UMTA-VA-06-0065-80-1

DETAILED DESIGN FOR A MIS FOR THE SOUTHERN CALIFORNIA RAPID TRANSIT DISTRICT

OCTOBER 1980

T
58.6
.L82
c.1

PREPARED FOR:
U.S. DEPARTMENT OF TRANSPORTATION
URBAN MASS TRANSPORTATION ADMINISTRATION
Office of Technology Development and Deployment
Washington, D.C. 20590

This document was disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for its contents or use thereof. The United States Government does not endorse products or manufacturers. Trade or manufacturer's names appear herein solely because they are considered essential to the object of this report.

Technical Report Documentation Page

1. Report No.
UMTA-VA-06-0065-80-1

2. Government Accession No.

3. Recipient's Catalog No.

4. Title and Subtitle
Detailed Design for a MIS for the
Southern California Rapid Transit District

5. Report Date
OCTOBER 1980

6. Performing Organization Code

7. Author(s)
John S. Ludwick, Jr.

8. Performing Organization Report No.
MTR-80W84

9. Performing Organization Name and Address
The MITRE Corporation, Metrek Division
1820 Dolley Madison Boulevard
McLean, Virginia 22102

10. Work Unit No. (TRAIS)

11. Contract or Grant No.
DOT-UT-90006

12. Sponsoring Agency Name and Address
U.S. Department of Transportation
Urban Mass Transportation Administration
Office of Technology Development and Deployment
Washington, D.C. 20590

13. Type of Report and Period Covered

14. Sponsoring Agency Code

15. Supplementary Notes
Ground Transportation Systems
Project Number: 1242B

16. Abstract
An Automatic Vehicle Monitoring System (AVM) being installed in Los Angeles will record a large amount of operational data that can later be used for management reports. The software required to provide this function is described and structured pseudo-code of the processes and detailed file descriptions are given.

17. Key Words
Automatic Vehicle Monitoring, Bus Transit Management Information System, Software Design

18. Distribution Statement
Available to the Public through the National Technical Information Service, Springfield, Virginia 22161.

19. Security Classif. (of this report)
Unclassified

20. Security Classif. (of this page)
Unclassified

21. No. of Pages

22. Price

Reproduction of completed page authorized

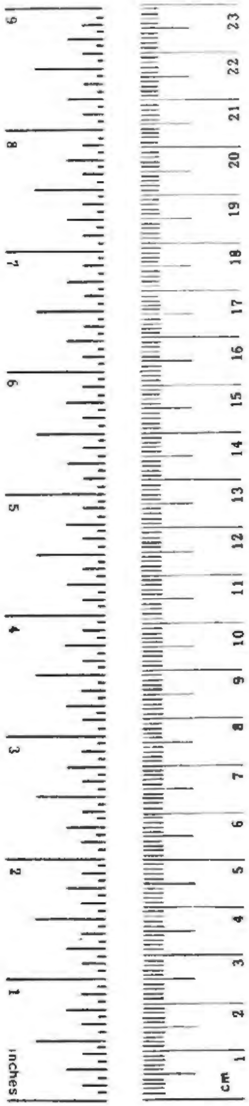
T
58.6
.L82
c.1

METRIC CONVERSION FACTORS

Approximate Conversions to Metric Measures

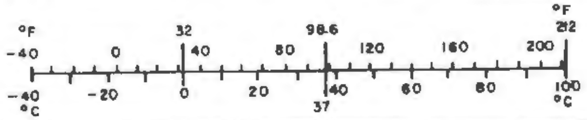
Symbol	When You Know	Multiply by	To Find	Symbol
		LENGTH		
in	inches	*2.5	centimeters	cm
ft	feet	30	centimeters	cm
yd	yards	0.9	meters	m
mi	miles	1.6	kilometers	km
		AREA		
in^2	square inches	6.5	square centimeters	cm^2
ft^2	square feet	0.09	square meters	m^2
yd^2	square yards	0.8	square meters	m^2
mi^2	square miles	2.6	square kilometers	km^2
	acres	0.4	hectares	ha
		MASS (weight)		
oz	ounces	28	grams	g
lb	pounds	0.45	kilograms	kg
	short tons (2000 lb)	0.9	tonnes	t
		VOLUME		
tsp	teaspoons	5	milliliters	ml
Tbsp	tablespoons	15	milliliters	ml
fl oz	fluid ounces	30	milliliters	ml
c	cups	0.24	liters	l
pt	pints	0.47	liters	l
qt	quarts	0.95	liters	l
gal	gallons	3.8	liters	l
ft^3	cubic feet	0.03	cubic meters	m^3
yd^3	cubic yards	0.76	cubic meters	m^3
		TEMPERATURE (exact)		
°F	Fahrenheit temperature	5/9 (after subtracting 32)	Celsius temperature	°C

*1 in ≡ 2.54 (exactly). For other exact conversions and more detailed tables, see NBS Misc. Publ. 286, Units of Weights and Measures, Price $2.25, SD Catalog No. C13.10:286.



Approximate Conversions from Metric Measures

Symbol	When You Know	Multiply by	To Find	Symbol
		LENGTH		
mm	millimeters	0.04	inches	in
cm	centimeters	0.4	inches	in
m	meters	3.3	feet	ft
m	meters	1.1	yards	yd
km	kilometers	0.6	miles	mi
		AREA		
cm^2	square centimeters	0.16	square inches	in^2
m^2	square meters	1.2	square yards	yd^2
km^2	square kilometers	0.4	square miles	mi^2
ha	hectares (10,000 m^2)	2.5	acres	
		MASS (weight)		
g	grams	0.035	ounces	oz
kg	kilograms	2.2	pounds	lb
t	tonnes (1000 kg)	1.1	short tons	
		VOLUME		
ml	milliliters	0.03	fluid ounces	fl oz
l	liters	2.1	pints	pt
l	liters	1.06	quarts	qt
l	liters	0.26	gallons	gal
m^3	cubic meters	35	cubic feet	ft^3
m^3	cubic meters	1.3	cubic yards	yd^3
		TEMPERATURE (exact)		
°C	Celsius temperature	9/5 (then add 32)	Fahrenheit temperature	°F



03898

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (Concluded)

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

A demonstration of an Automatic Vehicle Monitoring (AVM) system, sponsored by the Urban Mass Transportation Administration (UMTA), is being performed in Los Angeles, with approximately 200 buses and four lines of the Southern California Rapid Transit District (SCRTD). Although the primary purpose of the system is to determine the effectiveness of various on-line control strategies, a large amount of data is recorded daily and can be used to provide management information. A previous Working Paper (WP-79W00738) detailed requirements for MIS reports using AVM data, and a later letter (W24-5026) presented a preliminary design that would provide the required data.

This document expands the preliminary design to a detailed design. In Section 2, the latest data flow diagrams are given, and the inputs, outputs, and processing required by each process are described. Section 3 discusses the structured approach and conventions of the pseudo-code used to describe the processing. Appendix A alphabetically presents the contents of the files and some of the tables referred to in Section 2 and Appendix B presents the detailed pseudo-code described in Section 3.

The contents of this document include revisions based on discussion with SCRTD personnel of an earlier draft copy; it is anticipated that further refinements will result during the implementation phase.

2. PROGRAM DESCRIPTIONS

The programs required to process the AVM log tape and eventually provide MIS reports are described here. The general format is to discuss the required inputs to each program, the resultant outputs and the processing required. The programs follow from the data flow diagrams developed during the preliminary design, although some of the processes in that document have been further decomposed and some modifications have been made. Figure 2-1 shows the first level processes, Figure 2-2 and 2-3 show the second level, and Figures 2-4 through 2-7 show the third and lower levels. Figure 2-8 shows an expansion of Process 1: Form AVM Files.

2.1 Form AVM Files

To summarize the processes involved (see Figure 2-2), the daily AVM log tape is processed by Gould Fortran programs, forming data files which are further selected, sorted and processed by Cobol programs. A research tape is formed including all of the AVM data bus stop; this will probably require a new tape every day. A daily TP-trip file is formed, including, for each time point on each trip, the basic information that is needed to later provide MIS reports. For each report that is to be provided, a separate daily aggregate file is formed from the TP-trip file and is used to update a summary tape, which maintains statistical data from the beginning of the desired time period to the present. At any time, the summary tape can be accessed by report programs to format and provide final totals.

In general, the program names used are the same as the processes on the data flow diagrams. As there are a large number of files involved, and they are referred to not only in this section, but also in the next, the details are not given here, but are provided in the Appendix, where the files and some tables are presented in alphabetical order.

