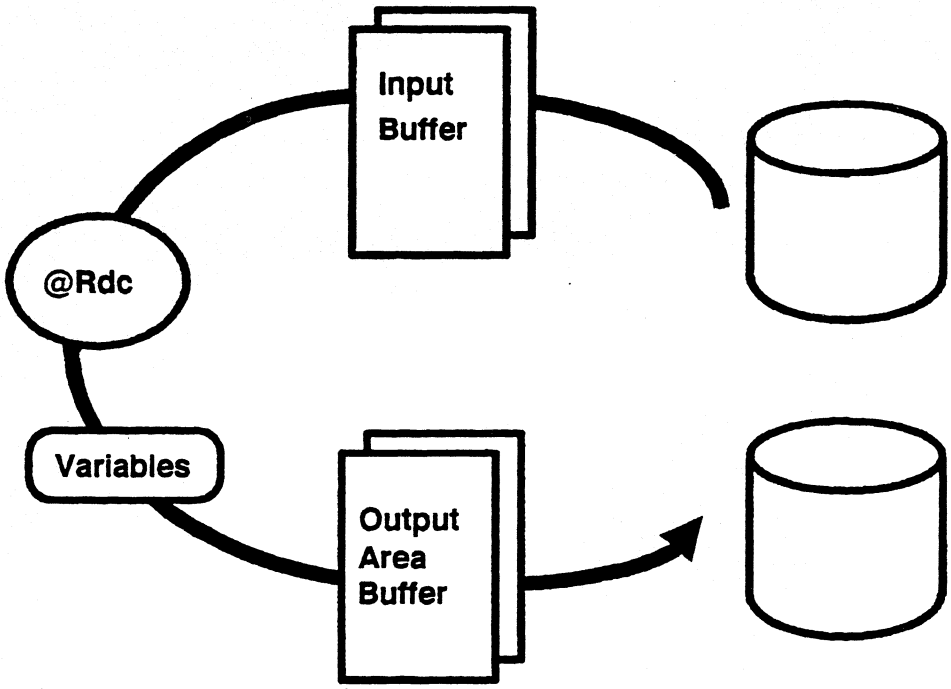
The **RDC** (Read Continuous) is also used to read lines or parts of lines from a report.

With the **RDC** statement:

- Output line must follow
- Reads a line and places data read into variables, writes data to Output Area buffer and reads next line
- Makes use of the Output Area buffer as its output buffer
- There is an implied loop with the RDC
- Usually used to read a block of data

Point: Can specify line type to process; can specify the number of lines to read; no labels or loops are necessary; can reformat data in the Output Area.

Counterpoint: There are none.



Reading Lines : FDR

The **FDR** (Find and Read) statement. This statement does the same thing as a FND/RDL combination except the FDR costs one less I/O.

Using the **FND/RDL** combination MAPPER:

- Reads the FND code
- Allocates buffer space
- Positions the pointer to the correct find
- Reads the RDL code
- Allocates buffer space
- Reads the correct line

Whereas with the **FDR/RLN** combination:

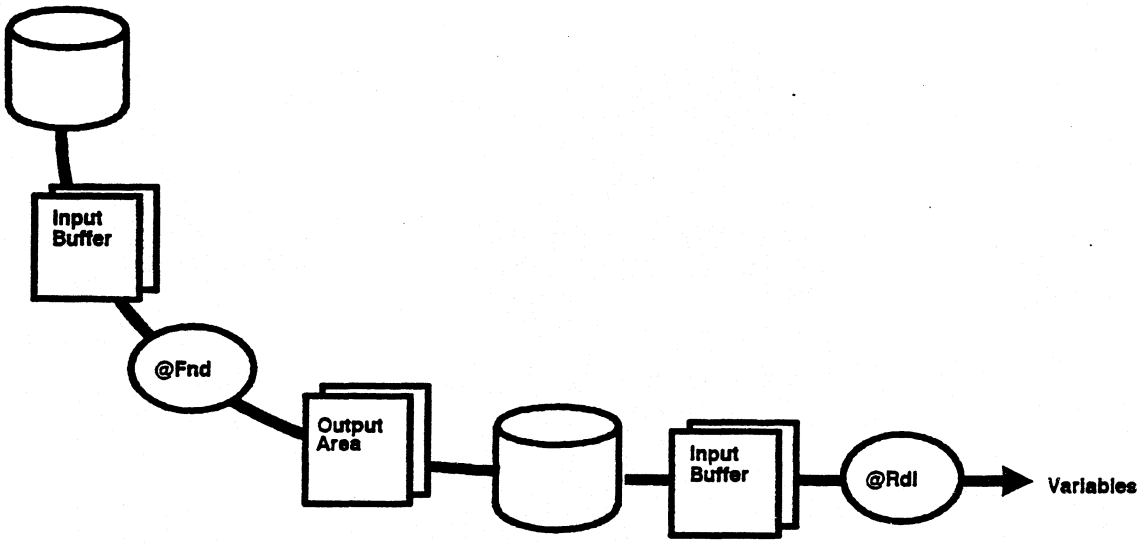
- FDR finds and reads actual line
  - Recall RLN statement works with current read buffer thus no additional I/Os are incurred positioning to the next line.

Point: Costs one less I/O over the FND/RDL combination.

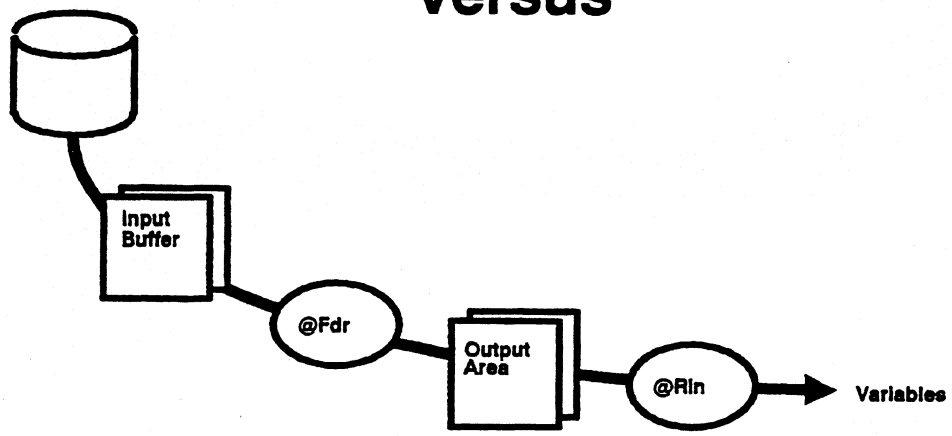
Counterpoint: There are none.

Note: Careful with FDR's syntax: @FDR,c,d,r,sl,ql,lab vs *@FND,c,d,r,sl,lab.*

FDR


Input
Buffer
@Fnd
Output
Area
Input
Buffer
@Rdl
Variables

versus


Input
Buffer
@Fdr
Output
Area
@Rln
Variables

Reading Lines

The following sections of code are an attempt to illustrate the differences between the various read line combinations.

The first block of code compares the FND/RDL to the FND/RDC combinations in a loop. Notice the difference between the amount of LLPs and DLPs.

The second block of code compares the FND/RDL to the FDR/RLN combinations in a loop. Notice the I/Os dropped proving that the FDR/RLN combo is more efficient.

Rule of Thumb: RDC processes less lines therefore it is faster. Use the FDR/RLN combination whenever possible to save I/Os.

Reading Lines

```
*=================================
@Fnd,0,b,5 " 'Product Type(1-5)' ,white  ,v1i3 .
@2:Rdl,0,b,5,v1,3 2-2 v2h2 .
v2
@Inc v1 gto 2 .
@3:Ldv,w v3i3=IO$, v4i3=LLP$, v5i3=DLP$ .
                IO"s = v3        47
                LLP"s = v4       2893
                DLP"s = v5       5389
@Gto end .

@Fnd,0,b,5 " 'Product Type(1-5)' ,white  ,v1i3 .
@Rdc,0,b,5,v1 2-2  v2h2 .
v2
@3:Ldv,w v3i3=IO$, v4i3=LLP$, v5i3=DLP$ .
                IO"s = v3        47
                LLP"s = v4       4
                DLP"s = v5       2501
@Gto end .
```

```
*=================================
@Fnd,0,b,5 " 'Product Type(1-5)' ,white  ,v1i3 .
@Rdl,0,b,5,v1 2-2 v2h2 .
v2
@2:Rln,,3  2-2 v2 .
v2
@Gto 2 .
@3:Ldv,w v3i3=IO$, v4i3=LLP$,v5i3=DLP$ .
                IO"s = v3        47
                LLP"s = v4       2892
                DLP"s = v5       5388
@Gto end .

@Fdr,0,b,5  " 'Product Type(1-5)  ,white  ,v1i3 .
@2:Rln,,3 2-2  v2h2 .
v2
@Gto 2 .
@3:Ldv,w v3i3=IO$, v4i3=LLP$, v5i3=DLP$ .
                IO"s = v3        37
                LLP"s = v4       2893
                DLP"s = v5       5389
@Gto end .
```

Writing Lines - WRLS

- Use WRL statement to write many lines at once rather than many WRL statements each writing one line.
- Multiple WRLs stay within buffer boundaries versus reading the WRL code many times and setting up buffer space each time.
- Accounts for direct savings in I/Os.

Point: Savings in terms of I/Os.

Counterpoint: When issuing a multiple line WRL statement, coding is tedious and maintenance becomes more difficult. Multiple line WRLs must be contiguous.

Note: Lock (@LOK) a report before issuing any WRL commands.

```
*=====================
@Lok,c,d,r
@Wrl,c,d,r,ln    n-n    lt,var .
```

versus

```
*=========================
@Lok,c,d,r
@Wrl,c,d,r,ln    n-n    lt,var/lt,var/lt,var/lt,var/lt,var . . .
                        (up to 23 lines)
```

Writing Lines - NOID

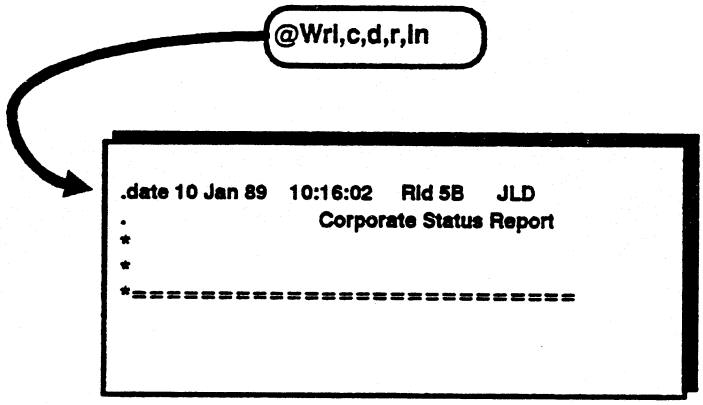
The WRL statement contains a subfield inhibiting the update of Line One/Date Line. Whenever an SOE update or WRL occurs on a report, the date, time and user-id are updated. This costs an extra write I/O.

By specifying a 'Y' in the NOID subfield of the WRL statement, the updates are inhibited and additional I/Os are eliminated.

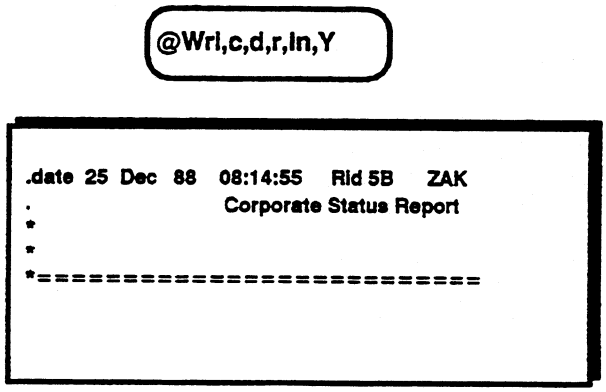
Point: Savings in terms of I/Os.

Counterpoint: Without the update of Line One, we lose track of when the updates occurred and who made them.

Note: The Auto Parameter, FRCUID, controls whether a 'Y' in the NOID subfield will work as expected.



versus



Exercises

Type in the following code sequences as three separate runs, @LOG, and:

- Make sure data report exists with at least 1600 lines of data
- Code RCR with @LOG as first statement
- Print out results

Discuss the comparisons and be prepared to offer explanations for the "whys".

```
*=========================
@Log .
@Rdc,0,b,5,6   1-80   v1s    .
v1
@Gto end .

@Log .
@Brk,,2000  .
@Rdc,0,b,5,6  1-80   v1s    .
v1
@Gto end .

@Log .
@Brk,,1500 .
@Rdc,0,b,5,6  1-80   v1s    .
v1
@Gto end .
```

5

Recovery Concerns

Module 5
Recovery Concerns

Objectives

Upon completion of this module the student should be able to:

1. Explain the differences between Purge, Cycle/Merge.
2. Recognize the ramifications of rid changes to the MAPPER system.
3. Explain various system recovery schemes.
4. Recover a run.

Recovery Tape

- 2 medias of recovery
  - Tape
  - Duplex Files

- Recovery Tape
  - All updates for a MAPPER session
  - How information is updated determines how quickly tapes fill

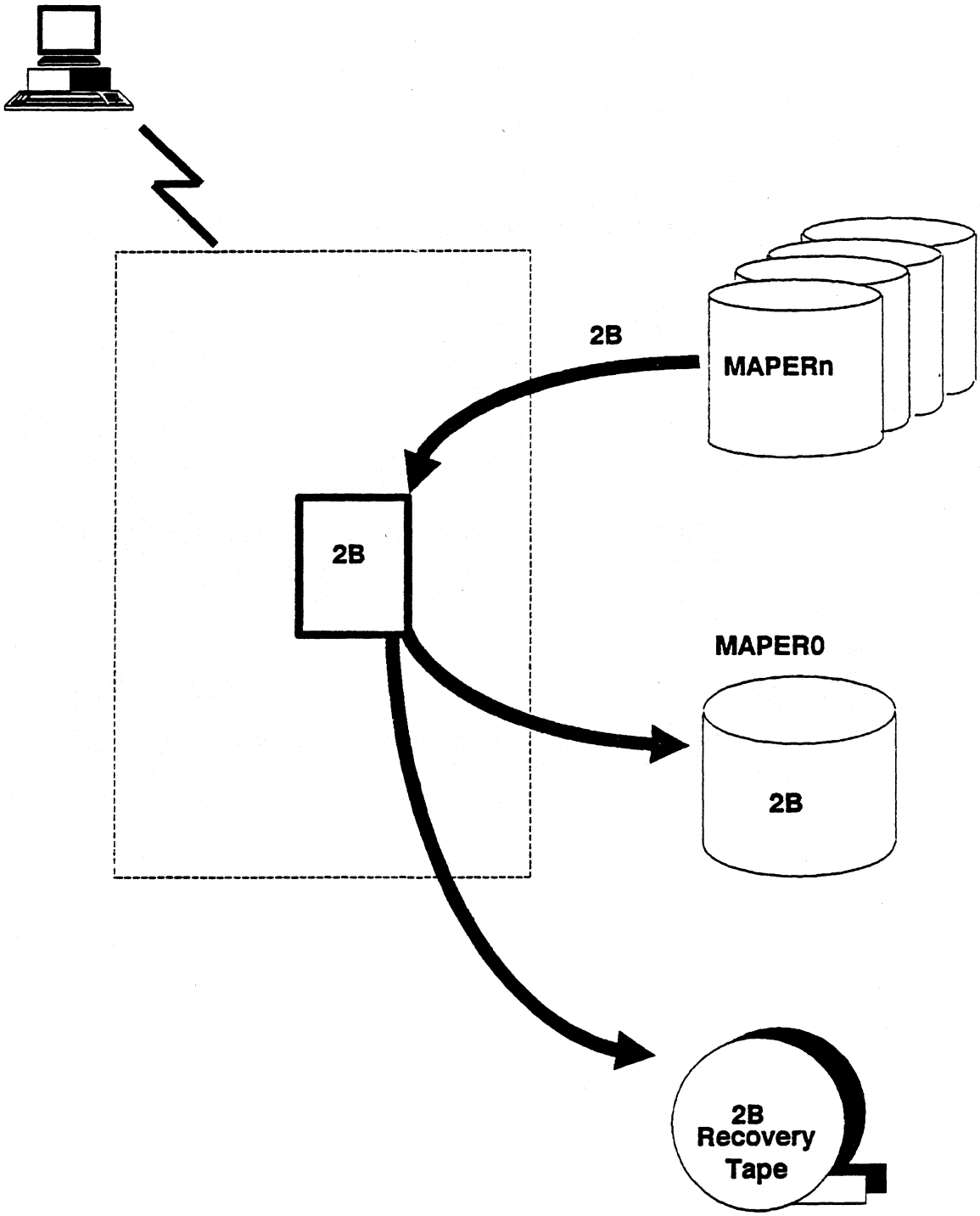
First update of 2B follows this path:

- Read/Retrieve 2B into MAPPER from the correct MAPERn file (I/O)
- Update 2B
- Write updated version to SPOS of MAPER0/UPOS of MUPERn (I/O)
- Write updated version to Recovery Tape (I/O)
- Subsequent updates of 2B are written to MAPER0/MUPERn (I/O) and to the Recovery Tape (I/O)
  - There is no need to retrieve 2B from MAPERn files because the latest version of 2B exists on MAPER0. It is the Purge or Cycle Merge processes that update the MAPERn files from MAPER0.

Any time an update occurs on a report/RCR, the updated report is written to:

- Disk
- MAPER0
- Recovery Tape

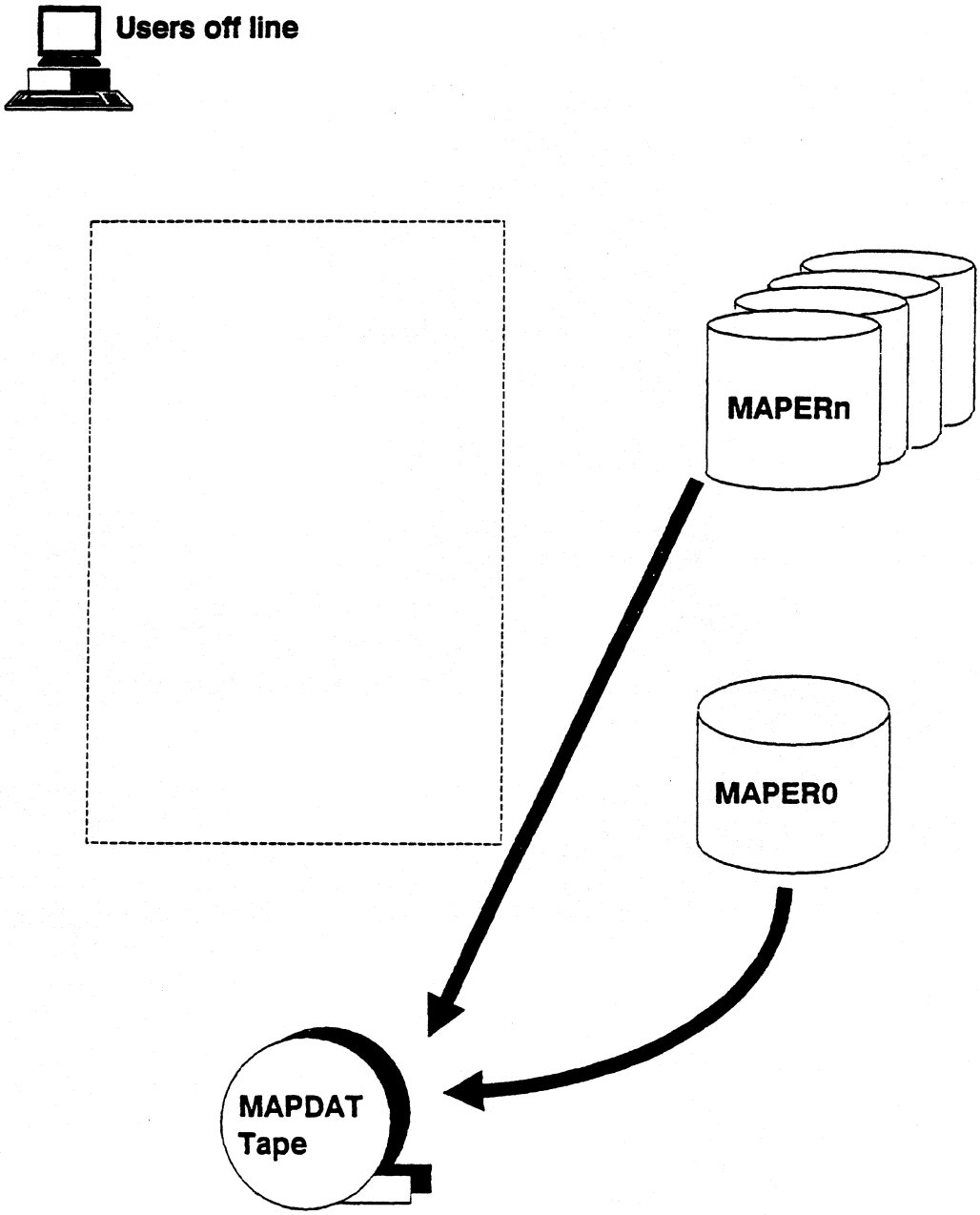
Recovery Tape



Purge Process

- All users are offline
- Idle/Active logos carry system messages indicating purge date and time
- Creates history tapes called MAPDAT tapes
- Procedure:
  - All terminals offline
  - Updated reports copied from MAPER0 to MAPDAT tape
  - Reports not updated are copied from MAPERn files to MAPDAT tapes
  - Packs MAPERn files
- PREMAP/PRESTR runstream executed
  - MAPDAT tapes are written out to MAPERn files
  - Database restored
  - MAPPER executed
  - System is up