2.1.1 Gould Fortran Programs

A set of programs has been developed by Gould to read the AVM log tape and form a set of files that they will later use to determine the effectiveness of various control strategies. Although the programs were not originally intended for use at SCRTD, a Fortran compiler is available on the Univac system, and it should be possible to use the same programs perhaps with a few modifications to accommodate differences between the different Fortran implementations. Although all the data

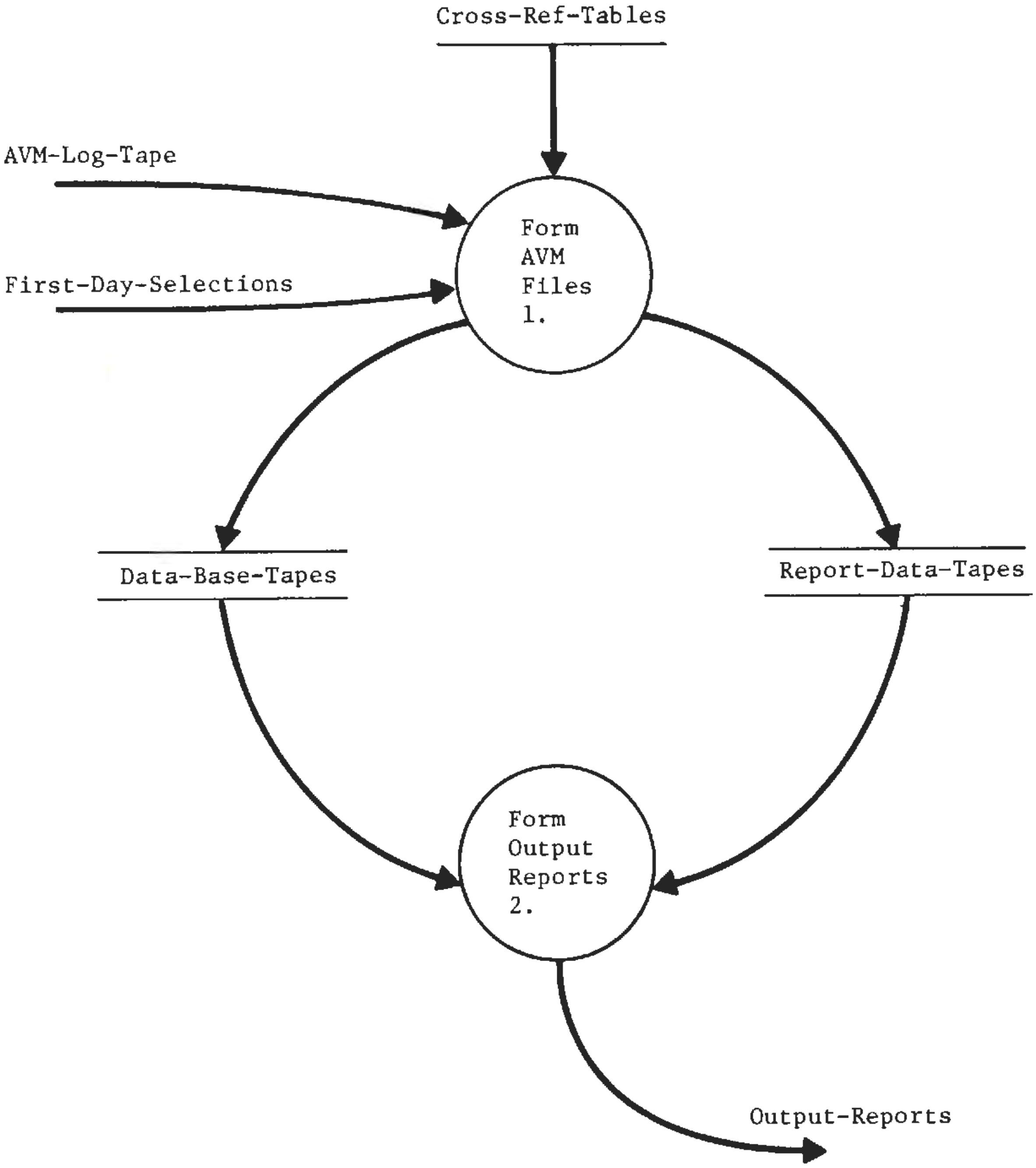


FIGURE 2-1. MIS DATA FLOW

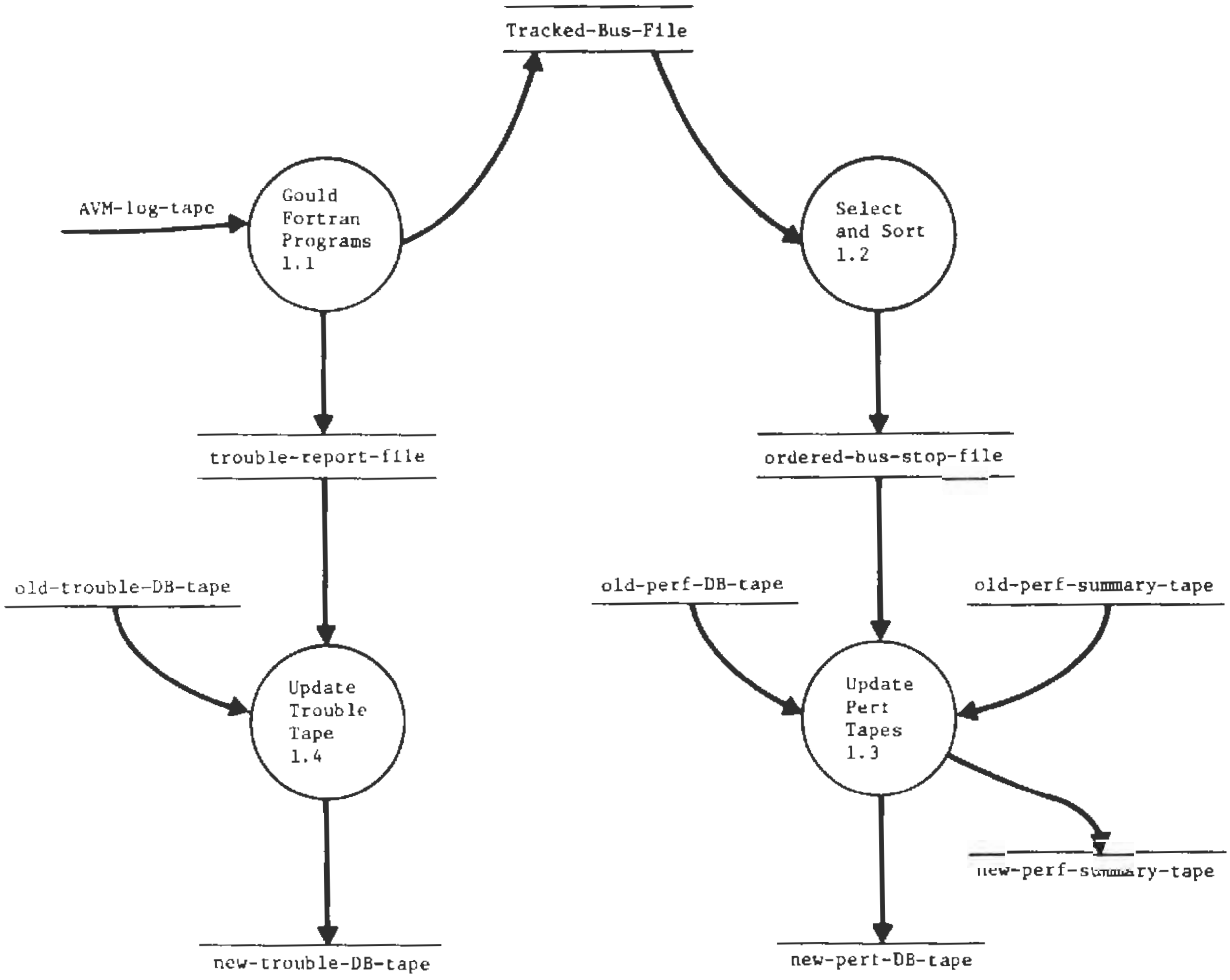


FIGURE 2-2. PROCESS 1: FORM AVM FILES

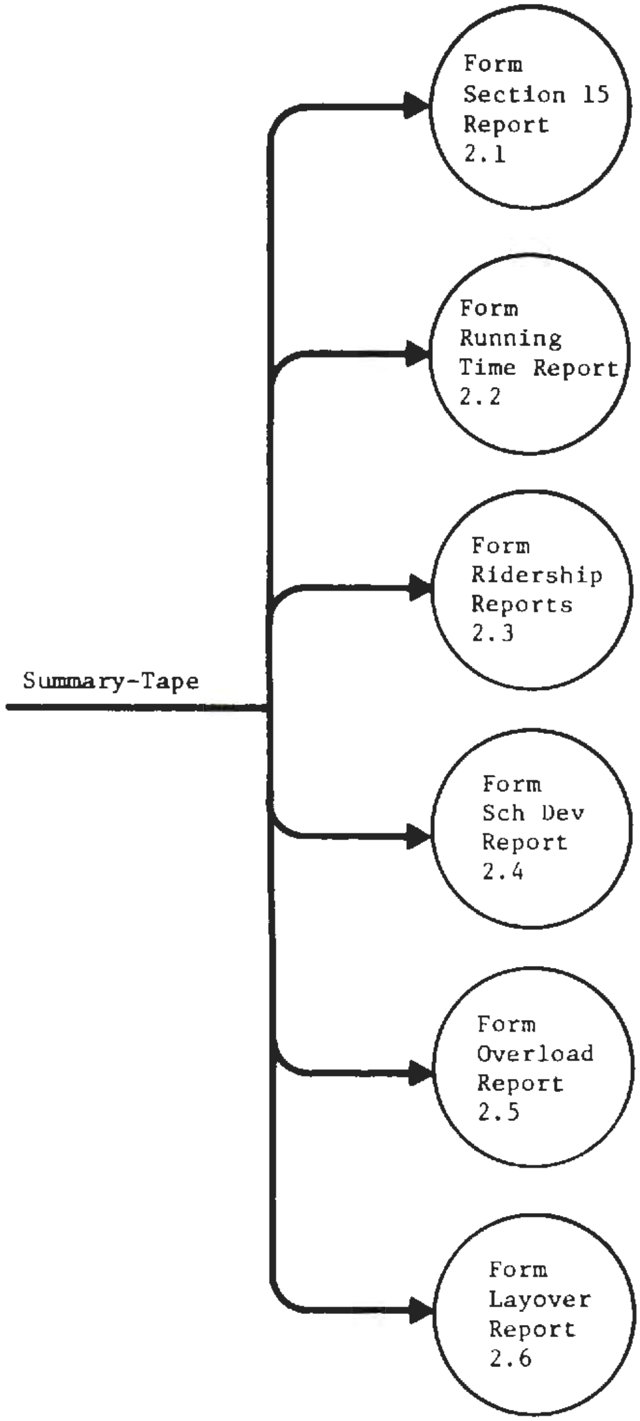


FIGURE 2-3. PROCESS 2: FORM OUTPUT REPORTS

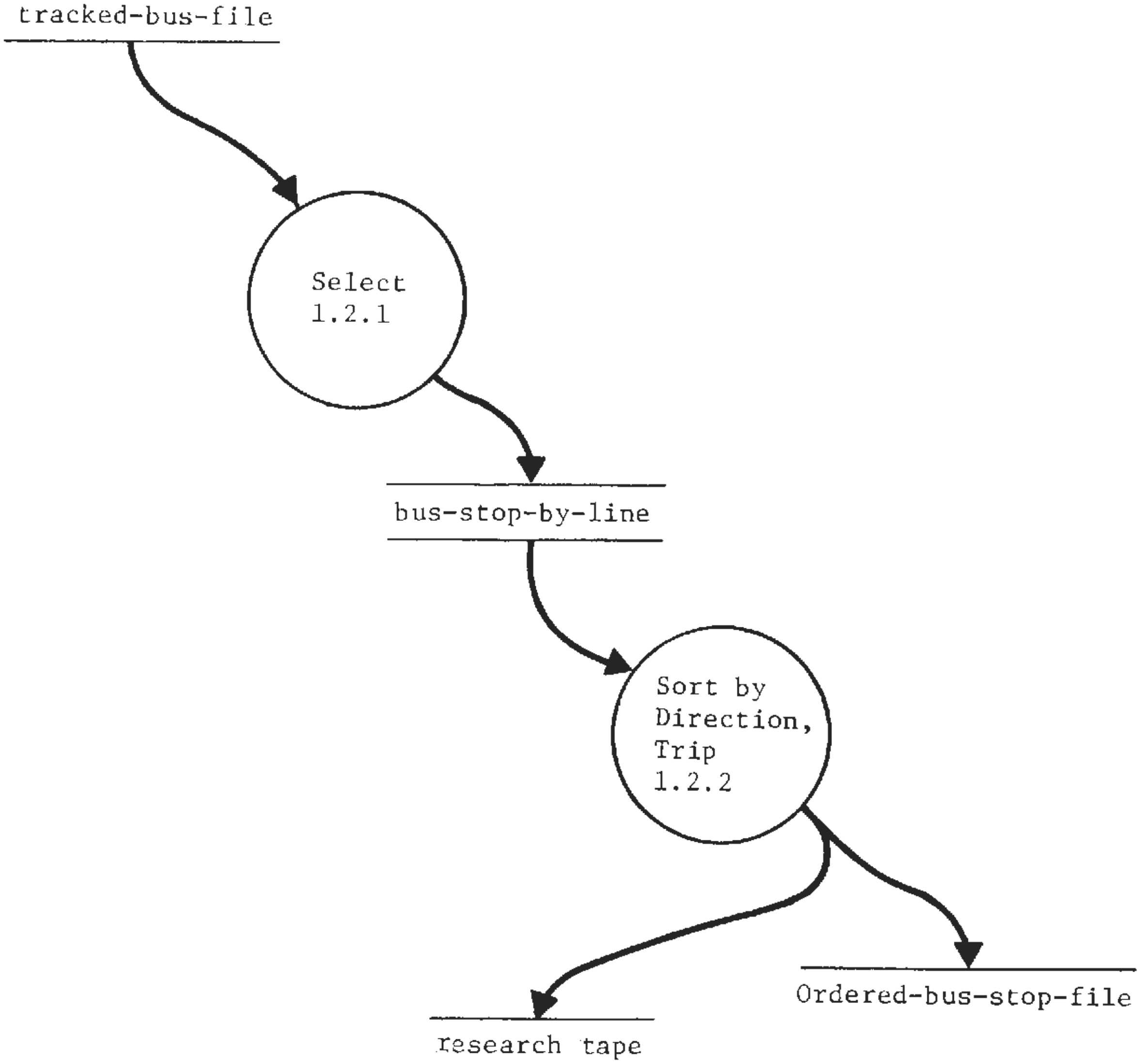


FIGURE 2-4. PROCESS 1.2: SELECT AND SORT

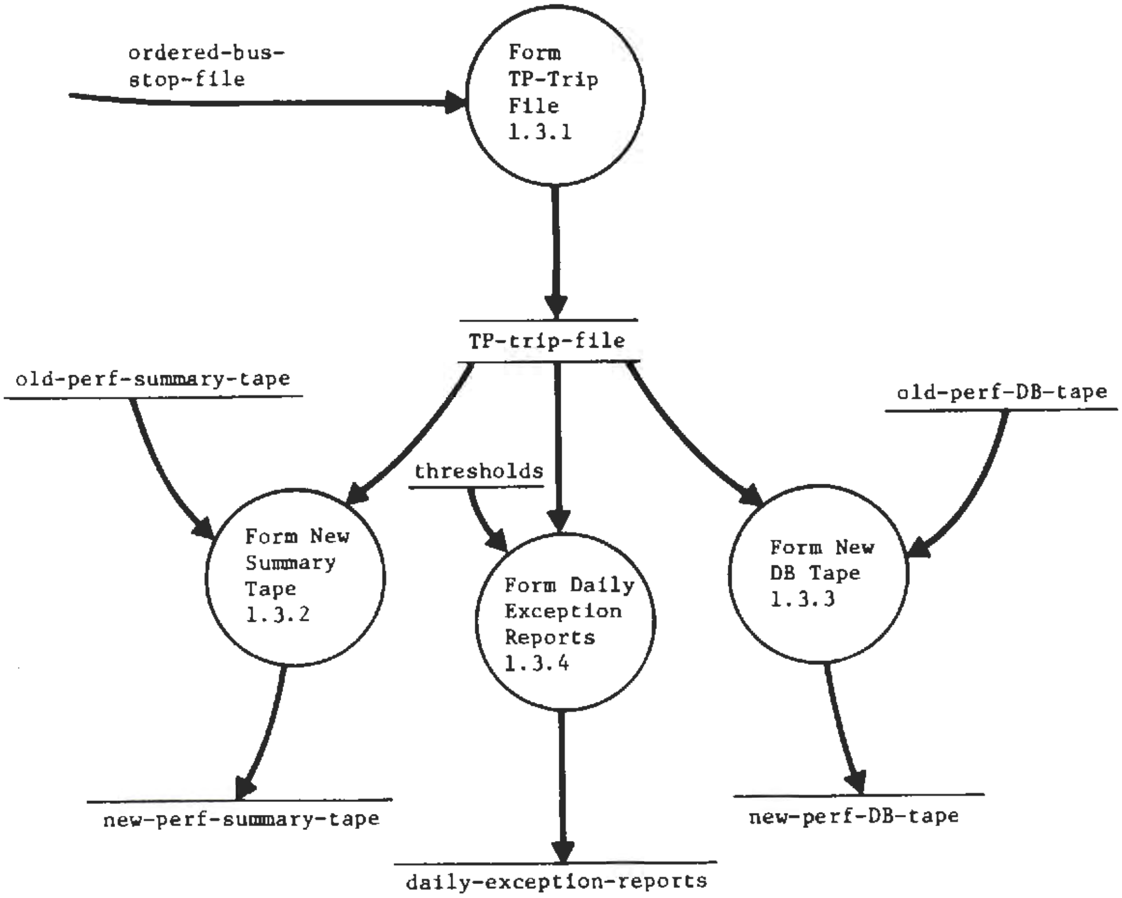
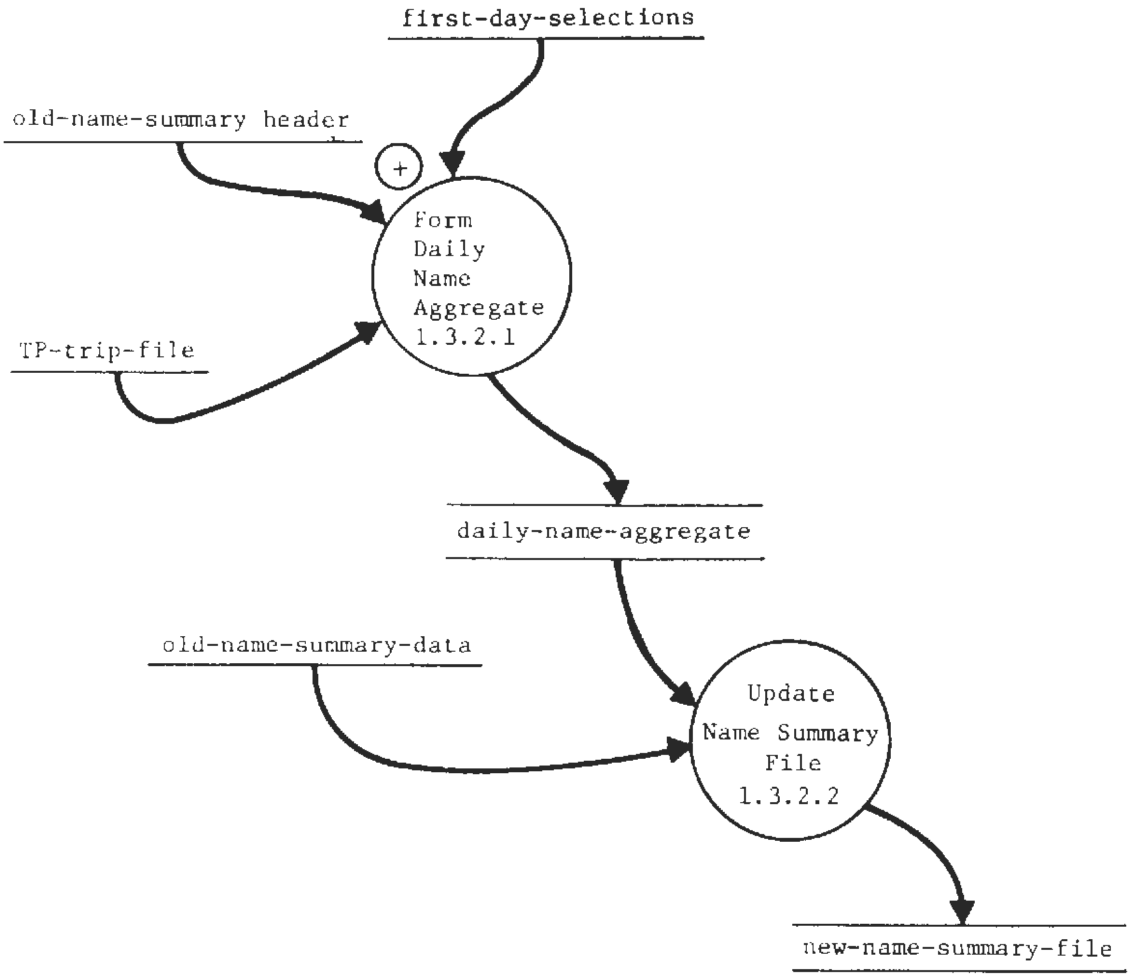