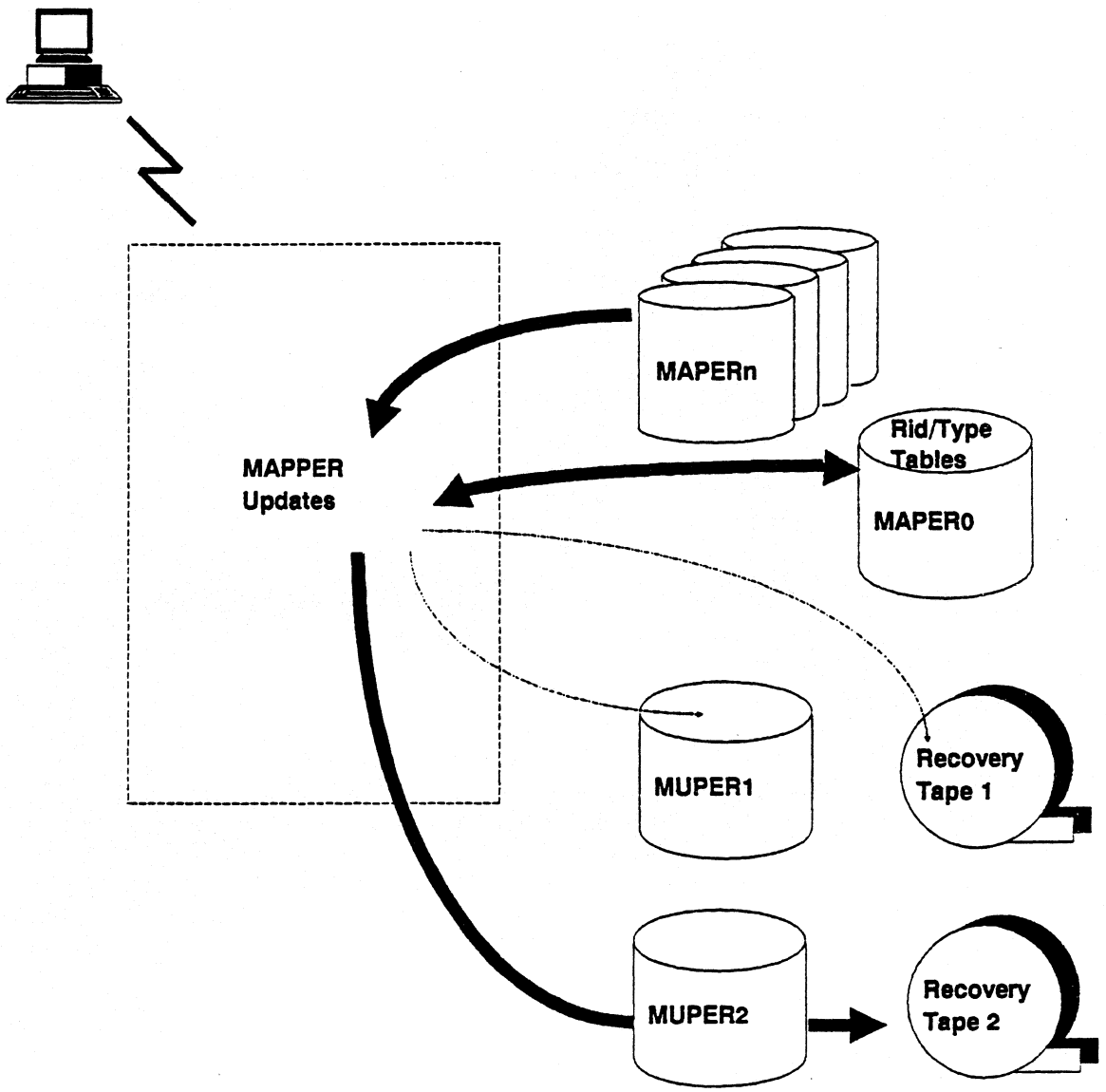
Purge Process



Cycle/Merge

- Installations running around the clock
- Dynamically updates and backs up database without taking the system down
- 2 files
  - MUPER1 and MUPER2 (SPOS of MAPER0)
    - Updated reports
- First update of the day
  - Read/Retrieve report into MAPPER from the correct MAPERn file
  - Updates report
  - Writes updated version to correct MUPER file and to Recovery Tape
- Subsequent updates pass through the MUPER files
- Cycle directive, issued by the operator:
  - Switches MUPER file pointer so that the second MUPER file accumulates updates
  - MAPPER switches Recovery Tapes, first tape is freed

Cycle

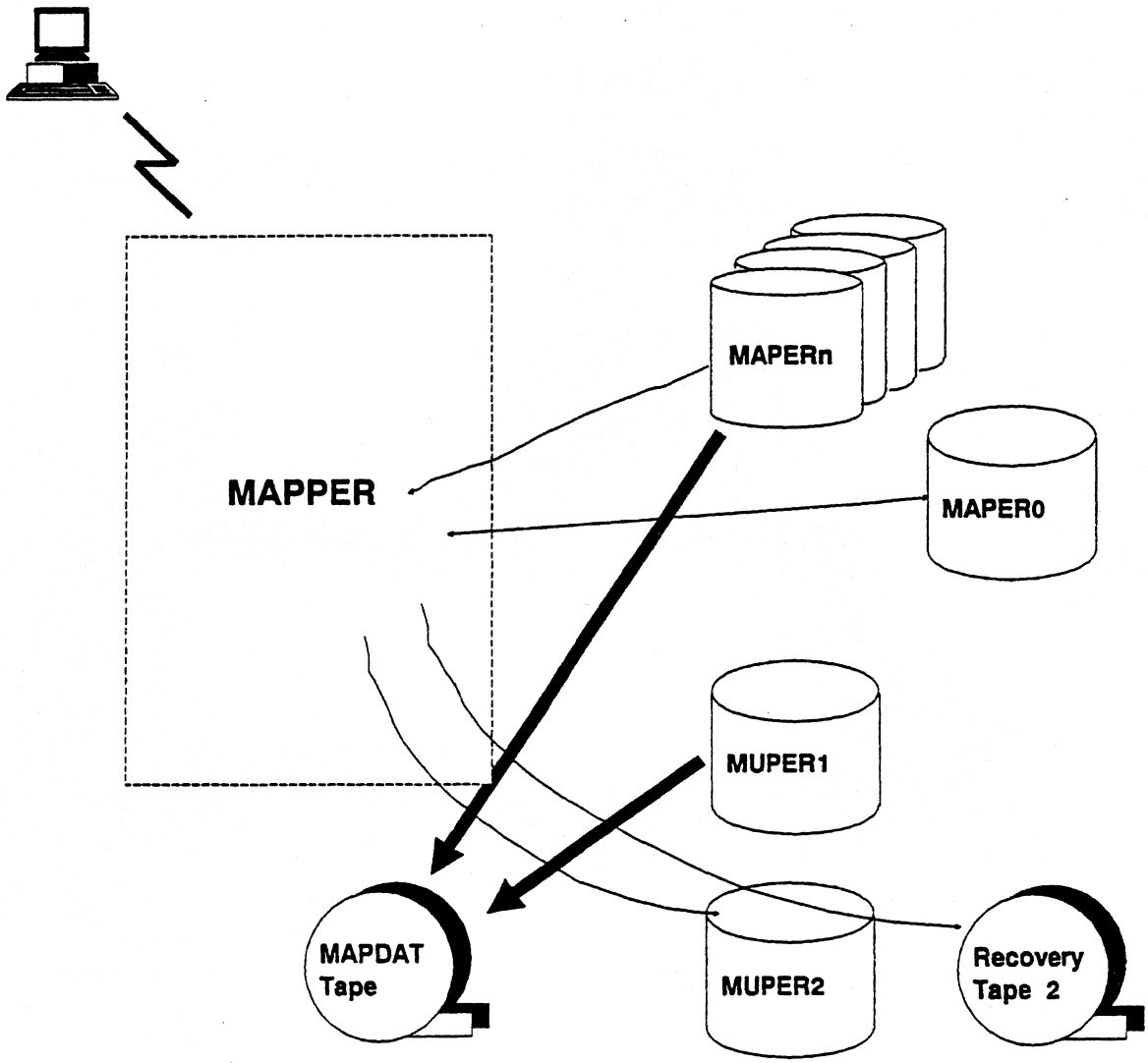

MAPERn
Rid/Type
Tables
MAPER0
MAPPER
Updates
MUPER1
Recovery
Tape 1
MUPER2
Recovery
Tape 2

Cycle/Merge

- **Operator initiates Merge function**
  - **Cycle function has completed**
- **Updated reports in the MUPER1 file are written to corresponding MAPERn files**
- **Database updated at the end of each Merge**
- **MUPER1 file cleared for the next Cycle**
- **MAPERn files written to MAPDAT Tapes**
- **Rid Tables in MAPER0 file updated, but files are not packed**

Note: Packing a file means to physically delete the data from the disk packs. Most delete functions do not physically remove the data. They simply flag that element for deletion. Furthermore, the Merge function can be configured as Merge and Secure - where Secure performs the actual write of the database on to tape. Or, the Cycle Merge keyin can be combined to perform both the Cycle and the Merge(CYCMRG).

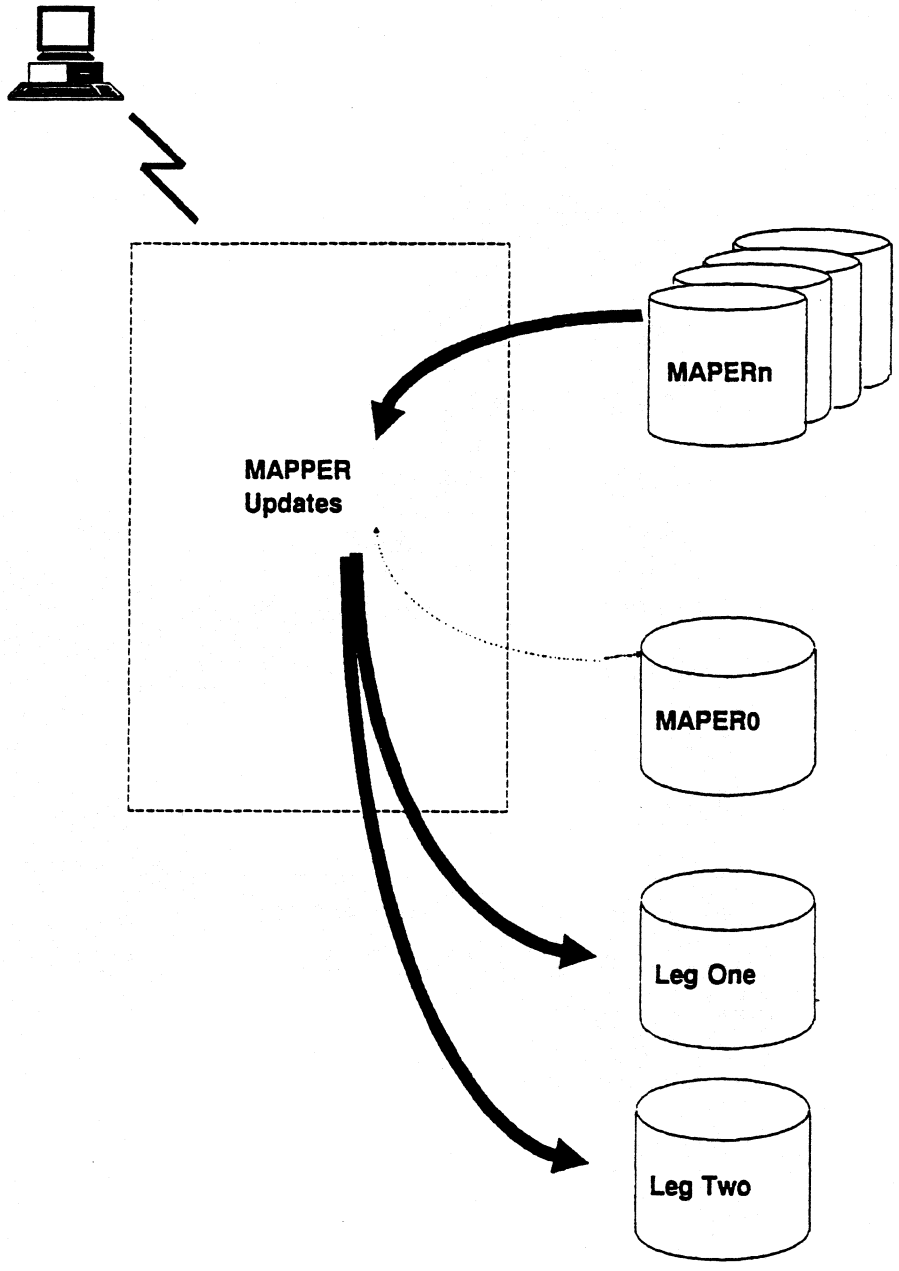
Merge


MAPERn
MAPPER
MAPER0
MUPER1
MAPDAT
Tape
MUPER2
Recovery
Tape 2

Duplex Files

- Duplicated files on mass storage
- Means of recovery
- Prevent down time for a system
- Each logical file has two physical files created referred to as legs
  - Write I/Os are doubled
  - Actual time not doubled - I/Os done in parallel
- Permits system to continue running should a disk crash
- May still run with a recovery tape

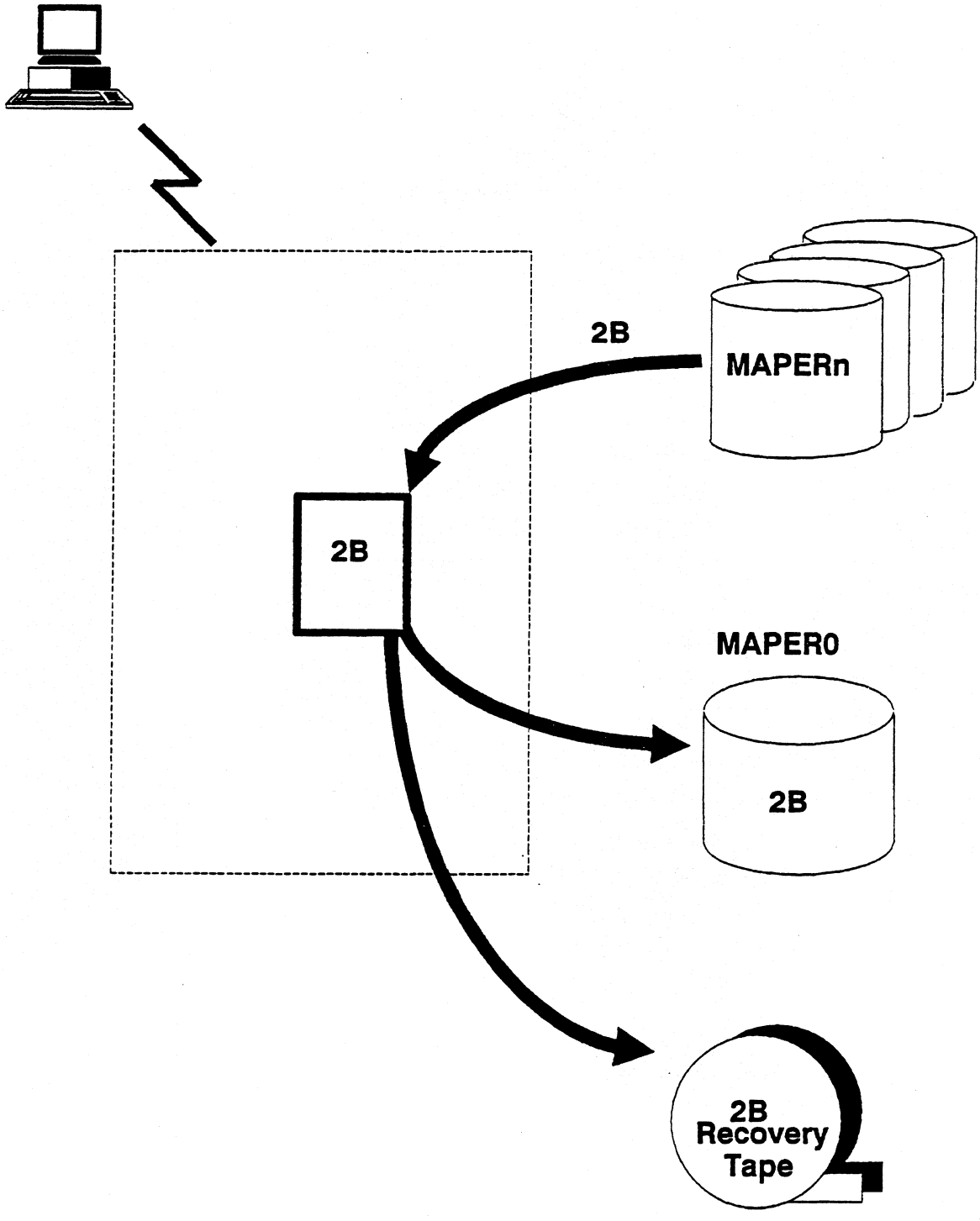
Duplex Files



Report Updates

- One line (small as one Buffer)
- Entire report written (change in report size)
- Update incurs a minimum of one additional I/Os
- Additional I/Os slow response time
- Update functions include:
  - First update of the day
  - Report manipulations - ADD, DUP, REP
  - Line manipulations - LN+,LN-,LNX ...
  - Update functions - MAU,SRU,CAU ...
  - SOE update or WRL
- Minimizing the number of updates issued to a report lessens recovery tape and MAPER0 impact - I/Os, response time, recovery time, and tape space

Report Updates



Updates

- Entire report is written to the recovery tape when an update changes the report size
- Keep reports at 500 lines
  - MAPPER allocates an initial blocksize of 500 lines for each report
  - Lines are written to the recovery tape
- Additional (LN +) and (WRL) updates impact MAPER0 and recovery tape
- Certain code sequences affect the recovery tape, MAPER0 and I/Os more than others.

Adding Data

- Placing new data at end of report
  - Lock report prior to update functions
    - Prevents occurrence of multiple updates
  - Find and capture last report line number
  - Increment
  - Add blank line
  - Write to it
- Minimum Cost in I/Os:

1 Find last line number

1 LN + to MAPER0/MUPER

1 LN + to recovery tape

1 WRL to MAPER0/MUPER

1 WRL to recovery tape

5 Total I/Os

Note: It would be better to add several blank lines versus a single blank line. Adding lines at the bottom of a report costs more - paging in and out of the Execution Window.

Adding Data

```
*==========================

@Lok,c,d,r
@Lln,c,d,r, <line> i4
@Ln+,c,d,r,<line>,1                    inc<line#>
@Wrl,c,d,r,<line>
```

```
*=====================

blank
blank
```

Adding Data

Maintaining blank lines at end of report.

- Use @FND vs @LLN to find first blank line
- Write to blank line
- Recovery Tape hit once
- Number of I/Os for Find increases as number of lines to process increases (Power Curve)
- No blank lines?
  - Modifying code to account for this situation
  - Find I/Os increase due to the added lines
  - Recovery Tape and MAPER0 I/Os increase due to LN + and WRL

Working within a buffer

- Find and capture last line of the report
- Decrement by 75 so that we remain in input buffer
- Issue FIND command to start search at decremented line number
- FIND captures blank line within the input buffer

Minimum Cost:

1 I/O to read last line number

1 I/O to FND

Recovery Tape and MAPER0/MUPER I/Os

Point:

I/Os remain constant

Number of lines to process doesn't increase (Power Curve) nor do the Recovery Tape, MAPER0 I/Os

No blank lines?

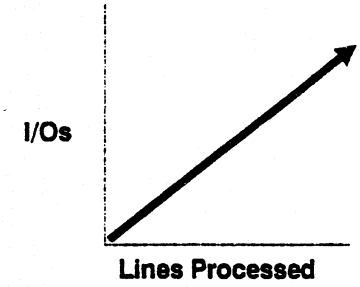
- Add one buffer full, 75 lines
- I/Os remain constant - Minimum of 1 additional I/O incurred due to the LN +

Problem - these two methods do not recoup blank lines throughout the entire run

Adding Data

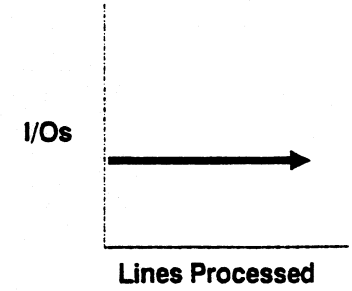
```
*=====================
 @Lok,c,d,r
 @Fnd,c,d,r ... <rid>,<line>
 @Wrl,c,d,r,<line>
```

```
*=====================
 @Lok,c,d,r
 @Fnd,c,d,r,,lin+1 ... <rid>,<line>  gto lin+2
 @Ln+,c,d,r,<line>,10                inc<line>
 @Wrl,c,d,r,<line>
```




```
*=====================
 @Lok,c,d,r
 @Lin,c,d,r,<line>                dec,75 <line>
 @Fnd,c,d,r,<line>... <rid>,<newline>
 @Wrl,c,d,r,<newline>
```

```
*=====================
 @Lok,c,d,r
 @Lin,c,d,r,<line>                dec,75 <line>
 @Fnd,c,d,r,<line>,lin+1... <rid>,<newline>
 @Gto lin+2
 @Ln+,c,d,r,<line>,75
 @Wrl,c,d,r,<newline>
```



Adding Data

Placement of new data not important

- Issue FIND to start at line six
- Blank line found, write
- No blank lines, push down, add one buffer full of blank lines and write
- This method recoups blank lines
- I/Os increase as the Find processes more data (Power Curve)

Cost:

1 or more I/Os on FND

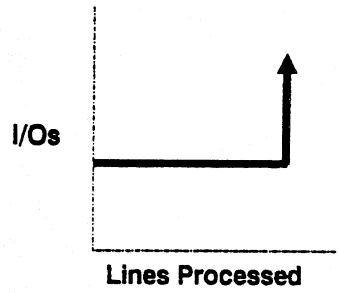
Recovery Tape and MAPER0/MUPER I/Os

Alternative: FDR function

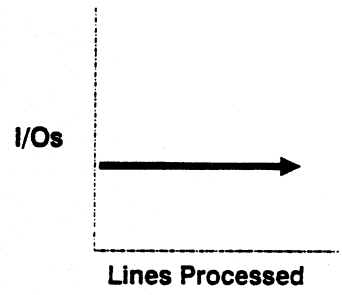
- Controls quantity of lines to search
- In one I/O
  - If blank is there
    - Writes
  - No blank lines
    - Add blank lines and write
- FDR processes the same amount of data
- Recovery tape, MAPER0 I/Os remain constant

Adding Data

```
*=====================
 @Lok,c,d,r
 @Fnd,c,d,r,6,001
 @Gto lin+2
 @001:Ln+,c,d,r,6,75
 @Ldv <line>i3=6
 @Wrl,c,d,r,6
```




```
*=====================
 @Lok,c,d,r
 @Fdr,c,d,6,75,label
 @Gto lin+2
 @Ln+,c,d,r          Ldv <line>i3=6
 @Wrl,c,d,r
```



Results - @RSL

- Solution to Recovery Tape impact
- Updates not recorded on the recovery tape
- Results recorded in MAPER0
- @RSL turns report into result
- Report remains intact while updates and manipulations occur to result form of report
- @RNM command permits up to eight results (-0 thru -7)
- Auto Parameter, MAXRNM, controls maximum number of renamed results allowed. Default is four.

Code sequence:

@LOK,c,d,r .

@RSL,c.d,r .

@WRL .

@LN + .

@REP,c,d,r .

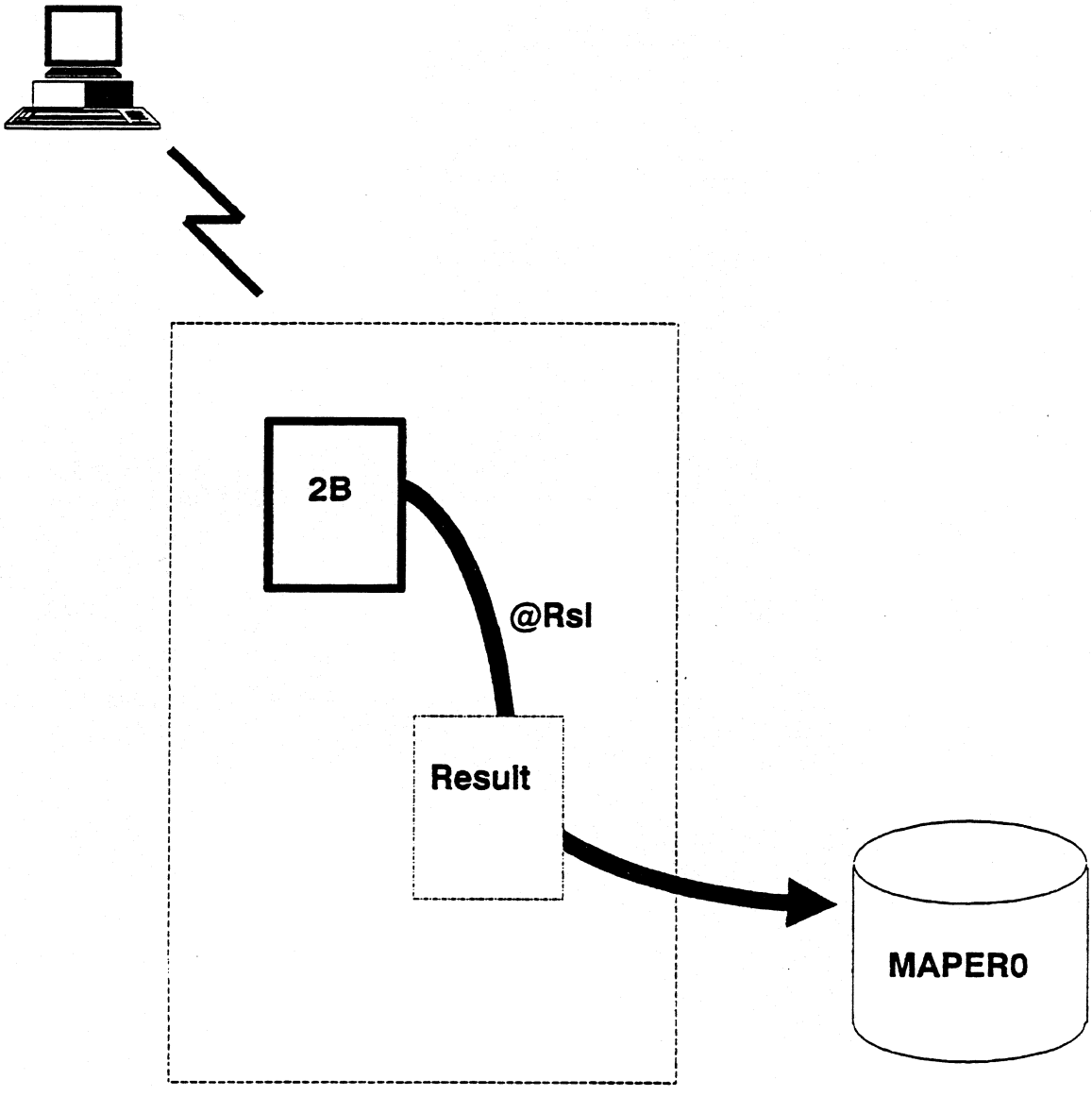
Minimum cost:

1 I/O for REP to Recovery Tape

1 I/O for REP to MAPER0/MUPER

Caution: Results are not recoverable

Results



Working Around Result Limitations - @RDC

- Used when Auto Parameter MAXRNM = 4
- Get around limited numbers of results

Example illustrates how to:

- Process several reports of data using the RDC command
  - RDC permits processing of specific line types
- Append that data

Code can be broken down into 4 sections.

1. The basic idea behind the first section of code is to "key" each line read. For example, if a line of data was read from a B drawer report, then that line is "keyed" with a B in column one. If from a C drawer report, then a C in column one is keyed.
Once the data lines have been "keyed", they are placed into -1.

2. Appending B Key type lines - This section reads this -1 result looking for B type lines and then appends this data to report 3B with the @ADD command.

3. This section does nothing more than read -1 result looking for C type lines.

4. This section illustrates yet another alternative to the @ADD function. A SRU is done against 2D for all blank lines. The result from the SRU is renamed to -2. The -1 result is read in search of all D type lines.

The lines of -1 and -2 are @ADD'd. To complete the loop, a UPD is issued and those lines are now permanently added to report 2D at the blank line.

Working Around Result Limitations @RDC

```
*==============================================
@Brk   Rdl,0,b,2,15  2-79  v131s   . Key line types
Bv131
@Rdl,0,c,2,15   2-79   v131s    .
Cv131
@Rdl,0,d,2,15   2-79   v131s    .
Dv131
@Brk   Rnm  -1   Dsp,-1   . Break and Rename the result -1


@Brk,0,b   . Clear out -0
@Rdc,0,e,-1,2,,b,5  2-79   v131s    . Read Continuous -1 for B type lines
' 'v131
@Brk   Add,0,b,-0,0,b,3    Dsp,-0   . Take read data and append


@5:Brk,0,c   . Clear out -0
@Rdc,0,c,-1,2,,c,6  2-79   v131s    . Read Continuous -1 for C type Lines
' 'v131
@Brk   Dsp,-0  . Break and Display


@6:Brk,0,d   . Clear out -0
@Lok,0,d,2   .
@Sru,0,d,2   ab(@)  2-1   ,@   Rnm  -2   Dsp,-0   . Search update for blank lines
@Rdc,0,e,-1,2,,d,7   2-79   v131s    . Read Continuous -1 for D type lines
' 'v131
@Brk   Add,0,d,-0,0,d,-2   Dsp,-0  . Take -1 and -2 data and append
@Upd   Ulk    Dsp,0,d,2,99  . Update , Unlock and display
@7:Gto  end   .
```

Working Around Result Limitations - Keying Line Types (@CAL)

1. Sort data by Customer Code field. V1 and V12 are then loaded with a Tic and a numeric value of one.

2. @CAL statement uses - If conditional and "keys" each line with a numeric (1 through 6) according to the Customer Code field. -0 result is renamed to -1.

3. Using the RDC statement, each line type (V12) is processed and placed into a result and displayed.

4. Once V12 equals 6 , quit.

Working Around Result Limitations @CAL

```
*==========================================
@Sor,0,b,2  "   'CustCode'    ,1 .
@Ldv,w  v1a1=Tic$,v12i1=1  .
@Cal,-0  "  45-4   ,a  if:a=v1amcov1;then:lt=1;if:a=v1arco;\
then:lt=2;if:a=v1dicov1;then:lt=3;if:a=v1fedsv1;then:lt=4;\
if:a=v1intrv1;then:lt=5;if:a=v1usscv1;then:lt=6 .
@Rnm  -1   Dsp  -1 .
@1:Rdc,0,b,-1,6,,v12  1-80  v10s80 .
v10
@Brk  Dsp,-0,,,,y .
@If  v12 = 6 gto 2 ; inc v12  gto 1 .
@2.
  stop run
@Gto end .
```

Run Recovery - Checkpoint Reports

- Maintains status of run in execution at certain logical points
- May contain any kind of information: run documentation, variables used, error dump routine
- Must be maintained
  - A consideration must be made to the cost of updating the Checkpoint Report during normal operations vs. the likelihood of having to utilize it to recover data.
- Should be "paired" with the RCR's report number but should reside in another type
  - The RCR resides in report 13E; therefore, the checkpoint report resides in report 13F.

Checkpoint Reports

```
. date 10 Jan 89    10:16:02          Rid 13E      JLD
*==================================================
@45: Task One
     Editing Sort
@Rep

@60: Task Two
     Updates to data
@Upd

@80:  Task Three
     Account Balances
@Rep

@125: Task Four
     Print the data
@Gto end
```

```
. date 10 Jan 89    10:16:04          Rid 13F      JLD
* Task        Start      Start      Recv      Rids
* Number .   Date   .   Time   .   Labl .   Beg  .End.  Mod   .Type.
*==================================================
    1        881114     093614      45       18
    2        881114     094834      60       18
    3        881114     100502      80      140  143
    4        881114     102042     125       60

. STATUS: (In Process, Completed)    Completed, 11 Nov 88, 102549
. Keys for Recovery:
```

Checkpoint Reports

Example - RCR is divided into four tasks:

- Each task ending in a "non-resultant" function (IE. UPD,REP)
- As task completes, time and date stamp for each new task is updated
- Recovery label is also noted in the Checkpoint Report along with the reports that were executed in that task
- These stamps are a good indication of whether a run terminated normally or prematurely due to system failure.The status, date and time a run began executing is refreshed each time the run is executed after a successful completion
- A task oriented recovery scheme is more efficient than a line oriented scheme
  - In a line oriented scheme each update is logged separately versus the updates being incorporated all at once at the end of each task

Caution: In the case of background runs, watch the time window for this process. If a task completes and the system goes down before the date and time stamps for the next task are updated, it will be difficult to determine exactly where the run left off.

Checkpoint Reports

```
. date 10 Jan 89    10:16:02                Rid 13E        JLD
*==============================================
@45: Task One
    Editing Sort
@Rep

@60: Task Two
    Updates to data
@Upd

@80:  Task Three
    Account Balances
@Rep

@125: Task Four
    Print the data
@Gto end
```

```
. date 10 Jan 89    10:16:04                Rid 13F        JLD
* Task        Start       Start       Recv       Rids
* Number .    Date   .    Time   .    Labl .    Beg   .End.   Mod   .Type.
*==============================================================
    1        881114      093614       45        18
    2        881114      094834       60        18
    3        881114      100502       80        140    143
    4        881114      102042      125        60

. STATUS: (In Process, Completed)   Completed, 11 Nov 88, 102549
. Keys for Recovery:
```

Checkpoint Reports

Example:

- Each task's date (**START DATE**) and time (**START TIME**) stamps are checked to see if they have been updated/completed.
- The date and time stamps in each data report (**RIDS BEG END**) involved are checked to pinpoint exactly where the system failure occurred.

Problems:

- Difficulty determining what point to restart run
- Too many updates to trace
- Need to restore all involved reports to original state prior to run
    - The @REH retrieves the original report from the database without any updates since the last Purge or Merge. Simply re-execute the run from the beginning.

What if a Checkpoint Report shows all zeroes in every start date and time stamp for each task?

- This means the run may have been in the PreRun process when the system failed, but the first task never started. Therefore, to recover the run, restart it from the beginning.

All Zeroes

. date 10 Jan 89 10:16:04 Rid 13F JLD

* Task Number .	Start Date .	Start Time .	Recv Labl .	Rids Beg	.End.	Mod	.Type.
1	000000	000000	45	18			
2	000000	000000	60	18			
3	000000	000000	80	140	143		
4	000000	000000	125	60			

. STATUS: (In Process, Completed) (status), (date), (time)
. Keys for Recovery:

Checkpoint Reports

The following Checkpoint Report contains updated date and time stamps for the first two tasks only, implying the system failed somewhere in the middle of task 2. Perhaps task 2 made it to completion, but the run terminated before task 3 started.

To pinpoint exactly where run should be restarted:

- Compare date and time stamp of task 2 with its associated updated report, report 18.
  - Date and time stamp on the date line of report 18 is greater than date and time stamp of task 2 in the Checkpoint Report
    - Report 18 was successfully updated
    - Run should be restarted at the beginning of task 3, recovery label (RECV LABL) 80.
  - Date and time stamp of report 18 is less than date and time stamp of task 2
    - System failed somewhere in the middle of task 2
    - Run should be restarted at recovery label 60.

In The Middle

```
. date 14 Nov 88     10:16:04              Rid 13F          JLD
* Task        Start       Start      Recv      Rids
* Number .   Date   .   Time  .    Labl .   Beg  .End.  Mod  .Type.
*=========================================
   1        881114     093614      45      18
   2        881114     094834      60      18
   3        000000     000000      80     140  143
   4        000000     000000     125      60

. STATUS: (In Process, Completed)   In Process,14 Nov 88, 093550
. Keys for Recovery:
```

```
. date 14  Nov 88    10:00:55          Rid 18B        JLD
* Last Name          . Street Address              . Phone
*=================================
 mullen              123 south                     555-1212
 tattershall         426 waverly                   876-9123

                  ***** End Report *****
```

versus

```
. date 14 Nov 88     10:16:04              Rid 13F          JLD
* Task        Start       Start      Recv      Rids
* Number .   Date   .   Time  .    Labl .   Beg  .End.  Mod  .Type.
*=========================================
   1        881114     093614      45      18
   2        881114     094834      60      18
   3        000000     000000      80     140  143
   4        000000     000000     125      60

. STATUS: (In Process, Completed)   In Process,14 Nov 88, 093550
. Keys for Recovery:
```

```
. date 28 Oct 88     13:12:55          Rid 18B        JLD
* Last Name          . Street Address              . Phone
*=================================
 mullen              123 south                     555-1212
 tattershall         426 waverly                   876-9123

                  ***** End Report *****
```

Checkpoint Report

If every date and time stamp for a task has been updated, but the status at the bottom of the Checkpoint Report still reflects that the run is in process, has the run completed or is it still somewhere in the middle of task 4?

In this case:

- Compare date and time stamp of task 4 with date and time stamp of its associated report, report 60.
- If date and time stamp of report 60 is greater than that of task 4, the report has been successfully updated and the system failed before the run status changed. No recovery is necessary.
- If date and time stamp of report 60 is less than that of task 4, the run should be restarted at recovery label 125.

The End

```
. date 14 Jan 88    10:16:04          Rid 13F          JLD
* Task      Start      Start      Recv      Rids
* Number .  Date   .   Time  .   Labl .  Beg  .End.  Mod  .Type.
*=========================================
   1       881114    093614     45     18
   2       881114    094834     60     18
   3       881114    102630     80     140  143
   4       881114    110243    125     60

. STATUS: (In Process, Completed)      In Process   14 Nov 88, 093550
. Keys for Recovery:
```

```
. date 14 Nov 88 11:32:55          Rid 60B      JLD
* Last Name        . Street Address          . Phone
*=================================
 mullen             123 south                555-1212
 tattershall        426 waverly              876-9123

                  ***** End Report *****
```

versus

```
. date 14 Nov 88    10:16:04          Rid 13F          JLD
* Task      Start      Start      Recv      Rids
* Number .  Date   .   Time  .   Labl .  Beg  .End.  Mod  .Type.
*=========================================
   1       881114    093614     45     18
   2       881114    094834     60     18
   3       881114    102630     80     140  143
   4       881114    110243    125     60

. STATUS: (In Process, Completed)   In Process,14 Nov 88, 093550
. Keys for Recovery:
```

```
. date 28 Oct 88 13:12:55          Rid 60B      JLD
* Last Name        . Street Address          . Phone
*=================================
 mullen             123 south                555-1212
 tattershall        426 waverly              876-9123

                  ***** End Report *****
```

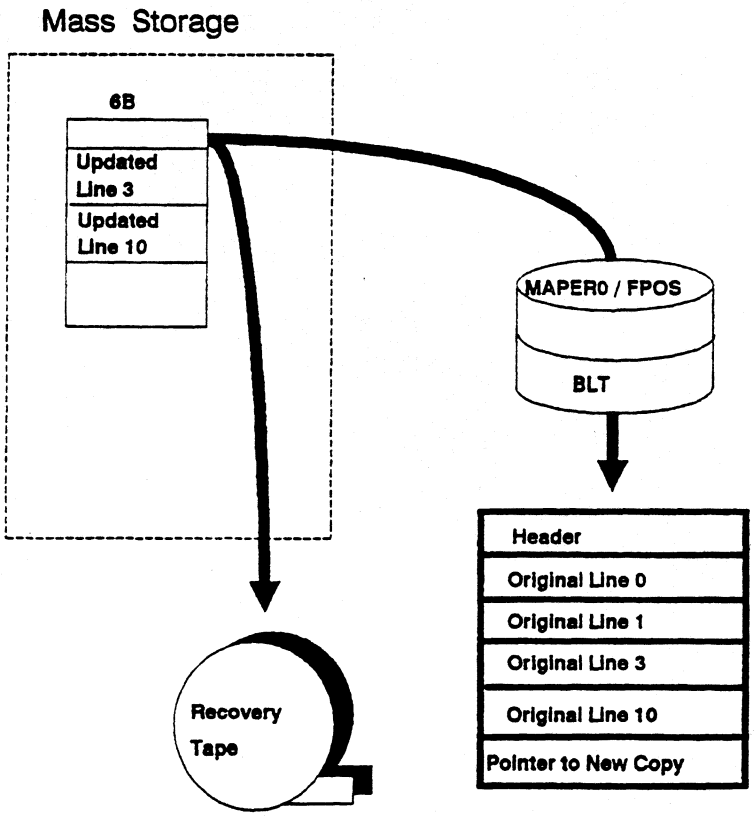
Run Recovery - Deferred Updates

- Add update security against abnormal run terminations or system crashes
- "All or none" type of updating
- Permits us to defer updates for as many as ten reports
- Label used to jump to if one of the reports is currently locked
- Updates to reports not made until an @CMU (Commit Update) command is encountered
- If deferred updates do not occur, @DCU (Decommit Update) is issued
- Following commands cannot be executed between an @DFU and @CMU/DCU combination: @DFU, DSP, OUM, OUT, REL, RTN, RUN, WAT
- Can be expensive in terms of Maper0 and recovery tape impact
- Used prior to @RSL command
- Before Look Table resides on Maper0
  - Original lines 0 and 1
  - Original lines that change in a reports under deferred update cabinet
  - Information about how to reconstruct the original report if an @DCU is issued
- Efficiency Hints
  - Keep reports small
  - Keep the amount of time a report is locked as short as possible
  - When performing several WRL statements, create a result of report, update, and replace report with result before @CMU