FIGURE 2-5. PROCESS 1.3: UPDATE PERF TAPES



where name = S15
Run
Ride
Sch Dev
Overload
Layover

FIGURE 2-6. PROCESS 1.3.2: FORM NEW SUMMARY TAPE

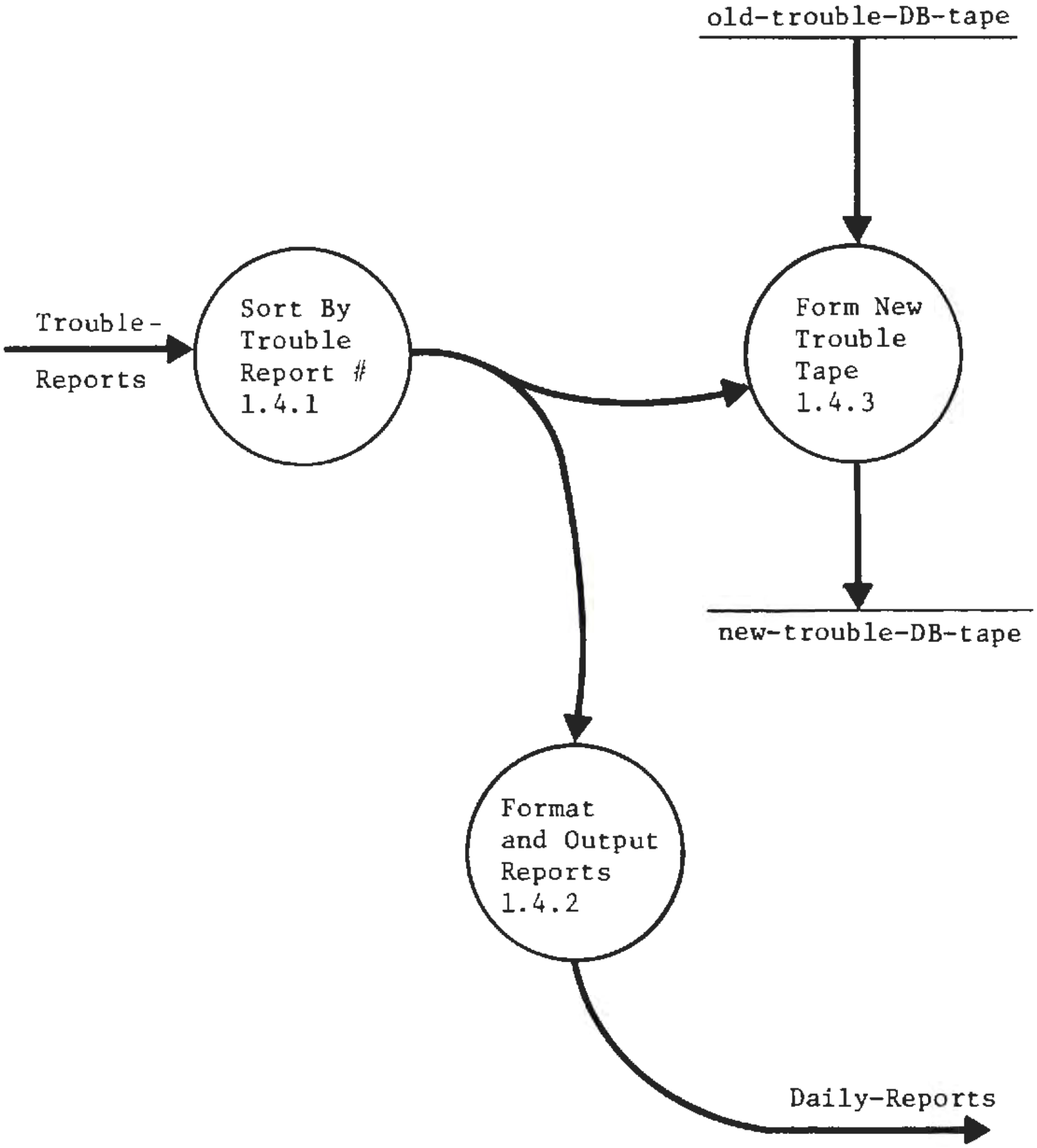


FIGURE 2.7. PROCESS 1.4: UPDATE TROUBLE TAPE

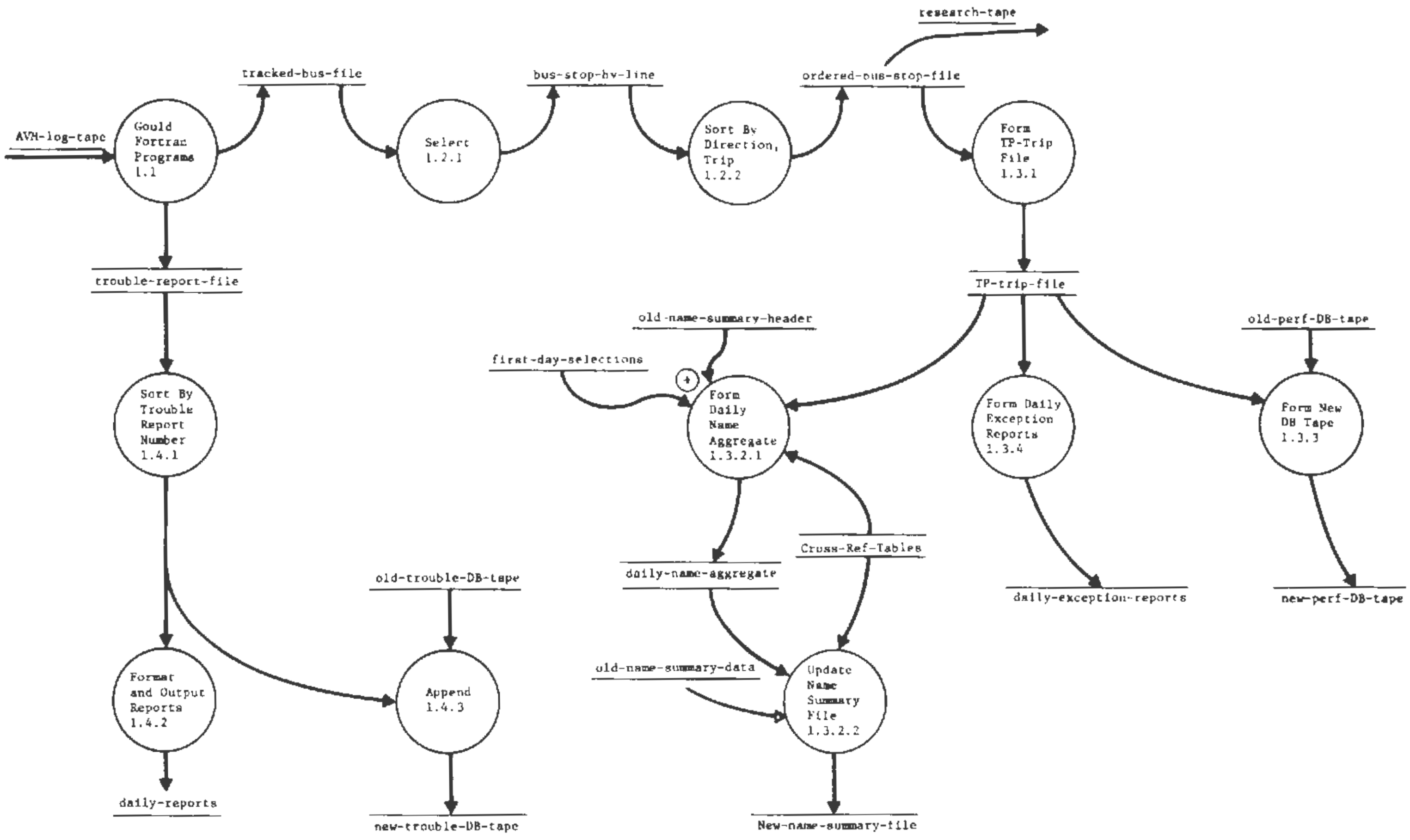


FIGURE 2-8. PROCESS 1, EXPANDED

available from these programs are not needed to satisfy reporting requirements, it is less complicated to use the existing programs to form files which will then be read by Cobol programs developed by SCRTD than to build a new set of programs from scratch to provide just the data needed. Also, all the data provided by one of the Gould files will be sorted and copied onto a tape which will be available for later analysis by researchers.

2.1.1.1 Input

The AVM log tape is the primary input. It is possible that some control data will also need to be provided.

2.1.1.2 Output

Four types of files can be provided. Two types of files are appropriate; the other two, which are not yet defined in detail, are not needed here and will not be output. This will be accomplished by skipping calls in the main program or by providing dummy JCL when the job is run. The files useful for MIS purposes, the Trouble Report File and the Tracked Bus Data File, will be stored on disk for later processing. The Trouble Report File consists of all closed trouble reports, written in 1200 (2-byte) word blocks, each block containing a separate trouble report. Since trouble reports may not appear sequentially (by number) on the raw tape, the first block of this file will contain block pointers indexed by trouble report number (i.e., word 13 of block 1 contains the block number of trouble report 13). The Tracked Bus Data File contains a time history of bus transactions at each stop on each AVM line. It is organized into 2,040 word blocks; each block contains 51 logical records 40 words in length. There are four record types:

1. Block Header Record--First 40 words of each block, providing common information,

2. Bus Stop Record--Primary data on a bus transaction at a stop,

3. Bus Assignment/Pullout Record--Data on line/run assignments, pullouts and pullins, and

4. Dispatcher Action Record--Describes completed AVM function key actions by dispatcher.

Each of these records is defined in detail in Table 2-1 and Table 2-2. Records are not sorted and appear in the order in which the event occurred regardless of record type or context.

2.1.1.3 Processing

The details of the processing have been provided by Gould to SCRTD and are not within the scope of this document.

2.1.2 Select and Sort

Although the data in the Tracked Bus File are in time sequential order, later processing is facilitated if the data records are ordered, from major to minor: bus line, bus direction, bus trip and time at bus stop. Rather than do one large sort on what may be over 80,000 records per day, a two-step process will be performed.

2.1.2.1 Select

2.1.2.1.1 Input

The Tracked Bus Data File, previously described, is the input.

2.1.2.1.2 Output

Four files will be created, one for each AVM line, called here Bus Stop By Line. The four files will be copied, one after the other to form the research tape.

2.1.2.1.3 Processing

The first Block Header Records read will provide data for the Bus Stop Record Header. Later ones will be ignored as will Bus Assignment/Pullout Records and Dispatcher Action Records; only Bus Stop Records will be read and written into the appropriate file.

2.1.2.2 Sort

2.1.2.2.1 Input

Each of the four Bus Stop By Line files will be used as input, one at a time.

2.1.2.2.2 Output

A single file, the Ordered Bus Stop File.

TABLE 2-1 - BLOCK HEADER RECORD (TRACKED BUS FILE)

BLOCK HEADER RECORD (1ST 40 WDS OF EACH BLOCK)

	DESCRIPTION	MNEMONIC	UNITS	FORMAT	NOTES
	Time of Day	TOD1		I2	Starting Time (AVM System Time) Of This Block
		TOD2		I2	
		TOD3		I2	
	Date	MO		I2	1-12
		DAY		I2	1-31
		YR		I2	79, 80, 81
	Day of Week			A4	M, T, W, T, F, S, S
	Weather			A2	CL, RA, CY, FG
	AVM Test Period			I2	# of 2 wk TP Since Start of Exps.
1	Day of TP			I2	# of Day this TP (1-14)
2	Line Number			I3	RTD Line Number
3	Schedule Number			I5	RTD Schedule Number
4	AVM Configuration			I2	1-4
5					
6					
7	Repeated for				
8	Each AVM Test				
9	Line				
10					
11					
12					
13					
14					
15					

SOURCE: Gould, Inc.

TABLE 2-2 - BUS STOP RECORD (TRACKED BUS FILE)

	DESCRIPTION	MNEMONIC	UNITS	FORMAT	NOTES
1	Time of Day	TOD1	Hrs.	I2	Military Time
2		TOD2	Mins.	I2	
3		TOD3	Secs.	I2	
4	Line #	LI		I3	RTD Line # (On Which Bus is Operating)
5	Run #	RU		I5	RTD Run # (or LI/RU for)
6	Bus #	BU		I4	RTD Bus #
7	Trip #	TRP		I4	RTD Trip #
8	Direct Code	DIR		I2	RTD DIR CODE 1 = N 3 = S 13,31 RND 2 = E 4 = W 24,42 TRIPS
9	Origin Code	OC		I2	COL # of ORIG. From Sched.
10	Destination Code	DC		I2	" " " DEST. " "
1	Stop #	STP		I4	RTD STOP #
2	Time Point Code	TPC		I2	If stop is TP, RTP COL #
3	Sched. Deviation	SDEV	Secs	I5	As Adjusted, From TOD
4	Sched. Adjustment	SADJ	Secs	I5	(As Specified by Dispatcher)
5	Sched. Headway	SHDW	Secs	I5	As Adjusted
6	Headway Deviation	DHDW	Secs	I5	
7	Passengers On-Board	POB		I3	At Arrival at this Stop
8	Passengers Boarding	PB		I3	At this Stop
9	Passengers Alighting	PA		I3	" " "
20	Total Boardings	TB		I3	For Trip
1	Total Alightings	TA		I3	" "
2	Passengers On-Board Leader	POBL		I3	At Arrival at this Stop
3	Load Difference	PDIF		I3	POB-POBL
4	Running Time Increment	RTS	Secs	I5	From Last Stop
5	Passenger Waiting Time	PWT	Secs	I5	PB x (SHDW-DHDW) x 0.4
6	Passenger Trip Lot	PTL	10 Ft	I5	POB x Dist. from last stop
7	Passenger Trip Time	PTT	Secs	I5	POB x RTS
8	Passenger Trip Delay	PTD	Secs	I5	(PB-PA) x SDEV
9	Tactical Sit Code			I2	1-11
30	TRBL REPORT # (Open)			I3	1-999
1	SPM Position			I1	0-7 0 = Off, 7 =
2	Tactics in Effect 1			I2	MSP MSGS + Voice/
3	2			I2	
4	3			I2	
5	Other --Uplink/Dnlink Flgs			B16	
6					
7					
8					
9					
40					

2.1.2.2.3 Processing

The Univac sort utility will be used to sort the records by direction and trip. As they have already been separated by line the files can simply be concatenated after sorting. As the records are originally in time sequence, after sorting by direction and trip, they would still be in time sequence within trips, if the Univac sort utility were "stable". As it has been learned that the sort is not stable, the sort will have to be by direction, trip and time.

2.1.3 Update Perf Tapes

This is the most complex process of the four second level processes in Form AVM Files. First a file is formed that includes data only at time points, but includes enough data to provide all required output reports. This file is then appended to previous Detailed Data Base files to form a New Perf Data Base Tape. It is also used to form a New Summary Tape, described later, and to form Daily Exception Reports for schedule deviations and for overloading.