Deferred Updates

The following code depicts a very simple deferred update example. The user wants to delete all GREENBOX1's from Reports 2B and 1C. They also want to make sure that either both reports are updated or neither is updated.

```
*=========================
@Dfu,10  0,b,2,0,c,1   . defer updates
@Sru,0,b,2,6,,99  dh  15-9  ,greenbox1  del .
@Sru,0,c,1,6,,99  dh  2-9  ,greenbox1  del .
@Cmu . commit updates
@Gto end .
@10 .
    Both reports are not available
@Gto end .
```

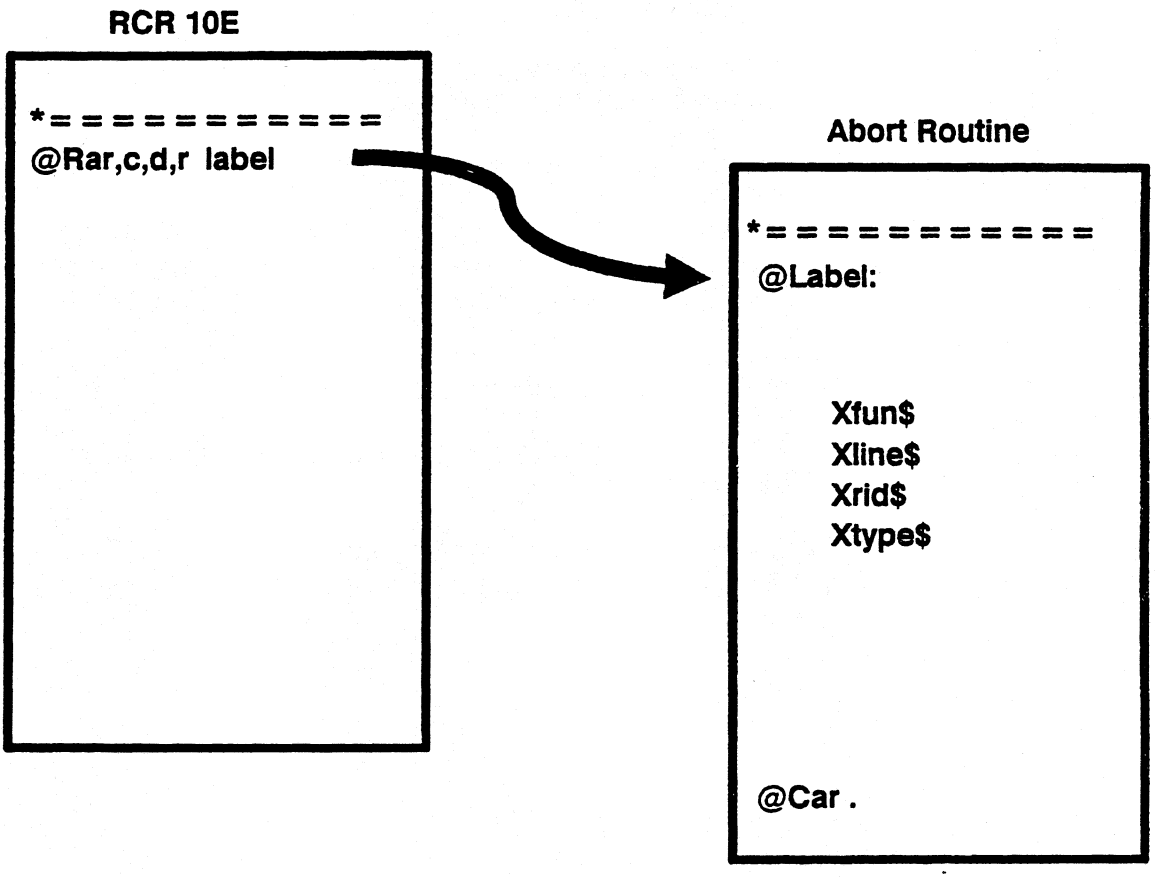


Abort Routines - RCR aborts

- Hitting F4 key under CMS
- MSG WAIT key
- @RAR statement
  - Registers subroutine to be executed in event a user deliberately aborts run
  - Internal or external
  - Subroutine must end with @CAR (Clear Abort Routine)
    - Breaks connection to originating run and subroutine
- Abort routines
  - Pass control to the run instead of to MAPPER
  - Following reserved words may be loaded during an abort

XFUN$	Last function executed before abort
XLINE$	RCR line number when aborted
XRID$	RCR report number when aborted
XTYPE$	RCR type number when aborted

Abort Routines



Error Routines

- **Handling recoverable and nonrecoverable errors**
  - Recoverable error
    - Control may be passed back to a particular point in the run to continue processing
  - Nonrecoverable error
    - Control cannot be passed back to run and run must terminate in error (reading a line from a report that doesn't exist, or a syntax error within run statement itself)
  - No error routine within run
    - MAPPER's internal error handling sends an error message to the run
  - Message varies depending on the nature of error and whether user of run is a registered run designer

Error Routines

Not a Run Designer:

"Run Erring Contact the Run Designer"

Run Designer but not Last Updater:

"Detail Error Message"
"Max I/Os Exceeded"

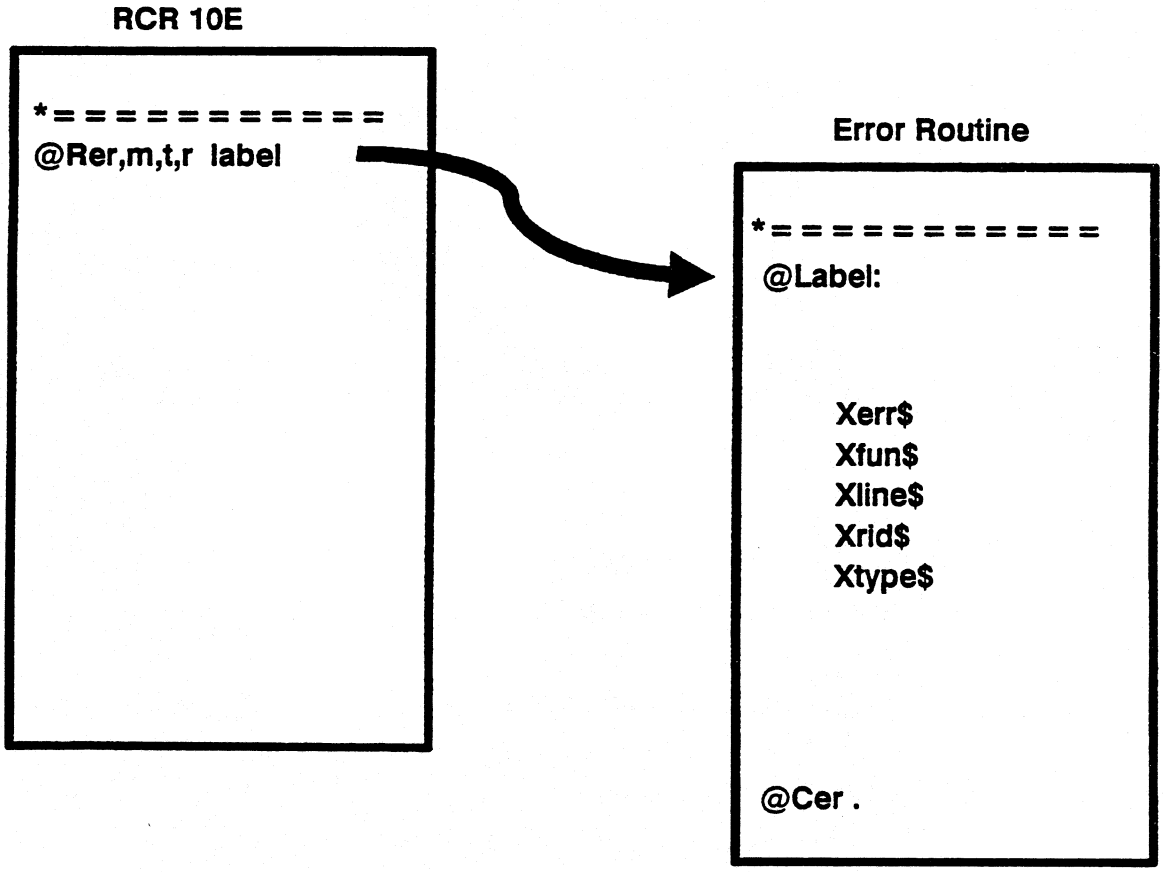
Run Designer and Last Updater:

RCR on display at line with error and
"Detail Error Message"

Error Routines @RER (Register Error Routine)

- Permits user to register a subroutine to be executed if run encounters an error
- Internal or external
  - External subroutine must be in same character set as the RCR
- Upon return to RCR, run must encounter an @CER (Clear Error Routine) which clears the connection and reverts back to MAPPER's internal means of handling errors
- Error routine should:
  - Display an error message to the user
  - Document pertinent information about run for subsequent error detection by the designer of the run such as
    - Function and line that erred
    - Variable contents
    - Time of error
    - Station and user executing the run
    - Output area contents
    - Contents of result(s)

Error Routines @RER



Error Routines - Nonrecoverable Error

Example:

- First run statement registers an internal error subroutine
- Error routine is executed starting at label 10

The following reserved words are loaded and should be used in an Error routine:

XERR$	Error number
XFUN$	Function name that erred
XLINE$	Line number in RCR where run erred
XRID$	Report number of RCR where run erred
XTYPE$	Numeric form drawer of RCR where run erred

Note: The LSM (Load System Message) statement converts the error message number (XERR$) to the system message and loads the message into a variable.

- Error Routine
  - Report 2E of Cabinet 0 in Limited Character Set (LCS)
  - Report 2F of Cabinet 0 in Full Character Set (FCS)
  - Provides:
    - Run name, error message, cabinet, drawer and report of RCR
    - User id of user executing the run
    - Section of the run that erred
    - Variable definitions for V1 through V189
    - First 20 lines of the output area, and -0 through -4

Nonrecoverable Error

```
*==========================================
@Rer 10 .
@Srh,0,b,2   'St Cd'  ,or .
@Dsp,-0 .
        .
        .
@10:Ldv,w  <err>i3=xerr$,<func>h3=xfun$,<rept>i3=xrid$,\
 <line>i3=xline$ .
@Lsm,<err>  <msg>s80 .
```

This run (rid - <rept>) has erred on line <line>, while executing the <func> function.

The following error message was received -
<msg>

```
*==========================================
```

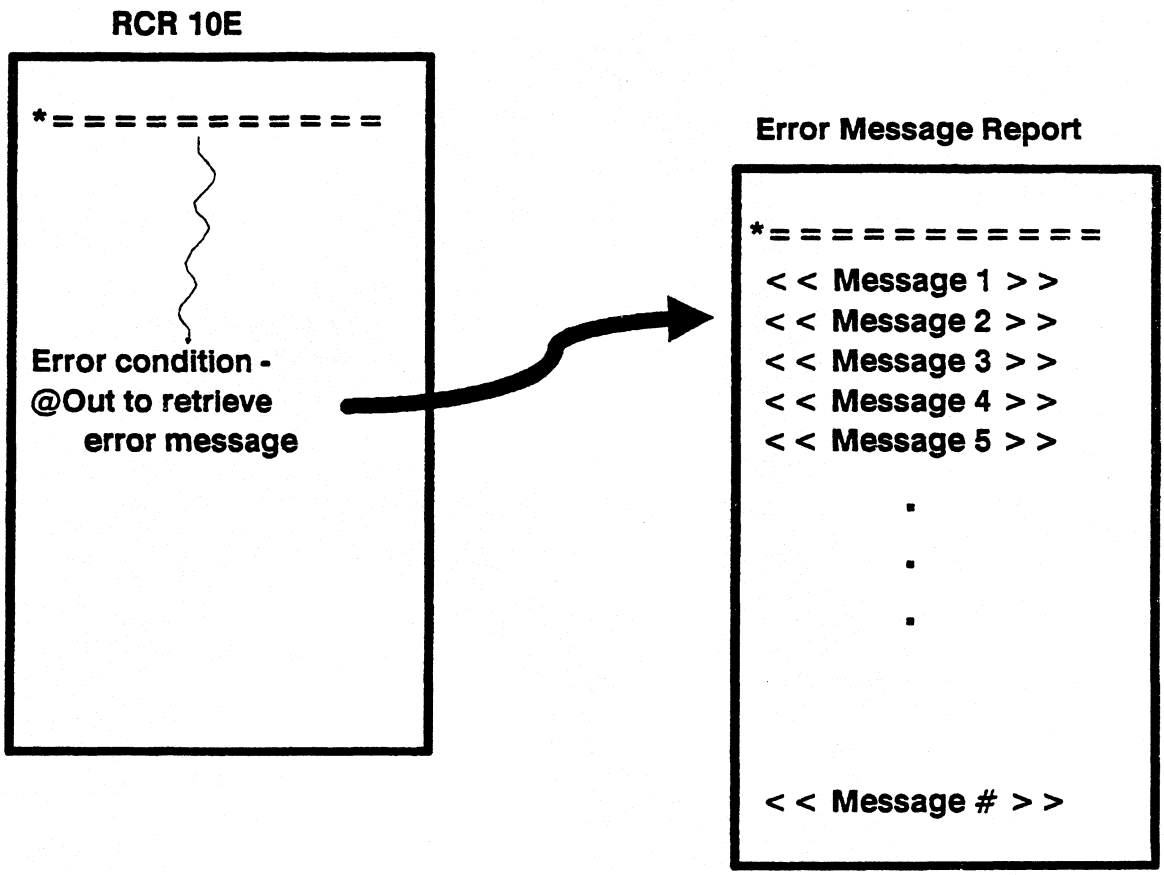
This run (rid - 6) has erred on line 5, while executing the SRH function.

The following error message was received -
<< This field must be a numeric field >>

Error Messages

- Cabinet 244 Report 1B
- Error message reports provide a centralized location for messages shared by a number of runs
- Alternative is to include error messages within the RCR
  - Accessing them with an @OUT statement
  - Using an Error message report reduces number of lines within RCR

Error Messages



Error Routines

- @OUT statement is initialized to directly read a line from the Error message report
  - Advantages for this method:
    - One OUT statement accommodates many error messages
    - Error messages are all in one centralized location, allowing for easier maintenance
    - It is more efficient than having an error section in the RCR, based on line changes to the report
    - Does not affect current result (-0)
  - Disadvantages for this method:
    - Error message report is line sensitive. Therefore, one change to the report will affect one or more runs
    - Variable data in error messages are not allowed
      Lines in the Error message report do not go through Execution Window; therefore, variables are not initialized with data. Remedy this problem with an @RSR statement accessing the Error message report
    - Additional I/Os are incurred
  - Keep error messages within Execution Window
  - Trade offs must be weighed (ie. how often will the errors occur)

Note: Output screens may also be contained in a separate report to provide a common library of often used screens. The same advantages and disadvantages apply as with the above.

Error Message Reports

```
* Error Message Report
*==================================
 << Invalid date - Reenter >>
 << Invalid status code - Choices are IP,OR,SH, SC >>
 << Box is wrong color - Choices are Green or Black >>
                    .
                    .
                    .
                    End Report
```

```
   Run Control Report
*==================================================
@Brk .
      Enter the following -
                Status Code
                Status Date
                Product Type
@Brk .
@1:Out,16,e,-0,2,9,1,1 .
@Chg input$ v1a2,v2i6,v3a9 .
@If v1 ne IP & ne OR & ne SC & ne SH ldv v10i2=7 gto 2 .
@If v2 not = ... a valid date ... ldv v10i2=6 gto 2 .
@if v3(1-5) ne GREEN & ne BLACK ldv v10i2=8 gto 2 .
                .
                .
                .
@2:Out,16,a,150,v10,1,20,,,y .
@Wat 5000 .
@Gto 1 .
@Dsp,16,b,-0 .
```

Exercise

1. All _____ are written to the recovery tape. If these _____ change the size of a report, the _____ is written to the recovery tape.
2. Duplicate report 3B of the demonstration database and:
   a. Write a run to perform a number of updates to the report. Defer the updates using the @DFU statement. Execute the run, and obtain a RUNA result.
   b. Repeat 2a deferring updates by first locking the report, turning the report into a result, then making updates to the result. Then replace the result back into the original report. Execute the run and obtain a RUNA result.
3. How do these two methods compare?
4. Write a simple run to search Report 2B in the demonstration database for a specific code and sort the result by status date. Solicit from the user a valid status code.
   a. Include an error message to the user if no data is found, and allow the user to reenter a valid status code. In addition, write and register an error routine to handle syntax errors within the run. In this routine, capture information about the run, such as the line number at which the error occurred, and the name of the function in error.
   b. Write and register an abort routine to handle deliberate run termination by the user.

6

Modular Run Structure

Module 6
Modular Run Structure

Objectives

Upon successful completion of this module you should be able to:

1. Be aware of internal label tables.
2. Build a Label Table for a run and understand its efficiencies.
3. Utilize various branching and subroutine methods.
4. Define constants for run maintenance purposes.

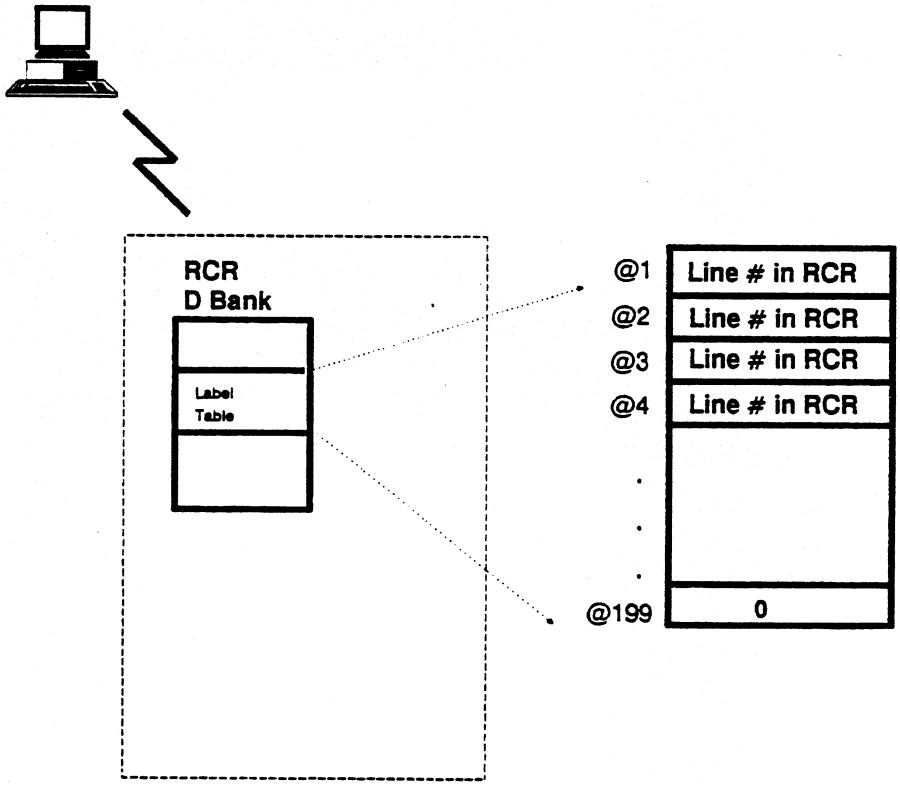
Label Table

- Functionally similar to Variable Table
- Every run has its own Label Table
  - Resides in the RCR's D Bank
  - Indexed by label number
  - Order of labels is insignificant
  - Undefined labels contain a 0 in label table entry
- Size is controlled by LABNUM
  - LABNUM default value = 199
  - LABNUM maximum = 399
  - As labels are encountered during run execution, line numbers are entered into table
  - Table expands if LABNUM is increased

Label Table

When a reference is made to a label during run execution MAPPER:

- Scans Label Table to locate line number on which label is defined
- If a line number exists in label entry
  - MAPPER determines if line is currently in Execution Window
    - Yes - Control passes directly to that line
    - No - Cost one I/O to bring that line (plus 77) into Execution Window
- If MAPPER cannot find a line number in its Label Table
  - MAPPER will scan the entire RCR until the label is found
    - Yes - Label is found, MAPPER updates Label Table with associated line number.
    - No - Label not found, an appropriate error message is displayed.
  - Any other labels encountered are updated in the Label Table.
- A great number of I/Os can incur due to paging in and out of Execution Window.