2.1.3.1 Form TP-Trip File

2.1.3.1.1 Input

The Ordered Bus Stop File.

2.1.3.1.2 Output

The TP-Trip File, including for each trip, the direction, run andbus numbers, for each time point during the trip, the time at the time point, the adjusted schedule deviation, number of passengers aboard at that time point, number of passengers on and off between the time point and the previous one, the maximum number of passengers aboard at any bus stop between time points and the number of the bus stop where that maximum occurs.

2.1.3.1.3 Processing

The Ordered Bus Stop records are read one at a time and at each bus stop the maximum number of passengers aboard since the last time point is updated, as well as the identification of the maximum passenger bus stop. At each time point, the number of passengers boarding and alighting since the last time point is computed by subtracting, from the latest totals, the totals recorded at the previous time point. The adjusted schedule deviation is taken as the schedule deviation plus the schedule adjustment. Passengers aboard is taken directly from the bus stop record time. To allow later determination of layover time and run time to the first time point, data from the first and last bus stop records received during a bus trip are also saved, with unique identifiers stored instead of the time points.

2.1.3.2 Form New Summary Tape

This is the most complex process among the second level process in Figure 2-5, and is further decomposed in Figure 2-6. Basically, a two-part procedure is followed to create each new summary file: first, the daily TP-Trip File is read and the data processed to provide statistics aggregated in the same format as is presently provided on the Old Summary Tape. The Daily Aggregate File is then read along with the Old Summary File of the same type; corresponding data elements (i.e., the same line, direction, day-of-week aggregation, time-of-day aggregation, and time point) are combined, and a New Summary File is written.

Figure 2-9 shows the general layout of the various files and headers on the Summary Tape. It is anticipated that a given summary tape will accumulate data over a monthly period, although other periods could also be used. The Summary Tape will remain essentially the same length after each update, as new data is combined with previous data in each file, not appended to it; therefore there is no real limit on how long a given summary tape can be used to accumulate data. The first day of a summary, the files to be included are selected and the aggregation periods and limits required by each are provided as inputs. That data is then included in the header of each summary file, and is read and used to control the aggregation and selection for the rest of the summary time period. That is, the files selected to be summarized and the criteria to be used for each remain the same for a given summary period, but may be changed at the beginning of the next period.

2.1.3.2.1 Form Daily Aggregate File

A different program of this type is required for each of: Section 15, Running Time, Ridership (two types), Schedule Deviation, Overloading (in addition to daily Schedule Deviation and Overloading Reports, the data for which are not summarized here).

Summary Tape:

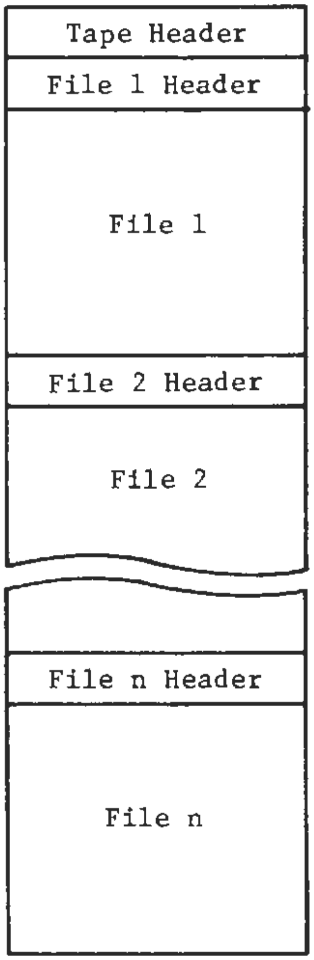


FIGURE 2-9. SUMMARY TAPE LAYOUT

In general, all programs of this type operate in the same manner and provide an output similar to that shown in the Apendix for the summary files. The weather-condition and school-condition pertain to the entire day's operation. The day-interval and time-interval refer to the manner in which the data are aggregated and will be further described below.

To allow flexibility in choice of aggregation period, an approach using user-defined table input has been chosen. The layout of the time-of-day (TOD) table is shown in the Appendix. A TOD code first indicates if aggregation by time period is desired, or whether all trips are to be individually saved. If aggregation is to be performed, the table-length tells how many table entries follow. Each entry consists of a finish-time for an interval (the start-time is assumed to be the previous finish-time; the first start-time is assumed as 4:00 as SCRTD defines their shcedule day to begin at 4:00 A.M. The day and date saved in various headers refer to the beginning of the schedule day). For example, the entries 5:00, 6:00, 7:00, 7:20, 7:40, 8:00 would define:

Time-Interval	Times Included	
1	4:00	5:00
2	5:00	6:00
3	6:00	7:00
4	7:00	7:20
5	7:20	7:40
6	7:40	8:00

The time at which a trip begins determines the time-interval into which that trip falls, and summations, averages or maximums are carried out over all data occuring in that time interval. As the TOD-table will be included in the Old Summary File header after its information, the same time intervals will be used to create each later Aggregation File, until the next period begins.

A similar technique is used for day-of-week (DOW) aggregation, only here the days need not be contiguous. The Appendix shows the format of the DOW-table, which has a DOW-code to indicate if there will be aggregations or whether each day should be handled individually; and a table-length if there is to be aggregation.

In order to accommodate all the possibiities of aggregation that may be desired, each table entry includes seven positions, corresponding to day of the week (Monday = 1, Sunday = 7). A non-zero table entry indicates that that day will be included in the aggregation for that day-interval, e.g.,

.	.	.	.	.	.	.
1				1		
	1	1	1			
					1	1

would accumulate Monday and Friday as day-interval 1, Wednesday through Thursday as day-interval 2 and Saturday and Sunday as day-interval 3. As each Aggregation File involves only one day's data, there is no DOW aggregation performed during this process; it occurs during the Update Summary process.

The use of table input to define aggregation intervals allows as much flexibility as the user desires, without having to modify the programs.

Although the computations required to form the various daily aggregate files are different, the technique is basically the same in each case. Individual records are read from the TP-Trip File, the necessary computations are performed, sometimes involving data from the previous time point as well as the current one, and the results are used to update the element of a vector corresponding to that time point. When the next record read is for a new trip, the start time is checked to determine if a new time interval has been entered. If not, processing proceeds as before with the same elements being updated by new data; if so, statistics are computed for each vector element written out, and computations for the new time interval are begun. In addition to between-time-point computations, trip totals are computed and stored as the last TP element. In the detailed descriptions that follow, only the computational differences will be discussed.

2.1.3.2.1.1. Form Daily Section 15 Aggregate File

Data on passenger ridership is required yearly in UMTA Section 15 reports. All the needed data can be derived from AVM data, and simulation using actual passenger loadings from SCRTD Line 44 has shown that accuracy requirements can be met using passenger loadings only at time points.

2.1.3.2.1.1.1 Input

The TP-Trip File.

2.1.3.2.1.1.2 Output

The S15-Aggregate Files containing passengers boarded, bus miles, passenger miles, bus minutes, passenger minutes, capacity miles, seat miles and bus trips.

2.1.3.2.1.1.3 Processing

The DOW-code and TOD-code are determined by UMTA reporting requirements: for weekdays, all trips in the AM-peak, midday, PM-peak and night (as specified by SCRTD) are aggregated; for Saturday and Sunday (separately), only all-day totals are used. Individual TP-trip records are read and desired measures are computed and summed until the line, direction or time interval changes, when an output record is written. Passengers boarded is directly available from the input, the average number of passengers aboard between time points is computed from the passengers aboard at this and the previous time points and multiplied by the time and distance between time points to get passenger minutes, passenger miles and bus miles, and a table is referenced to find capacity miles and seat miles. The number of bus trips is incremented each time the first time point of a new trip is read.

2.1.3.2.1.2 Form Daily Run Aggregate File

2.1.3.2.1.2.1 Input

The TP-Trip File.

2.1.3.2.1.2.2 Output

The Daily Run Aggregate File and the Daily Layover Aggregate File. The Run File provides scheduled running time and deviation from scheduled running time between time points, from the start of the trip to the first time point (if the first bus stop on a trip is not a time point), and from the first to last bus stop of a trip. If time-of-day aggregation is specified, mean values are computed for scheduled run time and run time deviation and the standard deviation of the run time is also found. The inclusion of the run time standard deviation allows schedulers to better determine a run time that can be attained by most buses and to determine route segments that may require special service under certain conditions. The Layover File includes time information at the first and last bus stop of each trip.

2.1.3.2.1.2.3 Processing

The basic processing has already been described. The actual run time is computed as the time at this point minus the time at the previous point. The scheduled time at each point is determined by adding the time at each point to the adjusted schedule deviation at that point. The scheduled run time is then computed as the scheduled time at this point minus the scheduled time at the previous point and the run time deviation is computed as the scheduled run time minus the actual run time. If time-of-day aggregation is being performed, the trip vector accumulates the sums of scheduled run times, run time deviations and run times squared as well as the sample size for each time point. When the time interval, direction or line changes, the averages are computed by dividing the sums by the sample sizes, and the standard deviation of the run time is also computed.

A separate file is used for data that can be used to compute layover information, even though this could be considered to be part of running time data, because layover data could not easily be computed at the same time as the other run time data. That is, layover time is not directly available from the data given; it can be computed by first reordering the TR-Trip Records by line, direction and run number (they are presently ordered by line, direction and trip number). Then, for all trips in each run, the last bus stop record containing actual time and adjusted schedule deviation, will be immediately followed by the first bus stop record of the next trip for that bus and the actual and scheduled layover times can be computed from them. However, rather than resort the whole file, skip most of the records and go through many of the same computations as have already been done to form the Run Aggregate File, it was decided to write out the pertinent records as they are being processed for the Run File. A later sort will be much quicker with fewer and shorter records and desired aggregations can be performed independently of those chosen for the Run File.

2.1.3.2.1.3 Form Daily Ridership Aggregate File

This file is later accessed by two report generating programs to construct detailed and summary reports providing ridership data.

2.1.3.2.1.3.1 Input

The TP-Trip File. A table correlating number of seats and bus capacity with bus number and a table giving distance between time points is also needed.

2.1.3.2.1.3.2 Output

The Daily Ride Aggregate File. It summarizes, at each time point, the number of passengers aboard, the number of passengers on and off, the maximum aboard between time points and the bus stop at which the maximum occurred. It also includes the time with standees, in increments of five persons, and trip totals of passenger trips, passengermiles, passenger-miles per passenger trip, load factor, standee minutes and standee minutes per standee.

2.1.3.2.1.3.3 Processing

The first group of data is directly available from the TP-Trip File. The passenger miles are computed in the same manner as for the Section 15 data, and the same principle is used to compute the standee minutes; that is the number of passengers between time points is assumed to be fit by a straight line between the number of passengers at the time points. The time with standees is determined by the amount of time that the passenger straight line is above the horizontal line representing the number of seats. The area formed by that part of the passenger straight line that is above the horizontal seat line gives standee minutes, and this number divided by the time between time points gives the average number of standees. Figure 2-10 shows the relationship of the basic quantities involved, for conditions in which there are standees.

The computation of standee minutes per standee is more complex, as it first requires the determination of the average number of standees during a "standee cycle," which begins whenever there are standees and ends whenever there are no standees remaining. In Figure 2-11, there are three standee cycles. At the end of each cycle the average number of standees is found by dividing the standee minutes in the cycle by the total time with standees.

2.1.3.2.1.4 Form Daily Schedule Deviation Aggregate File

This file stores the sum of the number of schedule deviations in one-minute intervals for all trips in the chosen time intervals. When the output report is later run, the numbers are converted to percentages.

2.1.3.2.1.4.1 Input

The TP-Trip file.