BLT Run Reviewed

- Creates label table in RCR
- BLT entered on line 0 of RCR
- Produces a result
  - Must use REP to make permanent
- Table consists of
  - List of labels used in RCR
  - Line number on which they are located
- Labels are listed in sequential order
- Table is on colon-type lines
  - These lines do not go to the output area
- Improves run efficiency
  - Table begins after the divider line
  - First line(s) of code processed
  - MAPPER knows immediately where labels exist
- Must build new table if line changes are made to RCR

Build Label Table Run

- Sample RCR with BLT entered on line zero

```
LINE▶ BLT    FMT▶    RL▶ -      SHFT▶      HLD CHRS▶    HLD LN▶        ▶   30E0      ▶
.DATE 21 AUG 90  10:07:58  RID      30E   21 AUG 90  WEBMJG
.FOOD                                                 BY: I.M. HUNGRY
*=================================================================================
@BRK LDV,W <SOE>A1=SOE$,<COUNTER>A1=0 .
@001:

      PLEASE MAKE YOUR LUNCH SELECTION:

   <SOE> PIZZA[        <SOE> BURGER[       <SOE> TACO[       <SOE> SALAD[
@BRK OUT,-0,2,4,,1 .
@002:IF CURH$ = 12 LDV <MESSAGE>$19='WITH PEPPERONI ONLY' GTO 006 ; .
@003:IF CURH$ = 28 LDV <MESSAGE>$21='MAKE IT A CHEESBURGER' GTO 006 ; .
@004:IF CURH$ = 41 LDV <MESSAGE>$20='EAT A MINT WHEN DONE' GTO 006 ; .
@005:IF CURH$ = 55 LDV <MESSAGE>$17='WHAT A HEALTH NUT' ; .
@006:INC <COUNTER> .
                    <MESSAGE>
@BRK OUT,-0,2,2,1,,,Y .
@WAT 5000 .
@IF <COUNTER> < 2 GTO 001 ; .

                    YOU ARE ONLY ALLOWED 2 SELECTIONS.......
                     ..... END REPORT .....
```

- Result contains label table

```
line▶ 1     fmt▶    rl▶ -     shft▶     hld chrs▶    hld ln▶    undo▶    RESULT    ▶
.DATE 21 AUG 90  10:07:58  RID      30E   21 AUG 90  WEBMJG
.FOOD                                                 BY: I.M. HUNGRY
*=================================================================================
LI=6,2=12,3=13,4=14,5=15,6=16
@BRK LDV,W <SOE>A1=SOE$,<COUNTER>A1=0 .
@001 .

      PLEASE MAKE YOUR LUNCH SELECTION:

   <SOE> PIZZA[        <SOE> BURGER[       <SOE> TACO[       <SOE> SALAD[
@BRK OUT,-0,2,4,,1 .
@002:IF CURH$ = 12 LDV <MESSAGE>$19='WITH PEPPERONI ONLY' GTO 006 ; .
@003:IF CURH$ = 28 LDV <MESSAGE>$21='MAKE IT A CHEESBURGER' GTO 006 ; .
@004:IF CURH$ = 41 LDV <MESSAGE>$20='EAT A MINT WHEN DONE' GTO 006 ; .
@005:IF CURH$ = 55 LDV <MESSAGE>$17='WHAT A HEALTH NUT' ; .
@006:INC <COUNTER> .
                    <MESSAGE>
@BRK OUT,-0,2,2,1,,,Y .
@WAT 5000 .
@IF <COUNTER> < 2 GTO 001 ; .

                    YOU ARE ONLY ALLOWED 2 SELECTIONS.......
                     ..... END REPORT .....
```

Define Statement

- Design / debug tool
- Assigns values to constant names used in run statements
  - Define constants at the beginning of the RCR
  - Define up to 400 constants
  - One define statement per line / constant
- Change the values of constants by updating the define statements
- Values can be
  - Labels
  - Run statement calls
  - Reports, drawers, and cabinets
  - Fields, parameters, and options
  - Output area data
  - Variable names
    - To initialize a variable (using a defined constant), enclose the type and size in single quotes
- Execute BLT run before placing run into production environment
  - Constants are converted to values defined
  - DEFINE statements and any comments are deleted
  - Replace or copy BLT result

Defining Constants

:DEFINE constant value

:DEFINE	Use either :DEFINE or :D.
constant	Constant name. Maximum = 18 characters (must begin with alpha). Alpha, numeric and underscores allowed.
value	Variable name, reserved word, literal data, or run statement call. Maximum 18 characters

DEFINE Example

- RCR to process

```
LINE▶ test1 FMT▶   RL▶ -     SHFT▶     HLD CHRS▶    HLD LN▶   UNDO▶   40E0     ▶
.DATE 06 SEP 90  16:01:19  RID     40E   30 AUG 90  CRAIG
.TEST1                                              BY: E.X. AMPLE       E000010
*=============================================================================
:DEFINE TASK SRH .        COMMENTS,
:DEFINE REPORT 0,B,2 .              COMMENTS,
:DEFINE FIELD1 45-4 .                         COMMENTS,
:DEFINE FIELD2 15-8 .                                     AND
:DEFINE PARAMETER1 AMCO .                                     MORE
:DEFINE PARAMETER2 GREENBOX .                                      COMMENTS
:DEFINE RESULT V1 .                                                         !!!
@LDV RESULT'A2'=-0 .
@TASK,REPORT '' FIELD1 [,PARAMETER1 .
@RNM -1 .
@TASK,REPORT '' FIELD2 [,PARAMETER2 .
@DSP,RESULT .

                    ..... END REPORT .....
```

- MAPPER interprets the code as .....

```
@LDV V1A2=-0 .
@SRH,0,B,2 '' 45-4 [,AMCO .
@RNM -1 .
@SRH,0,B,2 '' 15-8 [,GREENBOX .
@DSP,V1 .
```

- Result

```
Line▶ 1          Roll▶                                          RESULT
.
.    15 LINE(S) FOUND OUT OF 42 LINES
.
*              ********
*              GREENBOX
.
.DATE 10 JUL 90  12:26:20  RID      2B   26 FEB 90  ACAS
.@991231 Production Status Report                Corporate Production      B000002
*St.Status.By. Product .Serial.Produc.Order.Cust.Produc.Produc. Ship .Ship .Spc.
*Cd. Date .In.  Type   .Number. Cost .Numbr.Code. Plan .Actual. Date .Order.Cod.
*==.======.==.========.======.======.=====.====.======.======.======.=====.===.
 OR 831210 LS GREENBOX1                96751 FEDS
 IP 831227 LS GREENBOX1 605126         84385 FEDS 831225 831227
 OR 831211 LS GREENBOX4                96652 ARCO
 IP 831216 LS GREENBOX4 436295         85381 USSC 831215 831216
 SC 840103 LS GREENBOX4 675411         87947 USSC 840103
 SC 840109 LS GREENBOX4 675484         97942 USSC 840109
 IP 831230 LS GREENBOX4 974085         84581 INTR 831228 831230
 OR 831210 LS GREENBOX5                99753 DICO
 SC 840110 LS GREENBOX6 674481         95946 FEDS 840130
 SH 831206 LS GREENBOX7 669624         54682 AMCO 831201 831205 831206 S8553
```

DEFINE Example

- Change value of Parameter 2 from GREENBOX to BLACKBOX

```
Line▶ test1        Roll▶ -                                          40E0
.DATE 06 SEP 90  16:06:47  RID     40E   30 AUG 90  CRAIG
.TEST1                                              BY: E.X. AMPLE        E000010
*=============================================================================
:DEFINE TASK SRH .          COMMENTS,
:DEFINE REPORT 0,B,2 .                  COMMENTS,
:DEFINE FIELD1 45-4 .                           COMMENTS,
:DEFINE FIELD2 15-8 .                                     AND
:DEFINE PARAMETER1 AMCO .                                     MORE
:DEFINE PARAMETER2 BLACKBOX .                                     COMMENTS
:DEFINE RESULT V1 .                                                      !!!
@LDV RESULT'A2'=-0 .
@TASK,REPORT '' FIELD1 [,PARAMETER1 .
@RNM -1 .
@TASK,REPORT '' FIELD2 [,PARAMETER2 .
@DSP,RESULT .

                        ..... END REPORT .....
```

- MAPPER interprets code as .....

```
@LDV V1A2=-0 .
@SRH,0,B,2 '' 45-4 [,AMCO .
@RNM -1 .
@SRH,0,B,2 '' 15-8 [,BLACKBOX .
@DSP,V1 .
```

- Result

```
Line▶ 1          Roll▶                                          RESULT
.
.    27 LINE(S) FOUND OUT OF 42 LINES
.
*              ********
*              BLACKBOX
.
.DATE 10 JUL 90  12:26:20  RID      2B   26 FEB 90  ACAS
.@991231 Production Status Report                Corporate Production      B000002
*St.Status.By. Product .Serial.Produc.Order.Cust.Produc.Produc. Ship .Ship .Spc.
*Cd. Date .In.  Type    .Number. Cost .Numbr.Code. Plan .Actual. Date .Order.Cod.
*--.------.--.---------.------.------.-----.----.------.------.------.-----.---.
 IP 831224 LS BLACKBOX1 436767        84389 AMCO 831223 831224
 IP 831225 LS BLACKBOX1 436768        84390 AMCO 831223 831225
 IP 831219 LS BLACKBOX2 637071        84353 INTR 831218 831219
 OR 840110 LS BLACKBOX4               94754 ARCO
 SC 840110 LS BLACKBOX5 675281        97441 FEDS 840131
 IP 831222 LS BLACKBOX5 737582        84040 AMCO 831222 831222
 SH 831203 LS BLACKBOX0 746327        54237 FEDS 831201 831202 831203 $8738
 SH 831202 LS BLACKBOX6 368061        54438 FEDS 831201 831201 831202 $6937
 SH 831209 LS BLACKBOX6 777324        54232 DICO 831207 831208 831209 $8538
```

DEFINE Example

- Enter BLT on line zero before placing the run into production

```
Line▶ blt        Roll▶ -                                                40E0
.DATE 06 SEP 90  16:06:47  RID     40E   30 AUG 90  CRAIG
.TEST1                                               BY: E.X. AMPLE        E000010
*=================================================================================
:DEFINE TASK SRH .          COMMENTS,
:DEFINE REPORT 0,B,2 .                 COMMENTS,
:DEFINE FIELD1 45-4 .                          COMMENTS,
:DEFINE FIELD2 15-8 .                                   AND
:DEFINE PARAMETER1 AMCO .                                    MORE
:DEFINE PARAMETER2 BLACKBOX .                                     COMMENTS
:DEFINE RESULT V1 .                                                      !!!
@LDV RESULT'A2'=-0 .
@TASK,REPORT '' FIELD1 [,PARAMETER1 .
@RNM -1 .
@TASK,REPORT '' FIELD2 [,PARAMETER2 .
@DSP,RESULT .

                    ..... END REPORT .....
```

- BLT result

```
Line▶ 1          Roll▶ -                                                RESULT
.DATE 06 SEP 90  16:06:47  RID     40E   30 AUG 90  CRAIG
.TEST1                                               BY: E.X. AMPLE        E000010
*=================================================================================
@LDV V1A2=-0 .
@SRH,0,B,2 '' 45-4 [,AMCO .
@RNM -1 .
@SRH,0,B,2 '' 15-8 [,BLACKBOX .
@DSP,V1 .

                    ..... END REPORT .....
```

- Replace the BLT result

```
Line▶ rep 40e    Roll▶ -                                                RESULT
.DATE 10 SEP 90  14:03:09  RID     40E   30 AUG 90  CRAIG
.TEST1                                               BY: E.X. AMPLE        E000010
*=================================================================================
@LDV V1A2=-0 .
@SRH,0,B,2 '' 45-4  ,AMCO .
@RNM -1 .
@SRH,0,B,2 '' 15-8  ,BLACKBOX .
@DSP,V1 .

                    ..... END REPORT .....
```

Including constants defined in another run control report

:INCLUDE,c,d,r

:INCLUDE	Use either :I or :INCLUDE
c,d,r	Report containing constants to be included in his run.

- **Add all defined constants from another report to the calling run**
- **Place all include statements at the beginning of the calling run**
- **Specify only one report per INCLUDE statement**

INCLUDE Example

- Report to include

```
Line▶ 1          Roll▶ -                                          41E0
.DATE 06 SEP 90  15:49:16  RID     41E   06 SEP 90  CRAIG
.TEST2                                               BY: E.X. AMPLE        E000010
*=================================================================================
:DEFINE TASK2 SOR .
:DEFINE PARAMETER3 1 .

                    ..... END REPORT .....
```

- RCR to process (calling run) containing :INCLUDE statement
- Line 17 utilizes constants defined in another report

```
Line▶ test1      Roll▶ -                                          40E0
.DATE 06 SEP 90  16:12:33  RID     40E   30 AUG 90  CRAIG
.TEST1                                               BY: E.X. AMPLE        E000010
*=================================================================================
:INCLUDE 0,E,4 .
:DEFINE TASK SRH .          COMMENTS,
:DEFINE REPORT 0,B,2 .                  COMMENTS,
:DEFINE FIELD1 45-4 .                             COMMENTS,
:DEFINE FIELD2 15-8 .                                      AND
:DEFINE PARAMETER1 AMCO .                                     MORE
:DEFINE PARAMETER2 BLACKBOX .                                      COMMENTS
:DEFINE RESULT V1 .                                                          !!!
@LDV RESULT'A2'=-0 .
@TASK,REPORT '' FIELD1 [,PARAMETER1 .
@RNM -1 .
@TASK,REPORT '' FIELD2 [,PARAMETER2 .
@DSP,RESULT .
@TASK2,REPORT '' FIELD1 [,PARAMETER3 .
@DSP,RESULT .

                    ..... END REPORT .....
```

INCLUDE Example

- Enter BLT on line zero before placing into production

```
Line▶ BLT          Roll▶ -                                          40E0
.DATE 06 SEP 90  16:12:33  RID     40E   30 AUG 90  CRAIG
.TEST1                                               BY: E.X. AMPLE        E000010
*=============================================================================
:INCLUDE,0,E,41 .
:DEFINE TASK SRH .          COMMENTS,
:DEFINE REPORT 0,B,2 .                 COMMENTS,
:DEFINE FIELD1 45-4 .                          COMMENTS,
:DEFINE FIELD2 15-8 .                                   AND
:DEFINE PARAMETER1 AMCO .                                   MORE
:DEFINE PARAMETER2 BLACKBOX .                                   COMMENTS
:DEFINE RESULT V1 .                                                     !!!
@LDV RESULT'A2'=-0 .
@TASK,REPORT '' FIELD1 [,PARAMETER1 .
@RNM -1 .
@TASK,REPORT '' FIELD2 [,PARAMETER2 .
@DSP,RESULT .
@TASK2,REPORT '' FIELD1 [,PARAMETER3 .
@DSP,RESULT .

                        ..... END REPORT .....
```

- BLT result
- Values defined in 41E have been inserted

```
Line▶ 1            Roll▶ -                                      RESULT
.DATE 06 SEP 90  16:12:33  RID     40E   30 AUG 90  CRAIG
.TEST1                                               BY: E.X. AMPLE        E000010
*=============================================================================
@LDV V1A2=-0 .
@SRH,0,B,2 '' 45-4 [,AMCO .
@RNM -1 .
@SRH,0,B,2 '' 15-8 [,BLACKBOX .
@DSP,V1 .
@SOR,0,B,2 '' 45-4 [,[ .
@DSP,V1 .

                        ..... END REPORT .....
```

Clear Label Table Run Reviewed

- CLT entered on line 0 of displayed RCR
- Removes label table from run control report
- Produces result
- Used to remove table when debugging

BLT Run Statement

@BLT,c,d,r,*lab* .

c,d,r	Identifies report in which table is created
lab	Label to go to if error exists

- **Allows you to create label tables in other RCRs**
- **Handy when creating label tables in many RCRs**
- **Goes to specified label if @BLT errors**
- **Produces result**

CLT Run Statement

@CLT,c,d,r,*lab* .

c,d,r	Identifies report in which to clear label table
lab	Label to go to if there is an error

- **Allows you to remove label tables in other RCRs**
- **Goes to specified label if error occurs**
- **Produces result**

GO TO Statement Reviewed

- **Used to transfer control within an RCR**
- **Formats**
  - **@GTO END**
  - **@GTO label**
  - **@GTO lin+x**
  - **@GTO lin-x**
  - **@GTO variable**

JOHN ROSS
has a "RELABEL" ru

- RCR to execute

```
Line♦|TEST        Roll♦|-                                        100E0
.DATE    01 JUN 90  08:36:45  RID  100E   30 MAY 90  WILLIAMS
.TEST                                        Demonstration Runs          E0010
*=============================================================================
@GTO 001 .
@002 .
              CCCCCCCCCCCC
@GTO LIN+2 .
              EEEEEEEEEEEE
              DDDDDDDDDDDD
@GTO END .
              BBBBBBBBBBBB
@CHG V1I1 2 .
@GTO V1 .
@001 .
              AAAAAAAAAAAA
@GTO LIN-5 .


                        ..... END REPORT .....

 1       2Paint  3SOE    4Return 5       6Tasks  7View   8Help   9       10Edit
```

GO TO Statement

- Result after run execution

```
Line↕ 1          Roll↕ -                                          RESULT
.DATE   01 JUN 90  08:37:09     REPORT GENERATION   WILLIAMS
*=============================================================================
            AAAAAAAAAAAA
            BBBBBBBBBBBB
            CCCCCCCCCCCC
            DDDDDDDDDDDD
                        ..... END REPORT .....

 1      2Paint  3SOE    4Return 5      6Tasks  7View   8Help   9      10Edit
```

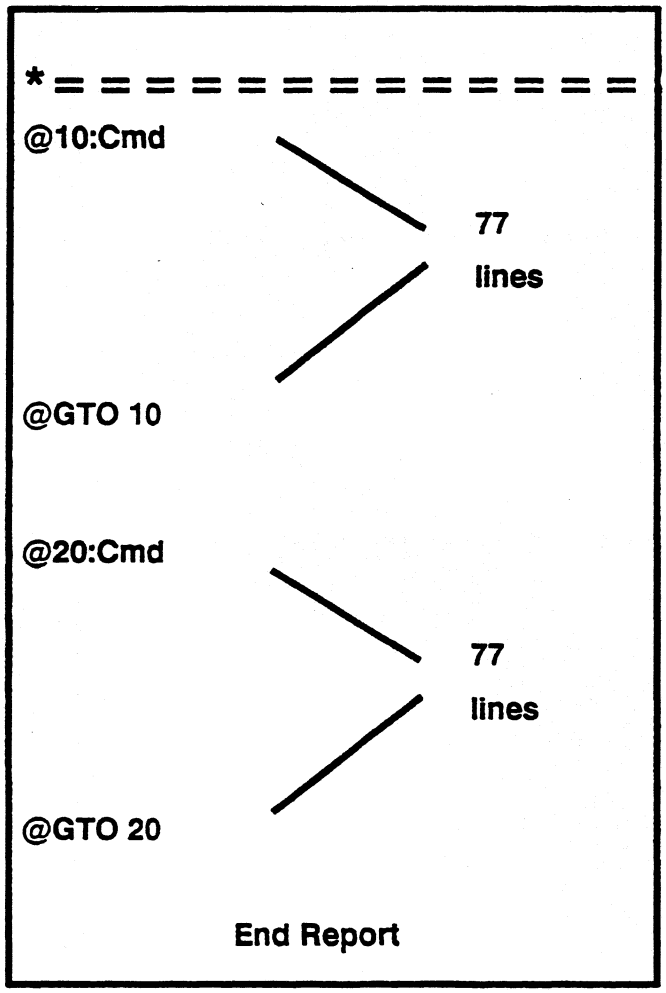
EXAMPLE EXPLANATION

- First run statement in RCR above tells MAPPER system to branch to label 1
- Next line - Output line of A's
- @GTO statement branches to current line minus 5 lines
  - Output line of B's
- @CHG statement initializes value of V1 to 2
- @GTO statement branches to the label number contained in variable V1, which currently has a value of 2
- Following label 2, is an output line of C's
- @GTO statement branches to the current line plus two lines
  - Output line of D's
- @GTO END statement
  - Displays the contents of the run output area.