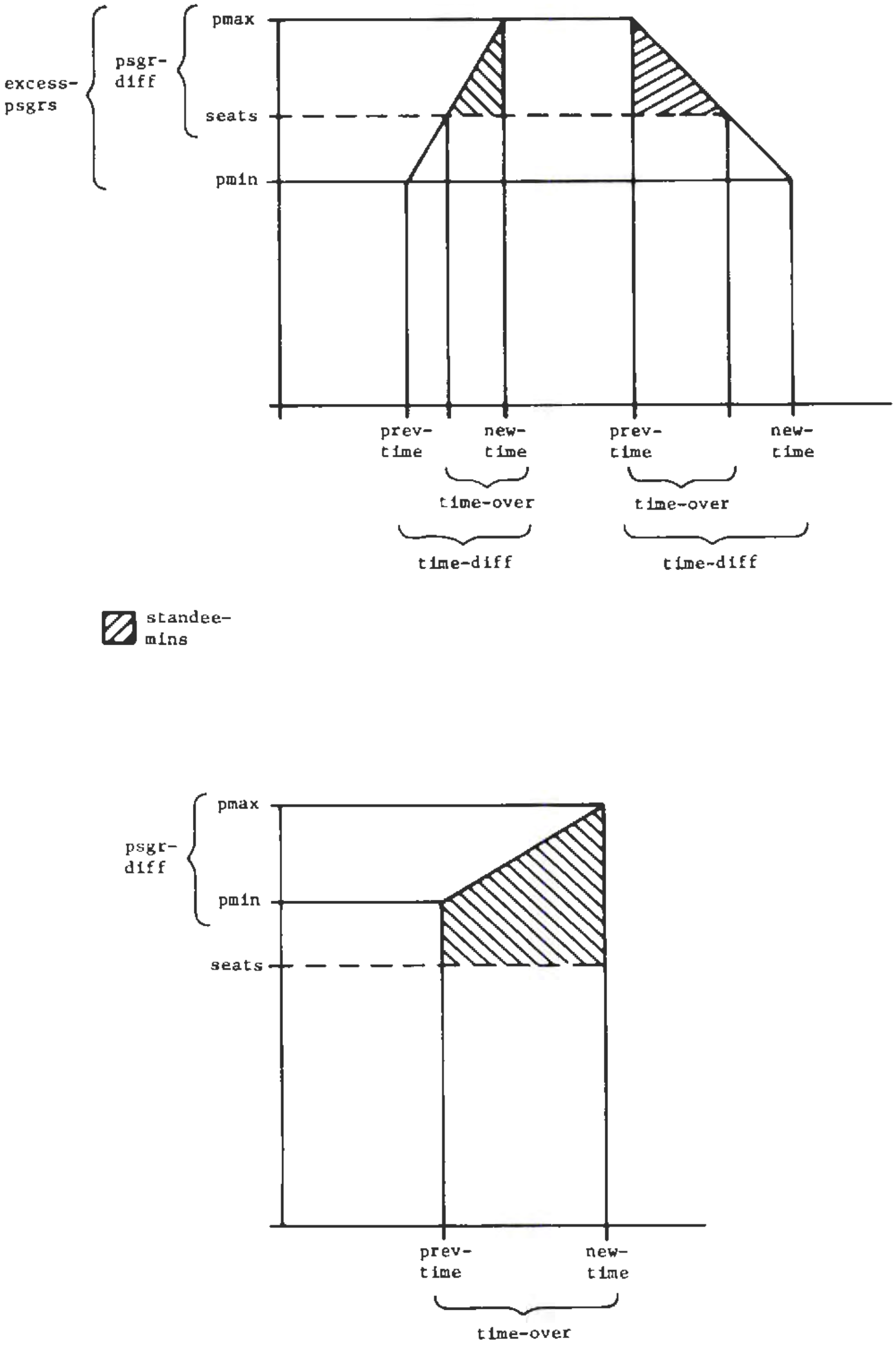


FIGURE 2-10. BASIC STANDEE COMPUTATIONS

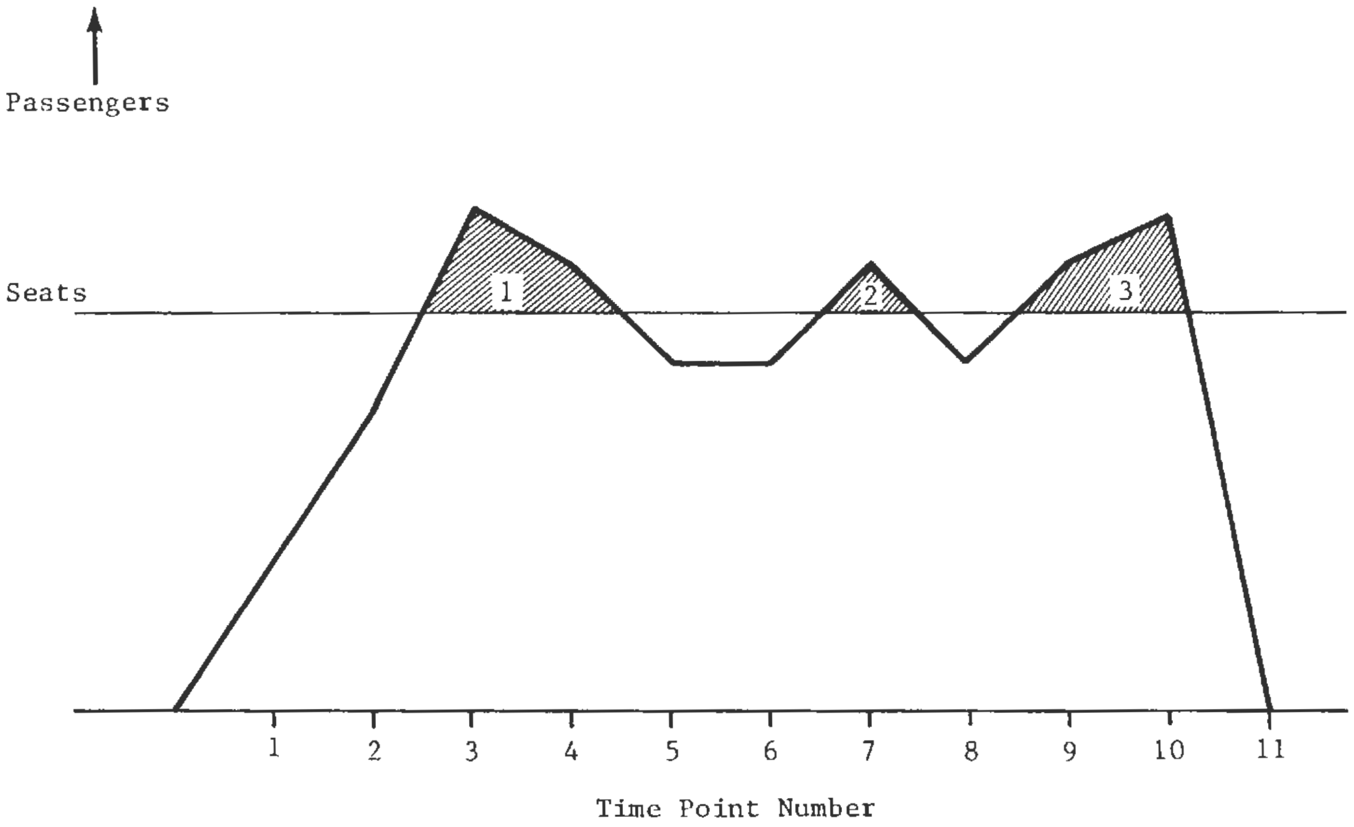


FIGURE 2-11. STANDEE CYCLE ILLUSTRATION

2.1.3.2.1.4.2 Output

The Schedule Deviation Aggregate File including, for each time point, a 22 element vector storing schedule deviation histogram data from minus to plus ten minutes, (including a less than -10 element and a greater than +10 element). The limits can be changed if desired.

2.1.3.2.1.4.3 Processing

Individual TP-trip records are read and the adjusted schedule deviation is examined to determine which histogram element should be incremented. When line, direction or time-interval changes, an output record is written.

2.1.3.2.1.5 Form Daily Overload Aggregate File

This file, originally specified in WP79W00738, has been changed to provide additional data. It now includes, in a similar fashion to the schedule deviation report just described, load factor data in .1 increments.

2.1.3.2.1.5.1 Input

The TP-Trip File. A table to provide the capacity of each bus is aslso required.

2.1.3.2.1.5.2 Output

The Daily Overload Aggregate File, including, for each time point, a 21 element vector storing the number of bus trips in the appropriate load-factor interval. The load factor limits can be changed if desired.

2.1.3.2.1.5.3 Processing

Individual TP-trip records are read and the average load between time points is found from the number of passengers aboard at this and the previous time points. The load factor is computed using the bus capacity from the load-table and the appropriate element of the overload histogram is incremented. When line, direction or time-interval changes, an output record is written.

2.1.3.2.1.6 Form Daily Layover Aggregate File

The Layover File formed during the running of the Run program is sorted by line, direction, and run and then used to determine the scheduled layover time and the layover deviation.

2.1.3.2.1.6.1 Input

The Layover File, consisting of only "first-bus-stop" and "last-bus-stop" records from the TP-trip file.

2.1.3.2.1.6.2 Output

The daily Layover Aggregate File.

2.1.3.2.1.6.3 Processing

The records are read one at a time; as the file only includes "first bus stop" and "last bus stop" records for each trip, each time a new record is different from the previous record (unless the run also changes) the scheduled layover time and layover time deviation are computed from the actual times and the adjusted schedule deviations or time interval at each bus stop. The scheduled layover time and layover deviation are added to previous values until the line, direction or time interval changes, when averages are formed and written out.

2.1.3.2.2 Update Summary Files

The format of the data in the daily output aggregate files is the same as that of the old summary tape data. Consequently, a standard "master file update" can be performed. A record is read from each file; if the old summary record key is less than the aggregate record key, the old summary record is written as a new summary record and another old summary record is read. If the keys are the same, the data from the aggregate record is combined with the old summary record, e.g., updating an average or total, and a new aggregate record is read. If the old summary record is greater than the aggregate record, the aggregate record is written as the new summary record and a new aggregate record is read. The key is composed of the line, direction, day-interval, and time-interval of trip (depending on whether time of day aggregation is being performed). As the end of either file is reached, the key for that file is set to "high-values", the highest collating value for the given Cobol implementation and, from the above description, it can be seen that the rest of the remaining file will be read and written. When both keys are high-values, the job is completed. As each aggregate file contains data from a single day, only those summary records corresponding to that day-interval are changed.

The method in which the data is combined depends upon the manner in which the data is presently aggregated. Most of the data are simply sums, e.g., bus miles, sample size, standee time; therefore the combining process simply consists of adding the daily aggregate to the old summary to form the new summary. There is also one case where only a extreme value, the maximum number of passengers aboard between time points, is saved. In this case, a simple comparison between the aggregate maximum and the old summary maximum is used to determine the new summary maximum. (The associated maximum passenger bus stop is also selected at this time.) Other types of aggregation are required to handle averages, used in the cases of run time, layover time and load factor; and variance, used in the case of run time. Slightly more processing is required here to form the new summary average from the old summary average, the aggregate average, and the sample sizes--the algorithm to accomplish this is given in Section 3.