Normal termination - Output area displayed looking like example shown

GO TO Statement and Efficiency

- **Confine segments of looping code within an execution window**
  - 'Paging' in and out of execution window causes extra I/Os



GO TO Statement and Efficiency

- **Use computational GTO when possible**
  - Number of commands processed for each @IF condition is reduced
  - LLPs and DLPs are also reduced

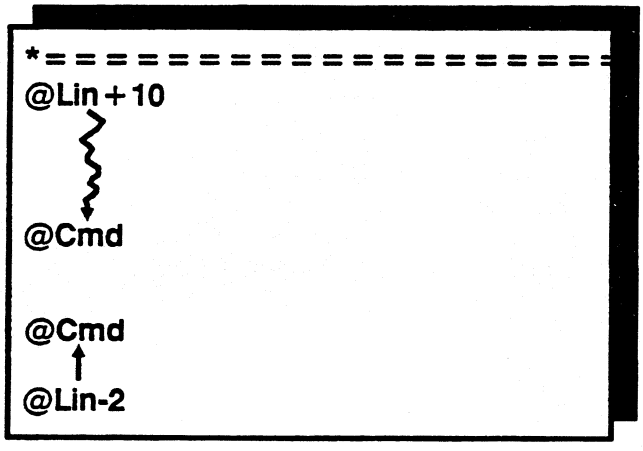
```
*===============
@If  v1  = 1,(10),2,(15),3,(20) .
@If  v1  = 4,5,6,(25),7,8,9,(30) .
```

versus

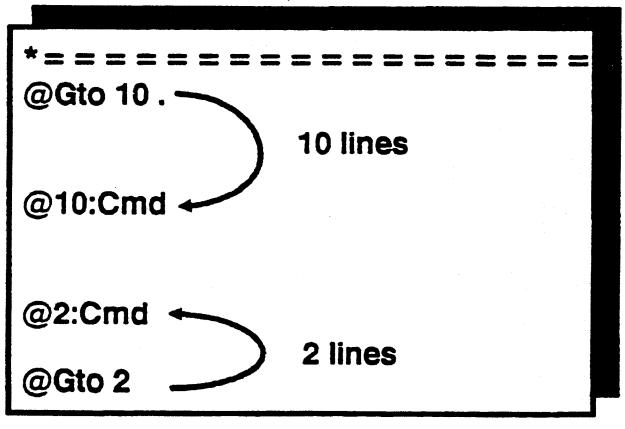
```
*===============
@If  v1  = 1  Gto 10 .
@If  v1  = 2  Gto 15 .
@If  v1  = 3  Gto 20 .

@If  v1  = 4,5,6  Gto 25 .
@if  v1  = 7,8,9  Gto 30 .
```

GO TO Statement and Maintenance

- **Use @GTO label versus @GTO lin**
  - @GTO lin is calculated by line number
  - If size of RCR is changed, run logic may become invalid



versus



Subroutines

Scenario

The Run Designer has the need to transfer control of a run to a subroutine.

Historical Solutions

Use series of GTO functions for internal subroutines
Use @GTO RPX function for external subroutines

- Permits a transfer of control to a subroutine in another report
- Drawbacks
  - Subroutine report must reside in same cabinet and drawer as the RCR
  - Always goes to line 3 of designated report
  - Connection from parent run is dropped - no way to return

Suggested Solution

Use @RSR, @ESR, and @CSR -or-
Use CALL and RETURN

@RSR

Nest 10 INT
or 1 EXT

@CALL

Nest 10 INT
or
10 EXT

@LNK

Nest 1 EXT

@ESR

All results
All variables } available

@RETURN

-ϕ passed only
passed variables

@GTO END

-ϕ passed only.
pass variables one way.

RSR, CSR, ESR Reviewed

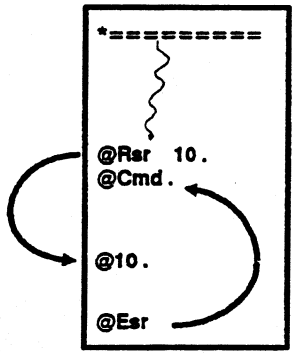
- @RSR,*c,d,r* lab .
  - Transfers control to a subroutine
  - Use for internal (same as RCR) or external subroutines
  - No other statements allowed on the same line
  - 10 levels of nesting allowed
  - Similar to @GTO RPX except:
    - Automatic return to calling RCR via @ESR
    - May access another RCR at any label via @RSR (instead of only line 3)
    - May access RCR in different cabinet and drawer
    - Variables and results created in subroutine are available to the calling RCR upon return

- @ESR,*q*
  - Returns to a line in the calling RCR
  - *q* quantity may be a positive or negative bias
  - No other statements on the same line

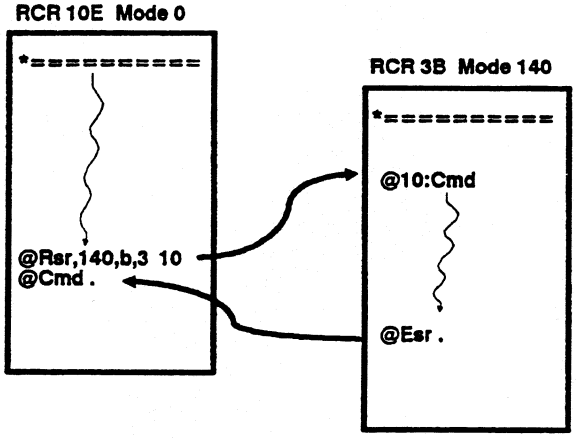
- @CSR
  - Breaks connection with calling RCR
  - Must be used before calling another external subroutine from within an external subroutine
  - May not use @ESR to return to calling RCR

RSR, CSR, ESR Reviewed

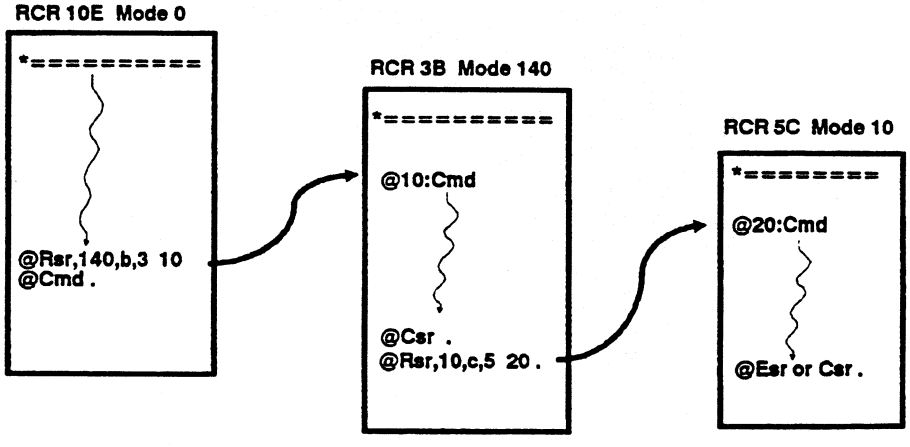
Internal



External with Esr



External with Csr



CALL Subroutine Parameters

@CALL,*c,d,r* lab (p1,p2,...)

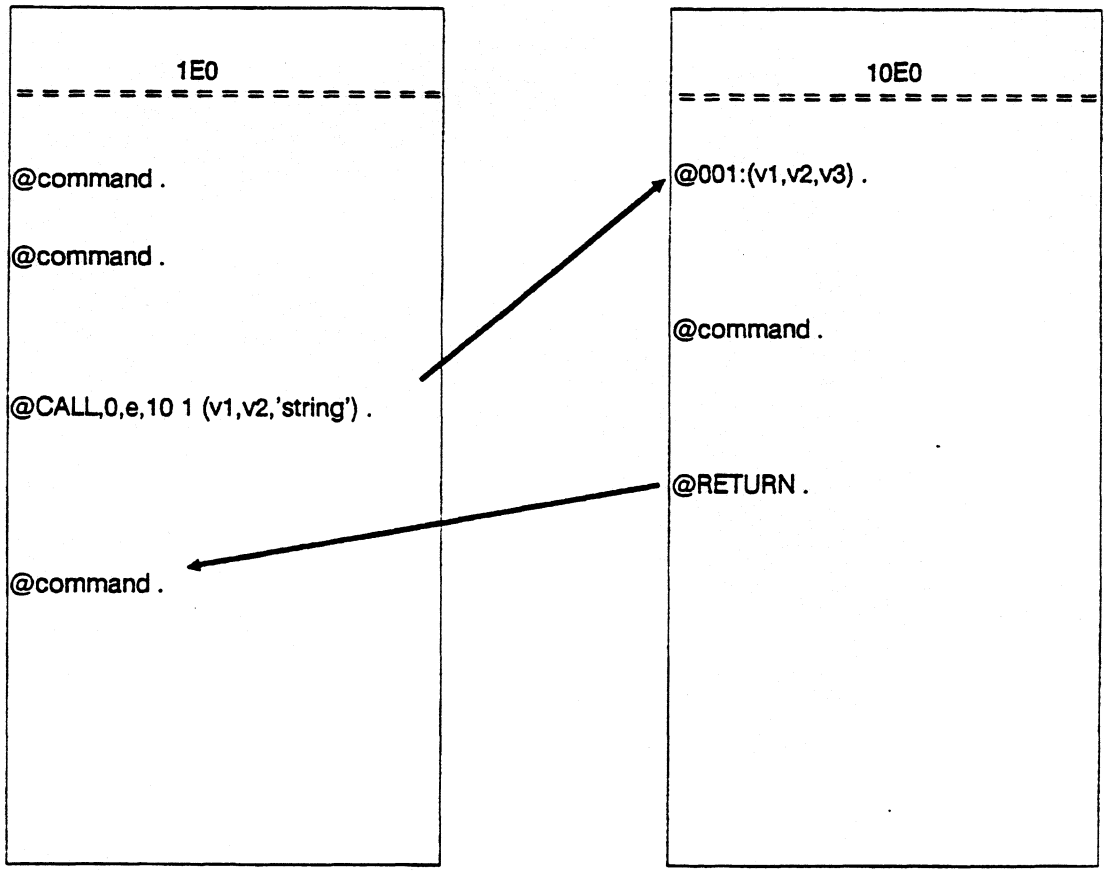
c,d,r	Report containing external subroutine
lab	Label where subroutine begins
	At subroutine label, you must include parentheses and a corresponding variable for each parameter sent
(p1,p2,...)	Parameters to pass to subroutine (40 maximum). Parentheses are required

- Parameters can be
  - V - variable (system will return any changed variable values to calling run)
  - *V - variable (calling run will retain the original values sent to subroutine regardless of activity of subroutine)
  - 'String' - literal character string
    - Character maximum = 256
    - Enclose each string in single quotes
    - At receiving label, enter a corresponding variable name or number for each string to pass
    - System initializes strings as 'string-type' variables to the size of string itself

Receiving the CALL Statement

@Lab:(variable1,variable2,variableN,...)

- List variable names following receiving label
- Parentheses are required around variable names



CALL Subroutine Statement

- Transfers control to a subroutine
- No other statement can follow @CALL
- Optionally specify parameters to pass
  - Maximum of 40
  - Passed values have a 1 to 1 relationship with variable names at receiving label
    - MAPPER equates passed values by position in statement - not by name
    - First variable in parenthesis = first variable of the CALL statement
  - Parameters can be
    - Variables
    - Literal character strings
- Variables manipulated in subroutine can be returned with
  - Original values
  - Changed values
- Variables initialized in subroutine can not be passed back to calling run
- Returns only -0 result created in subroutine
- Nest up to 10 CALL subroutines (internal or external)

CALL Subroutine Example

- RCR to process

```
Line▶ PROCESS    Roll▶ -                                          30E0
.DATE 21 AUG 90  16:00:22  RID     30E   21 AUG 90  WEBMJG
.PROCESS                                             BY: TRS
*=================================================================================
@SRH,0,C D 'PRODUCT TYPE(1-8)' [,BLACKBOX (FOUND)I4,(SCANNED)I4 .
@CAL,-0  '' 'RETAIL','WHOLE SALE','DEMO RESULTS' [,A,B,C+ C=A-B (RSLT)I6 .
@LDU (OPTIONS)A4=C(F),(FIELD)A12='PRODUCT TYPE' .
@CALL,0,1,34 003 ((OPTIONS),(FIELD)) .

@DSP,-0 .    DISPLAY THE SORTED RESULT

                    ..... END REPORT .....
```

- Subroutine is executed
- @RETURN will branch the run back to the calling RCR

```
LINE▶ 1     FMT▶   RL▶ -     SHFT▶     HLD CHRS▶    HLD LN▶        ▶   34E0     ▶
.DATE 21 AUG 90  15:45:58  RID     34E   21 AUG 90  WEBMJG
.SUBROUTINE                                          BY:                    E000010
*=================================================================================
@001 .
@COMMAND.....
@COMMAND.....

@002 .
@COMMAND......
@COMMAND......

@003:((OPTIONS),(FIELD)) .
@SOR,-0 (OPTIONS) '(FIELD)' [,1 .
@RETURN .

                    ..... END REPORT .....
```

CALL Subroutine Example

- Control is returned to the calling run
- SORT result is displayed

```
Line▶ 1          Roll▶                                          RESULT
.DATE 10 JUL 90  12:26:50  RID      1C   27 JUN 88  ACAS
.@991231 Factors Base Report                    Corporate Factors Base   C000004
* Product . Sub .Produc. Whole . Retail . Sales .Space.  Demo  .          .
*  Type   . Key . Cost . Sale$ .  $$$$  .Commiss. Req .Quantity. Demo Results .
*========.=====.======.=======.========.=======.=====.========.===============.
 BLACKBOX1     A  13500   16875    23625 2362.50   100        1            6750
 BLACKBOX1     A  13500   16875    23000 2362.50   100        1            6125
 BLACKBOX1     A  13500   16875    40000 2362.50   100        1           23125
 BLACKBOX1     A  13500   16875    29000 2362.50   100        1           12125
 BLACKBOX1     A  13500   16875    34000 2362.50   100        1           17125
 BLACKBOX1     A  13500   16875    22000 2362.50   100        1            5125
 BLACKBOX1     A  13500   16875    23625 2362.50   100        1            6750
 BLACKBOX1     A  13500   16875    23625 2362.50   100        1            6750
 BLACKBOX1     A  13500   16875    23625 2362.50   100        1            6750
 BLACKBOX1     A  13500   16875    23625 2362.50   100        1            6750
 BLACKBOX2     A  13600   17000    30000 2380.00   110        2           13000
 BLACKBOX2     A  13600   17000    50000 2380.00   110        2           33000
 BLACKBOX2     A  13600   17000    60000 2380.00   110        2           43000
 BLACKBOX2     A  13600   17000    23800 2380.00   110        2            6800
 BLACKBOX2     A  13600   17000    23800 2380.00   110        2            6800
 BLACKBOX2     A  13600   17000    23800 2380.00   110        2            6800
 BLACKBOX2     A  13600   17000    23800 2380.00   110        2            6800
 1Resume 2Paint  3       4Return 5       6Tasks  7View   8Help   9      10Edit
```

Explanation

- Calling RCR (30E)
- Performs a Search on a report
- Followed by a Calculate on result
- Result now needs to be sorted
  - Sort subroutine exists in 34E
- @CALL statement is used to transfer control to sort subroutine
  - Two values are passed - option(s) to be used and field(s) to Sort on
- Subroutine accepts control of run and collects values sent
- Values are initialized as variables in subroutine at receiving label
- Sort on current result is performed
- @RETURN statement returns control to the parent RCR where sorted result is displayed

Subroutine Differences

- **Refer to chart below when making a decision on which statement to use**

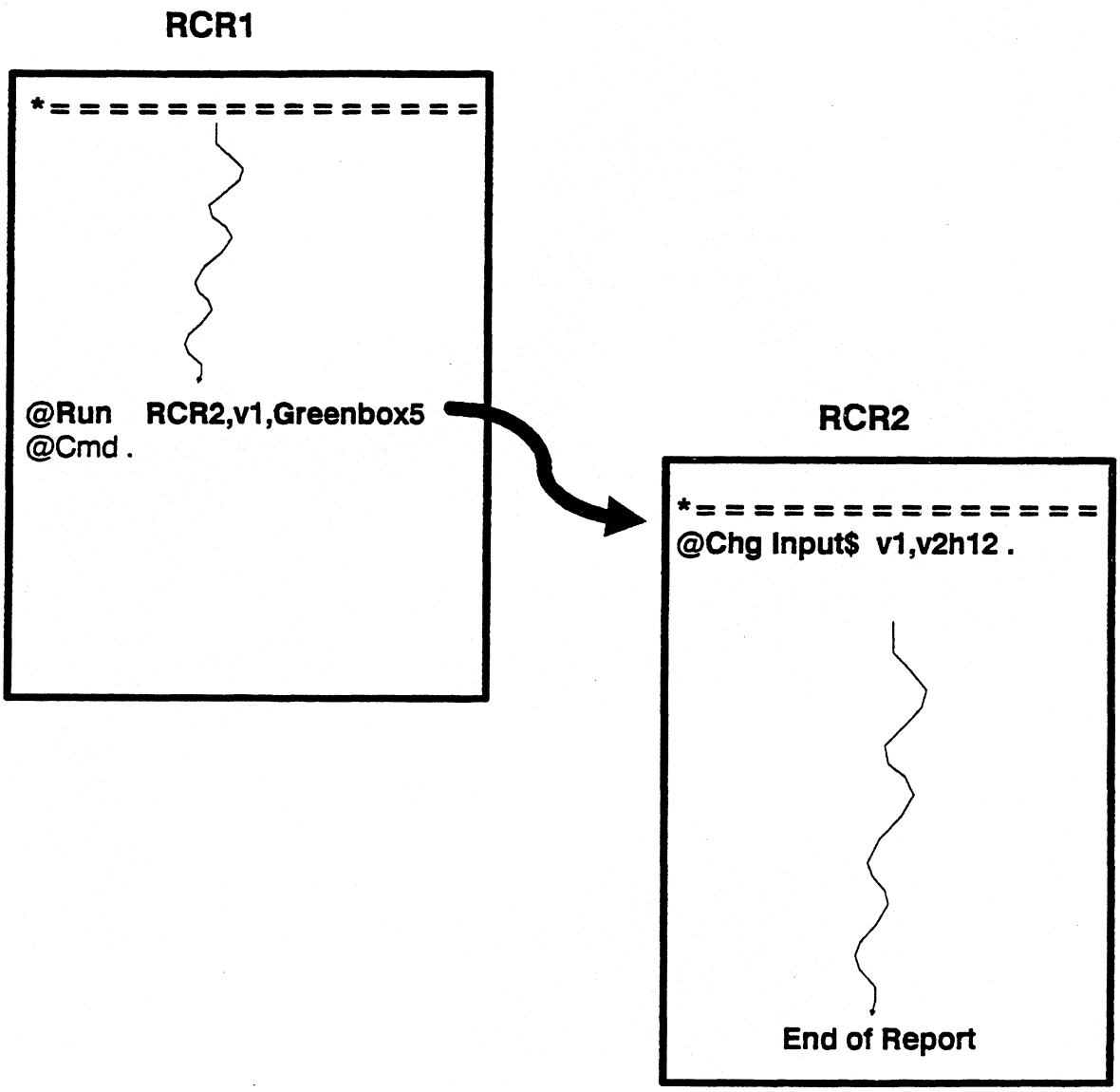
RSR/ESR	CALL/RETURN
All variables are automatically passed	Selected values can be passed
Literal strings can not be passed	Literal strings can be passed
Variables changed in the subroutine are returned to the calling run changed	Variables changed in the routine may be returned changed -or- returned with original values
Control can be returned to the calling run at a specific place in the run. A positive or negative bias can be specified in the @ESR statement	Control is returned to the calling run at the line following the @CALL statement
1 external subroutine can be nested	Nest up to 10 external subroutines

Starting a Run from a Run

@RUN run-name,data

- Starts a run from within a run
- Calling run is terminated when @RUN is executed
- Can send up to 40 data values
  - Data values captured via @CHG INPUT$ in started run
- No other statements can follow @RUN
- Overhead expenses are incurred
  - PreRun
  - New D Bank
  - New variable table
  - New label table
- Execution begins at line three of receiving RCR
  - Calling run is terminated
- Need not know the cabinet, drawer, and report to access
- Passes -0 result to the started run

@RUN Example

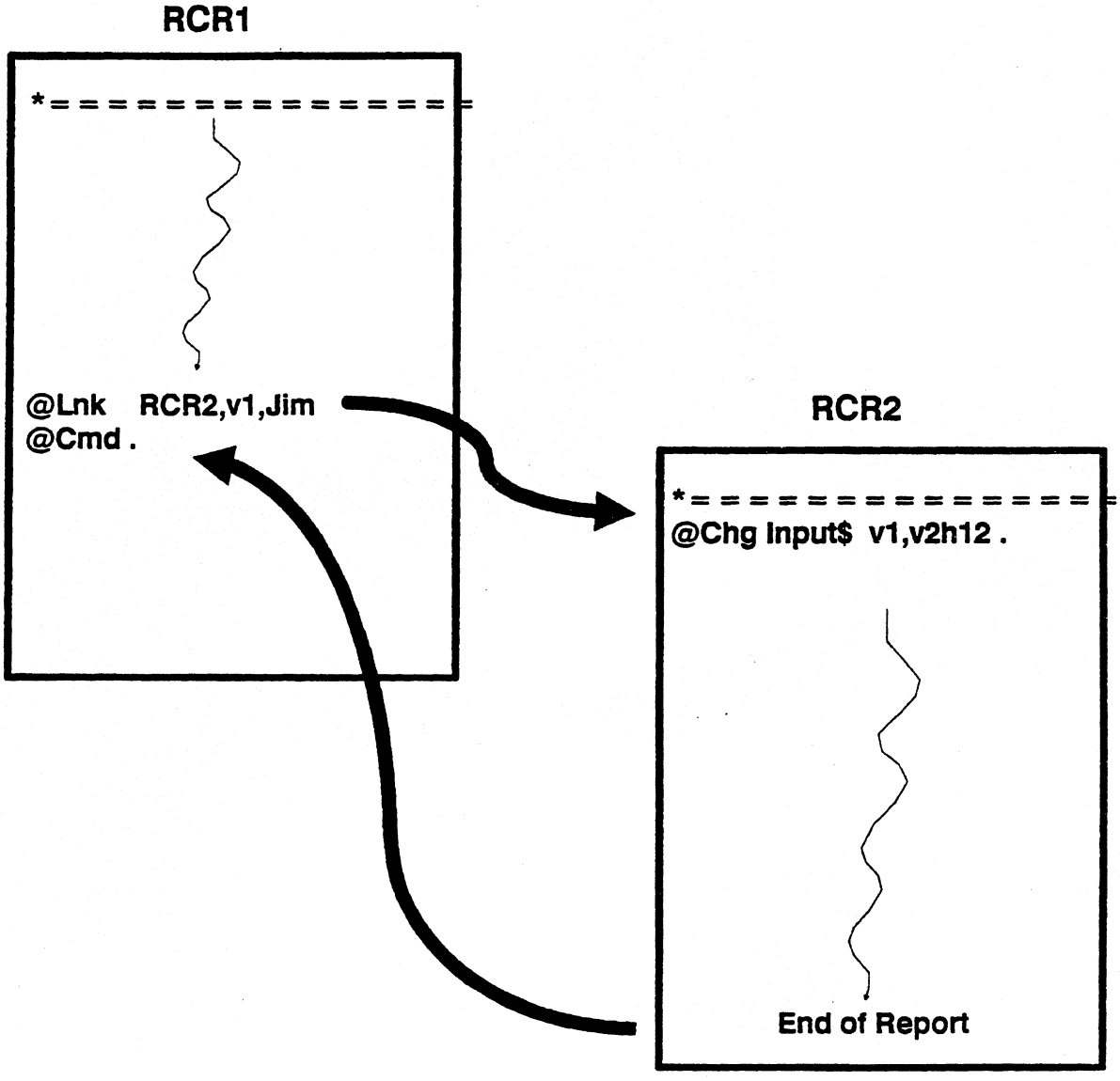

RCR1
*==========
@Run RCR2,v1,Greenbox5
@Cmd .
RCR2
*==========
@Chg Input$ v1,v2h12 .
End of Report

Starting a Run from a Run

@LNK run-name,data

- Same as @RUN except:
  - Started run returns to calling run when 'End of Report' is executed
  - Must use an @REL or @XIT to bypass execution of 'End of Report'
  - -0 result is returned to calling run
  - STAT words can be used to pass values back to calling run when @GTO END is executed
  - @GTO END,val1,val2,val3
    - Val2,val2,val3 are loaded automatically into reserved words STAT1$,STAT2$,STAT3

@LNK Example



Starting Background Runs from a Run

@BR,sn,lab run-name,data

sn	Station number to be notified when background run completes Default = no station notified
lab	Label to go to if the number of active background runs has already reached the maximum allowed
run-name	Name of the background run to start
data	Variables,literals,reserved words, or constants to pass to the background run. Up to 40 values captured via @CHG INPUT$ in the started background run

@BR Statement

- Starts and executes a run in 'background'
- Calling run continues to execute at your terminal
- Typically a 'non-interactive' run
- If station number is specified:
  - Receiving station beeps when background run is terminated
    - Press 'Msg Wait' to display message
  - Message verifies successful run completion along with other information on run -or-
  - Displays error information
- Can send up to 40 values with a 640 character maximum
- -0 result is passed to background run
- Registered one of two ways
  - Normal run eligible for background execution
  - Background run only
- Background run can not start another background run
- Certain functions not allowed: see online HELP for list

@BR Example

- RCR to process
- 'SORTIT' run will be executed via @BR statement

```
Line▶ PROCESS     Roll▶                                                35E0
.DATE 21 AUG 90  11:13:39  RID     35E   21 AUG 90  WEBMJG
.PROCESS                                            BY: JDT                E000010
*=================================================================================
@BR 11637,001 SORTIT .
@BRK LDV,W <SOE>A1=SOE$,<COUNTER>A1=0 .
@001 .

      PLEASE MAKE YOUR LUNCH SELECTION:

   <SOE> PIZZA[        <SOE> BURGER[       <SOE> TACO[       <SOE> SALAD[
@BRK OUT,-0,2,4,,1 .
@002:IF CURH$ = 12 LDV <MESSAGE>S19='WITH PEPPERONI ONLY' GTO 006 ; .
@003:IF CURH$ = 28 LDV <MESSAGE>S21='MAKE IT A CHEESBURGER' GTO 006 ; .
@004:IF CURH$ = 41 LDV <MESSAGE>S20='EAT A MINT WHEN DONE' GTO 006 ; .
@005:IF CURH$ = 55 LDV <MESSAGE>S17='WHAT A HEALTH NUT' ; .
@006:INC <COUNTER> .
                    <MESSAGE>
@BRK OUT,-0,2,2,1,,,Y .
@WAT 5000 .
@IF <COUNTER> < 2 GTO 001 ; .

                YOU ARE ONLY ALLOWED 2 SELECTIONS.......
 1       2Paint  3       4Return 5       6Tasks  7View   8Help   9       10Edit
```

- 'SORTIT' RCR

```
Line▶ 1          Roll▶ -                                              32E0
.DATE 21 AUG 90  11:12:01  RID     32E   15 AUG 90  WEBMJG
.SORTIT                                             BY:                    E000010
*=================================================================================
@001 .        SORTING ROUTINE
@SOR,0,B,39 '' 'PRODUCT TYPE'  ,1  REP,-0,0,B,39 .
@SOR,0,C,39 '' 'PRODUCT TYPE'  ,1  REP,-0,0,C,39 .
@SOR,0,D,39 '' 'PRODUCT TYPE'  ,1  REP,-0,0,D,39 .

                    ..... END REPORT .....
```

@BR Example

- Calling run continues to execute on calling screen

```
PLEASE MAKE YOUR LUNCH SELECTION:
▶ PIZZA        ▶ BURGER        ▶ TACO        ▶ SALAD
```

- 'SORTIT' run is executed on receiving terminal (number 11637)
- Upon background run completion, receiving terminal beeps
- Press 'MSG WAIT' key
- Confirmation message is displayed on receiving screen (11637)
- Enter 'OK' and transmit to acknowledge that message was received

```
ok                          Station to station message
.DATE 21 AUG 90  11:20:53  ** FROM STATION 99997    WEBMJG

BACKGROUND EXECUTION OF RUN: SORTIT STARTED FROM STATION 11636 AT 11:20:53
BY WEBMJG IN DEPARTMENT 100 SUCCESSFULLY COMPLETED AT 11:20:53

                    ..... END REPORT .....
```

XQT Statement

@XQT, *lab* or *vld* .

lab	Label or line number to execute
vld	Variables, literals, constants, reserved words, or any combination of these from which to formulate and execute a run statement

- **Construct and or execute run statements on the fly**
  - **Multiple commands can be executed**
  - **Execute up to 640 characters of data**
- **When a statement being executed errs:**
  - **Run goes to the label specified in statement that XQT is executing**
  - **If no label is specified in statement being executed, run errs at @XQT statement**
- **No other statements can follow @XQT**
- **After @XQT executes a function, the run continues at line following @XQT**
- **XQT statements can be nested**
  - **May be more difficult to maintain**

XQT Example

Scenario

Run designer is creating a run which displays a menu to the user

- Menu will offer three different function selections
  - Selection 1 will be to 'Search a report by Status Date
  - Selection 2 will 'Search the report by Ship Date'
  - Selection 3 will 'Sort the report by Product Type'
- Depending on selection, the run will go to a label in the RCR which contains the appropriate run statement to 'execute
- Statement is processed
- Run loops back up to line following the @XQT statement
- Run processing continues

- RCR to process

```
LINE▶        FMT▶   RL▶        SHFT▶      HLD CHRS▶    HLD LN▶        ▶   37E0     ▶
.DATE 21 AUG 90  11:59:21  RID     37E   21 AUG 90  WEBMJG
.EXECUTE                                            BY:                    E000010
*==========================================================================
@GTO 001 .
@002:SRH,0,B,2 '' 'STATUS DATE' [,831203 .
@003:SRH,0,B,2 '' 'SHIP DATE' [,831209 .
@004:SOR,0,B,2 '' 'PRODUCT TYPE' [,1 .

@001:BRK .
                    PLEASE MAKE YOUR SELECTION NOW

              SEARCH YOUR REPORT BY:
                                    STATUS DATE [
                                    SHIP DATE
                              OR

              SORT YOUR REPORT BY PRODUCT TYPE
@BRK OUT,-0,2,9,1,1 .
@IF CURV$ = 4 XQT,002 ;
@IF CURV$ = 5 XQT,003 ;
@IF CURV$ = 8 XQT,004 ;
@DSP,-0 .
```

XQT Example

- User selects 'Search by Ship Date'

```
                PLEASE MAKE YOUR SELECTION NOW
        SEARCH YOUR REPORT BY:
                                STATUS DATE [
                                SHIP DATE [
                        OR
        SORT YOUR REPORT BY PRODUCT TYPE [
```

- Appropriate Search function is executed

```
LINE▶       FMT▶   RL▶       SHFT▶     HLD CHRS▶    HLD LN▶        ▶   37E0     ▶
.DATE 21 AUG 90  11:59:21  RID     37E   21 AUG 90  WEBMJG
.EXECUTE                                            BY:                    E000010
*=================================================================================
@GTO 001 .
@002:SRH,0,B,2 '' 'STATUS DATE' [,831203 .
@003:SRH,0,B,2 '' 'SHIP DATE' [,831209 .
@004:SOR,0,B,2 '' 'PRODUCT TYPE' [,1 .

@001:BRK .
                PLEASE MAKE YOUR SELECTION NOW
        SEARCH YOUR REPORT BY:
                                STATUS DATE [
```

- Search result is displayed

```
Line▶ 1          Roll▶ -                                          RESULT
.
.     1 LINE(S) FOUND OUT OF 42 LINES
.
*                                                              ******
*                                                              831209
.
.DATE 10 JUL 90  12:26:20  RID      2B   26 FEB 90  ACAS
.@991231 Production Status Report                Corporate Production     B000002
*St.Status.By. Product .Serial.Produc.Order.Cust.Produc.Produc. Ship .Ship .Spc.
*Cd. Date .In.  Type   .Number. Cost .Numbr.Code. Plan .Actual. Date .Order.Cod.
*==.======.==.========.======.======.=====.====.======.======.======.=====.===.
 SH 831209 LS BLACKBOX6 777324        54232 DICO 831207 831208 831209 S8538
                     ..... END REPORT .....
```

Exercise

The following exercises are build upon one another.

1. Write a run to:
   a. Search report 2B0 for Customer Codes (cust-code) of AMCO. Sort the result by Product Type. Display the result.
   b. Execute the run. How many occurrences of AMCO were found?

   _____

2. Add a DEFINE statement before the @SRH statement.The constant value will = REPORT.The value will = 0,B,2.
3. Change the report location in the @SRH statement from 0,B,2 to REPORT.
   a. Execute the run. Was the same result produced as in 1b? _____
4. Change the DEFINE statement value to 0,D,1.
   a. Execute the run. How many occurrences of AMCO were found?

   _____

5. Add a report to the same cabinet / drawer that holds your RCR. This RCR will be a subroutine that will be accessed by your original RCR.
   a. In the new report, write a statement that searches the current -0 for all occurrences of BLACKBOX9 in the Product Type field.
   b. Use RSR / ESR to process the subroutine. Transfer control to the routine after the @SOR statement in the calling run has been executed. Return control to the calling run.
6. Perform the same task as in 5b, but use CALL / RETURN.

7

Working with Variables

Module 7
Working with Variables

Objectives

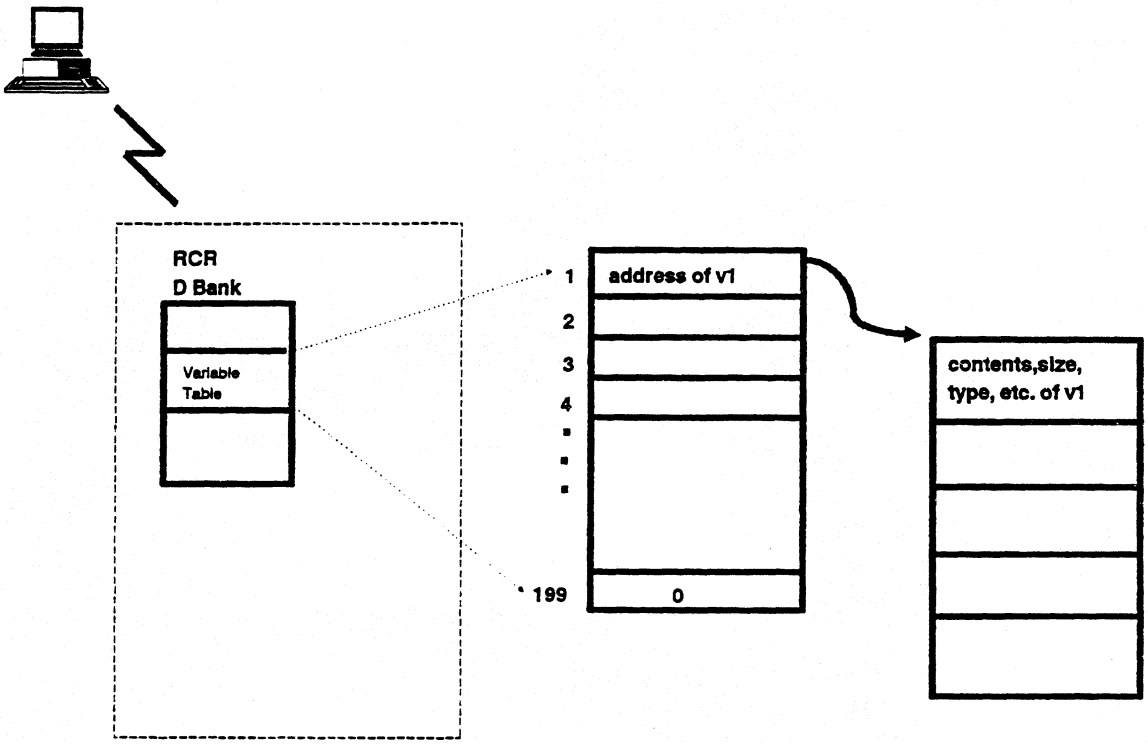
Upon completion of this module, you should be able to:

1. Discuss the set up of the variable table.
2. Implement Variable Stack commands.
3. Utilize alternative methods of initializing and documenting variables.
4. Evaluate efficiency techniques concerning variables.

Variable Table

- Every run has its own Variable Table
- Each Variable Table is
  - Stored in the RCR's D Bank
  - Fixed in size
  - Indexed by variable number
- VARNUM determines table size
  - VARNUM value range = 199 through 399
  - VARNUM default = 199
  - Table expands if VARNUM is increased
- The address of a variable points to another location where the actual contents, size, type, etc. of the variable resides
- Undefined variables contain a 0 in variable table entry

Variable Table



Variable Stacks

- Data structure that stores variables
- Consists of up to 10 levels
  - Each level is referred to by a relative number (-0 through -9)
  - The most current level is referred to as -0
- Use Stack commands to save, clear, and retrieve variables
- Use Variable stacks to
  - Maintain up to 11 times the maximum number of variables allowed per run without changing the value of VARNUM
  - Avoid variable conflicts when utilizing subroutines
  - Utilize more string variable space
- Variable Stacks are efficient
  - No I/Os are created by moving variables to and from each stack
  - A stack is actually a buffer in main storage
- The reserved word STACK$ contains the current level number

Variable Stack Commands

- **@PSH[,stvno,q,slv] .**
  - Saves the definition of a variable and its contents for later use
  - Creates a new level

- **@POP[,stvno,q,slv] .**
  - Restores a variable in the run control report with the contents of a variable in the stack
  - Removes a level

- **@PEK[,stvno,q,slv] .**
  - Restores a variable for use by the run control report without affecting the level or its contents

- **@POK[,stvno,q,slv] .**
  - Changes definition and/or its contents of a variable in a level

- **@RMV[,slv] .**
  - Removes a level and releases all its variables. Variables are not restored

- **@XCH[,stvno,q,slv] .**
  - Exchanges variables definition and contents between a specified level and the run control report

- **@CLV[,stvno,q,slv] .**
  - Releases a variable in the run control report

Variable Stack Example 1

- RCR to process

```
LINE▶ 1      FMT▶   RL▶ -     SHFT▶     HLD CHRS▶    HLD LN▶    ▶      fcs      ▶
.DATE 24 JUL 90  08:34:39  RID      1E   24 JUL 90  CRAIG
.TEST                    STACK  EXAMPLE
*==============================================================================
@BRK .
@LDV v1a4=Mary,v2A6=little,v3A5=lamb .
     v1 had a v2 v3
@BRK OUT,-0,2,1,1 .
@PSH,1,3 .
@CHG v1a8 McDonald CHG v2 larger CHG v3 cow .
      Old v1 had a v2 v3
@BRK OUT,-0,2,1,1 .
@PEK,2,2,-0 .
         v1 bought the v2 v3
@BRK OUT,-0,2,1,1 .
@CHG v1 dined CHG v2 chops CHG v3 wine LDV,P v1,v2,v3 .
        Later that night, he v1 on v2 and v3
@BRK OUT,-0,2,1,1 .
@POP .
          After hearing about her v3, v1 had v2 to say
@BRK OUT,-0,2,1,1 .

                        ..... END REPORT .....
```

- @LDV v1a4 = Mary,v2a6 = little,v3a5 = lamb
- Output line: v1 had a v2 v3
- @BRK OUT,....

```
Mary had a little lamb
```

- @PSH,1,3 . (PSH pushes v1-Mary,v2-little and v3-lamb out to level -0)
- @CHG v1a8 McDonald CHG v2 larger CHG v3 cow
- Output line: Old v1 had a v2 v3
- @BRK OUT,...

```
Old McDonald had a larger cow
```

Variable Stack Example 1

- RCR to process

```
LINE▶ 1      FMT▶   RL▶ -     SHFT▶     HLD CHRS▶    HLD LN▶    ▶      fcs      ▶
.DATE 24 JUL 90  08:34:39  RID      1E   24 JUL 90  CRAIG
.TEST                    STACK  EXAMPLE
*==========================================================================
@BRK .
@LDV v1a4=Mary,v2A6=little,v3A5=lamb .
     v1 had a v2 v3
@BRK OUT,-0,2,1,1 .
@PSH,1,3 .
@CHG v1a8 McDonald CHG v2 larger CHG v3 cow .
      Old v1 had a v2 v3
@BRK OUT,-0,2,1,1 .
@PEK,2,2,-0 .
         v1 bought the v2 v3
@BRK OUT,-0,2,1,1 .
@CHG v1 dined CHG v2 chops CHG v3 wine LDV,P v1,v2,v3 .
        Later that night, he v1 on v2 and v3
@BRK OUT,-0,2,1,1 .
@POP .
         After hearing about her v3, v1 had v2 to say
@BRK OUT,-0,2,1,1 .

                    ..... END REPORT .....
```

- @LDV v1a4 = Mary,v2a6 = little,v3a5 = lamb
- Output line: v1 had a v2 v3
- @BRK OUT,....

```
Mary had a little lamb
```

- @PSH,1,3 . (PSH pushes v1-Mary,v2-little and v3-lamb out to level -0)
- @CHG v1a8 McDonald CHG v2 larger CHG v3 cow
- Output line: Old v1 had a v2 v3
- @BRK OUT,...

```
Old McDonald had a larger cow
```

Variable Stack Example 2

- RCR to process

```
LINE▶ test  FMT▶   RL▶ -    SHFT▶    HLD CHRS▶    HLD LN▶    ▶      fcs      ▶
.DATE 24 JUL 90  13:05:01  RID      1E   24 JUL 90  CRAIG
.TEST                     STACK  EXAMPLE
*==============================================================================
@BRK .
@LDV v1a4=rain,v2A5=Spain,v3A5=plain .
     The v1 in v2 falls mainly on the v3
@BRK OUT,-0,2,1,1 .
@PSH .
@CHG v1 hail CHG v2 Wales CHG v3 mail .
     The v1 in v2 will slow down the v3
@BRK OUT,-0,2,1,1 .
@XCH,3,1 .
     The v1 in v2 also falls on the v3
@BRK OUT,-0,2,1,1 .
@POP .
     By the way, the v1 in v2 can also hold up the v3
@BRK OUT,-0,2,1,1 .

                    ..... END REPORT .....
```

- @LDV v1a4 = rain,v2a5 = Spain,v3a5 = plain
- Output line: The v1 in v2 falls mainly on the v3
- @BRK OUT,-0,...

```
The rain in Spain falls mainly on the plain
```

Variable Stack Example 2

- @PSH (psh 'pushes' v1-rain,v2-Spain, and v3-plain out to level -0)
- @CHG v1 hail CHG v2 Wales CHG v3 mail
- Output line: The v1 in v2 will slow down the v3
- @BRK OUT,-0,....

```
The hail in Wales will slow down the mail
```

- @XCH,3,1 (xch exchanges v3 from the RCR 'mail' and v3 from level -0 'plain')
- Output line: The v1 in v2 also falls on the v3
- @BRK OUT,-0,....

```
The hail in Wales also falls on the plain
```

- @POP (pop takes v1 'rain', v2 'Spain', and v3 'mail' off of the level -0 and places them back into the RCR)
- Output line: By the way, the v1 in v2 can also hold up the v3
- @BRK OUT,-0,...

```
By the way, the rain in Spain can also hold up the mail
```

Variable Stack Example 3

- RCR to process

```
LINE▶ test  FMT▶   RL▶ -     SHFT▶     HLD CHRS▶    HLD LN▶    ▶      fcs      ▶
.DATE 23 JUL 90  12:33:03  RID      1E   23 JUL 90  CRAIG
.TEST                     STACK EXAMPLE
*==============================================================================
@BRK .
@LDV V1A4=rain,V2A5=Spain .
@PSH .
@CHG V1 hail CHG V2 Wales .
@POK,1,1 .
     It can v1 in v2
@POP .
                 just as easy as it can v1 in v2
@BRK OUT,-0,2,2,1 .

                        ..... END REPORT .....
```

- Output area

```
It can hail in Wales
          just as easy as it can hail in Spain
```

Explanation:

- PSH statement pushes v1 'rain' and v2 'Spain' to stack level -0
- V1 and v2 are then re-initialized to 'hail' and 'Wales'
- POK statement puts v1 'hail' of RCR to -0 stack level - replacing 'rain'
- Value 'hail', now exists in RCR and on -0 level
- POP statement brings v1 and v2 from the level -0 and into the RCR
- V1 from the RCR has the same value as v1 from the stack

Using Variable Stacks to Avoid Conflicts with Subroutines

Scenario

The Run Designer, in parent run, has initialized four variables (V1 - V4). A subroutine which must now be accessed which also initializes V1 and V2. How does the Run Designer avoid the conflict of V1 and V2 in the subroutine?

Historical Solutions

Reinitialization of variables in parent run before they are passed.

Obtain a copy of subroutine and change the variable definitions.

Suggested Solution

Push variables V1 through V2 out to a stack level. This way, conflicts with variables in subroutine will be avoided. V3 and V4 will be common variables between parent run and subroutine.

Using Variable Stacks to Avoid Conflicts with Subroutines

RCR

```
@LDV V1I3=412, V2H2=ST, \
   V3H6=101060, V4I1=8 .

@PSH,1,2 .
@RSR .
```

Stack

```
V1=412
V2=ST
```

@RSR

Subroutine

```
@10: FND,0,B,V4,6,99 " 15-5 [,GREEN \
     V1I6, V2I6 .
.
.
@ESR .
```

Using Variable Stacks to Increase Available Number of Variables

Scenario

VARNUM is set to a maximum of 199 variables. The Run Designer has the need to utilize more than 199 **different** variables.

Historical Solutions:

Increase VARNUM - which will affect the entire system.

Begin to reinitialize certain variables that have already been used.

Load muti-values into string type variables and later break them up.

Suggested Solution

Utilize variable stack commands to save (push) the first 199 values out to level -0. Then variable positions can be used over again (reinitialized). The Designer will now have the capability to utilize 199 x 11 the number of variables.

Using Variable Stacks to Increase Available Number of Variables

VARNUM = 199

RCR

- Initialization
and use of
variables v1
through v199

@10:PSH,1,199

.
.
.

-Reinitialization
and use of
variables v1 through v199

@20:PSH,1,199 .

STACK

V1
V2
V3
.
.
.
.
.
.
V199

-0

V1
V2
V3
. Contents of V1
. through V199 at
. label 20
V199

-0

V1
V2
V3
. Contents of V1
. through V199 at
. label 10
V199

-1

Using Variable Stacks to Increase Available String Variable Space

Scenario

The Run Designer has the need to utilize several 'string' type variables. After 31 'string' type variables of 132 characters are initialized, the maximum string character limit of 4096 is reached. The Designer must now figure out a way to work around the 4096 limit.

Historical Solutions

Once again - variables could be reinitialized

Variable values could also be written out to another report and later retrieved.

Suggested Solution

Utilize variable stack commands. 'String' type variables of 132 characters can be pushed out to stack levels 31 at a time. The Designer will have 11 times the number of variables. The string character limit can be avoided - providing now 45,056 characters of space.

Using Variable Stacks to Increase Available String Variable Space

RCR

```
@LDV <REC1>S132=", \
<REC2>S132=", <REC3>S132=", \
<REC4>S132=", <REC5>S132=", \
<REC6>S132=", ... through <REC31>S132 .
        - Fill V1 through V31 with data
@10: PSH,1,31 .
        - Reuse V1 through V31
@20: PSH,1,16 .
```

Stack

```
<REC1>
<REC2>
<REC3>
. Contents of
. variables 1 through
. 31 at label 20
<REC31>
```

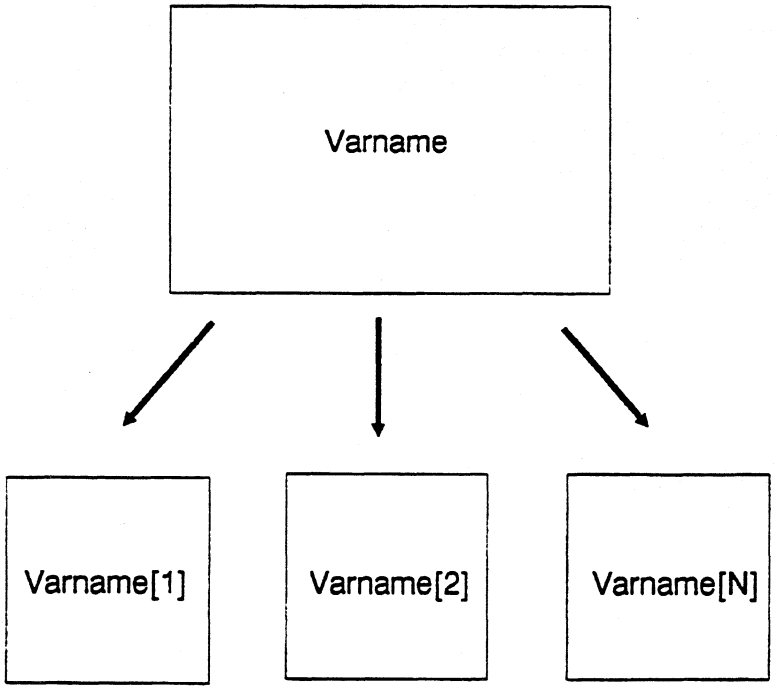
-0

```
<REC1>
<REC2>
<REC3>
. Contents of
. variables 1 through
. 31 at label 10
<REC31>
```

-1

Variable Array

- Group of values of the same type and size initialized under one variable name
- Each value in an array is called a member
- Each member is identified by a number placed in brackets following the variable name
- Use variable arrays to
  - Load several 'like' values with one command
  - Pass more than 40 values via @CALL



@LDA Statement

- Initialize a variable array
- Assign values to members
- Variable members can be
  - Variables
  - Literal data
  - Constants
  - Reserved words
- Each variable array name = one value in VARNUM regardless of number of members
- RCR must be in full character set (FCS or FCSU)
- Array members consume 'string' variable space regardless of type

@LDA Statement

@LDA,*O* nametypesize[N] = Value1,*Value2,Value3,....*

O	LDA options
nametypesize	Name of the variable array to initialize - immediately followed by the variable type and number of characters allowed in each member of the array
[N]	Number of members in the array. Brackets are required
Value	Data to load into each member of the array

Options

C	Centers data within the variable
L	Left-justifies the data within each variable
P	Packs data into each variable so that the variable contains only significant characters
R	Right-justifies the data within each variable
U	Converts all lowercase alphabetic characters to uppercase characters
W	Loads variables with the value of reserved words
Z	Zero fills each variable after the data is loaded

Using Variable Arrays Example 1

- RCR to process

```
LINE▶ test  FMT▶   RL▶ -     SHFT▶     HLD CHRS▶    HLD LN▶        ▶   6F0      ▶
.DATE 03 AUG 90  11:35:57  RID      6F   27 JUL 90  WEBMJG
.TEST                                               BY:                        F000012
*=================================================================================
@LDA <names>a5[4]=ROGER,NANCY,SUSIE,PETER .

      <NAMES>[1], <NAMES>[2], <NAMES>[3], AND <NAMES>[4] ALL WORK TOGETHER.

@BRK DSP,-0 .


                    ..... END REPORT .....
```

- Output area is displayed

```
line▶ 1     fmt▶   rl▶ -     shft▶     hld chrs▶    hld ln▶        ▶  RESULT  ▶
.DATE 03 AUG 90  11:37:25     REPORT GENERATION     WEBMJG
*=================================================================================

      ROGER, NANCY, SUSIE, AND PETER ALL WORK TOGETHER.

                    ..... END REPORT .....
```

Using Variable Arrays Example 2

Scenario

A Run Designer has the need to load eighty variables. These variables could be broken down into four groups of twenty 'like' values. What is the most user-efficient way to load these variables?

Historical Solution

Initialize each value individually

Initialize a couple of variables and use 'V sub V' technique of incrementing the values of a variable

Suggested Solution

Load variable values via the @LDA command.

Use four LDA commands - each name representing 20 'like' values.

Using Variable Arrays Example 2

```
@LDA <Product>a9[20]=BLACKBOX1,BLACKBOX2,etc
@LDA <Cost>i5[20]=13500,13600,etc
@LDA <Whole>i5[20]=16875,17000,etc
@LDA <Retail>l7[20]=2362.50,2380.00,etc

.

        The ABC company would like to order 20
          <Product>[15]s at a cost of $<Retail>[15].

@BRK DSP,-0
```

Versus

```
@LDV <Product1>a9=BLACKBOX1,<Product2>a9=BLACKBOX2,\
<Product3>a9=BLACKBOX3, ... through  <Product20>a9=WHITEBOX9.
.
.
@LDV <Cost1>i5=13500,<Cost2>i5=13600, ... through <Cost20>
.
.
@LDV <Whole1>i5=16875,<Whole2>i5, ... through <Whole20>
.
.
@LDV <Retail1>i7=2362.50,<Retail2>i7=2380.00, ... through <Retail20>
```

Variable Arrays Example 3

Scenario

A Run Designer has the need to transfer control of a run to a subroutine. The Designer needs to pass eighty values to the subroutine. (Remember, the maximum allowed is only forty)

Historical Solutions

Pass the values to the subroutine as 'packed' string-type variables and later break them up.

Suggested Solution

Use the LDA command four times - loading the four groupings of values under separate array names. The individual array members could then be referenced.

Using Variable Arrays Example 3

RCR

```
@LDA <Product>a9[20]=BLACKBOX1,BLACKBOX2,etc

@LDA <Cost>i5[20]=13500,13600,etc

@LDA <Whole>i5[20]=16875,17000,etc

@LDA <Retail>i7[20]=2362.50,2380.00,etc

@CALL ,0,E,20 2 (<Product>,<Cost>,<Whole>,<Retail>).
```

Subroutine

```
@002:(<Product>,<Cost>,<Whole>,<Retail>).

      Company ABC would like to order 2 <Product>[2]"s
       a cost of $<Retail>[2] each.

@BRK DSP,-0 .
```

Output is Displayed

```
      Company ABC would like to order 2 Blackbox2's at
      a cost of $2380.00 each.
```

Documenting and Converting Variables

- **BVT run**
  - Builds table displaying location of all variables in RCR
  - Tells line numbers where each variable is defined and used
  - Table is placed in a result at the end of RCR
- **BVT,Q**
  - Lists only the variables that have been defined
- **CVT**
  - Used to convert V-type variables to named variables
- **CVT,N**
  - Converts from named variables to V-type variables
- **CVZ**
  - Converts all numeric variables (V1) to 3 characters (V001)
- **CVT and CVZ use an existing table**
- **F1 (RSM) replaces RCR with the result created by the variable table results**

Documenting Variables

Scenario

A Run Designer wishes to devise a plan for keeping track or 'documenting' variables used in a run control report.

Historical Solutions

Variable Dump - After all variables in run are initialized, use an @GTO label to 'dump' them into output area along with a brief description of each variable.

Variable Dump into another report - Designers have also 'dumped' variables out to a different report as well by using @RSR.

Variable Worksheet - a self designed, 'pad and paper' method of writing down variable information for documentation purposes.

Suggested Solution

Use the BVT run. This run automatically creates a result of your RCR which contains system supplied variable information. Additional information can also be added to the table via SOE updates.

Documenting Variables

- 'BVT' entered on line zero of displayed RCR

```
LINE▶ BVT   FMT▶   RL▶ -     SHFT▶     HLD CHRS▶    HLD LN▶        ▶   8E0      ▶
.DATE 25 MAY 90  09:30:58  RID      8E   25 MAY 90  USER
.TEST                                               BY:                    E000010
*==========================================================================
@BRK .
@LDV <SOFTWARE>A6=MAPPER,<FUNCTION>A6=SEARCH .
@SRH,0,B,2 '' 'ST CD','PRODUCT TYPE(1-5)'  ,IP,BLACK  <FOUND>I3,<SCANNED>I3 .

      I AM USING <SOFTWARE> TO <FUNCTION> A REPORT.

         <FOUND> OUT OF A POSSIBLE <SCANNED> MET MY <FUNCTION> CONDITION.

@BRK OUT,-0,2,5,1 .
                        ..... END REPORT .....
```

- Result of RCR displayed containing Variable Table
- Press F1 to replace the RCR with the BVT result

```
LINE▶ 13    FMT▶   RL▶          BVT Complete - F1 to Replace   ▶    RESULT    ▶
.VARIABLE TABLE
*    NAME     .VNUM.SQ.            LINE  NUMBERS              .      COMMENT
*------------ ---- --- ------------------------------------- . ------------------
 SOFTWARE     V001 01 5*,8
 FUNCTION     V002 01 5*,8,10
 FOUND        V003 01 6*,10
 SCANNED      V004 01 6*,10
                        ..... END REPORT .....
```

Documenting Variables

- RCR is replaced with BVT table added at end

```
                          Your Report has been Replaced
.DATE              09:32:40  RID      8E   25 MAY 90  USER
.TEST                                                 BY:                    E000010
*=============================================================================
@BRK .
@LDU <SOFTWARE>A6=MAPPER,<FUNCTION>A6=SEARCH .
@SRH,0,B,2 '' 'ST CD','PRODUCT TYPE(1-5)'  ,IP,BLACK  <FOUND>I3,<SCANNED>I3 .

      I AM USING <SOFTWARE> TO <FUNCTION> A REPORT.

         <FOUND> OUT OF A POSSIBLE <SCANNED> MET MY <FUNCTION> CONDITION.

@BRK OUT,-0,2,5,1 .

.VARIABLE TABLE
*    NAME     .VNUM.SQ.            LINE  NUMBERS              .      COMMENT
*============.====.==.==============================.==================
 SOFTWARE     V001 01 5*,8
 FUNCTION     V002 01 5*,8,10
 FOUND        V003 01 6*,10
 SCANNED      V004 01 6*,10
                      ..... END REPORT .....
```

Table Field Definitions

NAME	Displays the variable name (default = N000).
VNUM	Displays the V-type number.
SQ	Sequence number used to match and save user comments. Identifies how many lines in this table are associated with this variable.
LINE NUMBERS	Line number on which variables are located. Asterisks indicate line where variable was initialized or reinitialized.
COMMENT	User-supplied comments entered via report update.

Variable Conversions

Scenario

A Run Designer has written a run which utilizes 'named' variables. This is a common practice today because the use of names, instead of 'V' numbers makes the run more maintainable. The drawback to this technique is that the use of 'named' variables is less efficient to the system when processing.

Historical Solutions

Copy the RCR and change 'named' variables to numbered variables. Put the RCR with the numbered variables into production.

Live with the higher cost of processing the run.

Suggested Solution

Write the run using 'named' variables and then use CVT,N. RCR will automatically convert named variables to numbered variables.

Put 'numbered variable' copy into production. Keep 'named variable' copy for maintenance purposes -or-

Keep only one copy with numbered variables and use CVT to convert the numbered variables back to named variables for maintenance.

- There would be a trade-off here:
  - Keeping only one copy would be more efficient to the system - less disk space would be utilized.
  - Keeping two copies would probably be more 'user-efficient'.

Variable Conversions

- Convert all of the named variables to numbered variables via CVT,N

```
Line▶ 1          Roll▶ BYE,N                                    8E0
.DATE            09:32:40  RID      8E   25 MAY 90  USER
.TEST                                                BY:                    E000010
*=========================================================================
@BRK
@LDV <SOFTWARE>A6=MAPPER,<FUNCTION>A6=SEARCH .
@SRH,0,B,2 '' 'ST CD','PRODUCT TYPE(1-5)'  ,IP,BLACK  <FOUND>I3,<SCANNED>I3 .

      I AM USING <SOFTWARE> TO <FUNCTION> A REPORT.

        <FOUND> OUT OF A POSSIBLE <SCANNED> MET MY <FUNCTION> CONDITION.

@BRK OUT,-0,2,5,1 .

.VARIABLE TABLE
*    NAME     .VNUM.SQ.            LINE  NUMBERS                .      COMMENT
*============.====.==.=================================.==================
 SOFTWARE     V001 01 5*,8
 FUNCTION     V002 01 5*,8,10
 FOUND        V003 01 6*,10
 SCANNED      V004 01 6*,10
                      ..... END REPORT .....

 1Resume 2Paint  3       4Return 5       6Tasks  7View   8Help   9      10Edit
```

- Result is displayed
- Press F1 to replace RCR with CVT result

```
LINE▶ 1     FMT▶   RL▶          CVT Complete - F1 to Replace  ▶    RESULT    ▶
.DATE 25 MAY 90  09:33:26  RID      8E   25 MAY 90  USER
.TEST                                                BY:                    E000010
*=========================================================================
@BRK
@LDV V1A6=MAPPER,V2A6=SEARCH .
@SRH,0,B,2 '' 'ST CD','PRODUCT TYPE(1-5)'  ,IP,BLACK  V3I3,V4I3 .

      I AM USING V1 TO V2 A REPORT.

        V3 OUT OF A POSSIBLE V4 MET MY V2 CONDITION.

@BRK OUT,-0,2,5,1 .

.VARIABLE TABLE
*    NAME     .VNUM.SQ.            LINE  NUMBERS                .      COMMENT
*============.====.==.=================================.==================
 SOFTWARE     V001 01 5*,8
 FUNCTION     V002 01 5*,8,10
 FOUND        V003 01 6*,10
 SCANNED      V004 01 6*,10
                      ..... END REPORT .....

 1Resume 2Paint  3       4Return 5       6Tasks  7View   8Help   9      10Edit
```

Exercise

1. Write a run to perform the following:
   a. Initialize 10 variables (v1 - v10) using several @CHG statements.
   b. Add a report to the 'A' drawer.
   c. Write the values (v1 - v10) to the 'A' report.
   d. Clear variables v1 - v10 by using @CLV.
   e. Reinitialize v1 - v10 with new values.
   f. Write these values to the 'A' report.
   g. Once again, clear v1- v10 (@CLV).
   h. Read the 'A' report and re-assign v1 - v10 to the original values.
   i. Add an @LOG statement to the beginning of this exercise.
   j. Execute the run and print out the LOG result.
2. Write an exercise to perform the following:
   a. Initialize 10 variables (v1- v10) using @LDV.
   b. Save these values on a variable stack.
   c. Reinitialize v1- v10 with new values.
   d. Retrieve the original values off of the stack.
   e. Add an @LOG statement to the beginning of this run.
   f. Execute the run and print out the LOG result.
3. Compare the two LOG results.
4. EXTRA: Repeat exercise number 2, but use @LDA to load 10 'like values.'