Figure 2-6 shows the contents of the updated files and the method used to combine the aggregate and summary files. The same update program, using different data definitions, should be able to be used to update all of the files.

2.1.3.3 Form New Perf Data Base Tape

2.1.3.3.1 Input

The Old Perf Data Base Tape and the newly formed TP-Trip File are inputs. The Old Perf Data Base Tape includes a tape header telling the dates included, followed by each included day's TP-Trip File and associated date header.

2.1.3.3.2 Output

The New Perf Data Base Tape. It is the same format as the Old Perf Data Base Tape.

2.1.3.3.3 Processing

The old tape header is read, updated and written to the new tape. All of the following records are read and rewritten to the new tape, a date header is formed and written, and the entire TP-Trip File is written after it.

2.1.3.4 Form Daily Exception Reports

These are of two types, schedule deviation and overloading. Each TP-Trip record is checked to determine if the adjusted schedule deviation indicates that a bus is too far ahead of or behind schedule (separate limits for each), or has a passenger load that is greater than a user-selected load factor. For each out-of-tolerance point, data for the preceding and following buses is also output.

2.1.3.4.1 Form Daily Schedule Deviation Exception Report

2.1.3.4.1.1 Input

The TP-Trip File and user provided early limit and late limit.

2.1.3.4.1.2 Output

The Daily Schedule Deviation Exception Report is printed. The contents of this report include scheduled time and schedule deviation at the time point for the out-of-tolerance trip and for preceding and following trips. An example of an output displaying information of this type, developed by Gould, is shown in the Appendix.

2.1.3.4.1.3 Processing

The TP-Trip File is read one record at a time until three trips have been read. Time point data for each trip are stored in a separate vector. The adjusted schedule deviation at each time point in the second vector is checked against the limits; if a limit is exceeded, the data stored for that time point in each of the three trip vectors are printed. When all of the time points in the second vector have been checked, the contents of the second vector are placed in the first vector, those of the third vector are placed in the second vector, and a new trip is read into the third vector. Modifications in this process are required to account for the first and last trips on a given line and direction.

2.1.3.4.2 Form Daily Overload Exception Report

2.1.3.4.2.1 Input

The TP-Trip File and user-provided load-factor limit. A bus number to bus capacity table is also required.

2.1.3.4.2.2 Output

The Daily Overload Exception Report is printed. The contents of this report include the time-point time, load factor and number of passengers over the load factor limit for the out-of-tolerance trip and for preceding and following trips. Format of this report is similar to the Daily Schedule Deviation Report shown in the Appendix.

2.1.3.4.2.3 Processing

The same general technique used in Schedule Deviation processing, checking the second of three trip vectors for out-of-tolerance condition, is used here. Slightly more processing is required here, however, as different buses may have different capacitites and the load factor is specified as a percent of capacity. Therefore, as each trip is read, the capacity of the given bus is looked up in a table. As each time point record is read the maximum passengers between time points is divided by the capacity to determine the maximum load factor.

The load factor is later compared to the load factor limit and information on that time point as well as the preceding and following buses at that time point are printed out.

2.1.4 Update Trouble Tape

The Trouble Report File generated by the Gould Fortran programs, ordered by time, is first sorted by Trouble Report Number. The resultant Ordered Trouble Report File is appended to the Old Trouble Data Base Tape in a similar manner as was done with the Perf Data Base Tape. Based on routing data included in the trouble reports the Ordered Trouble Report File is also used to produce a daily set of output reports which are sent to specified users.

2.1.4.1 Sort

2.1.4.1.1 Input

The Trouble Report File, produced daily by the Gould Fortran programs.

2.1.4.1.2 Output

The Ordered Trouble Report File.

2.1.4.1.3 Processing

Sort by Trouble Report Number.

2.1.4.2 Form Output Trouble Reports

2.1.4.2.1 Input

The Ordered Trouble Report File.

2.1.4.2.2 Output

Daily Trouble Report Reports. The Appendix shows the format of the trouble reports presented on a Dispatcher's display. The same format will be used for the output report.

2.1.4.2.3 Processing

The Trouble Reports will be read and, based on the distribution codes included, written to separate files. Each file will then be read, the data elements formatted as previously shown, and a separate report (which may contain multiple copies) printed for each distribution code.

2.1.4.3 Form New Trouble Tape

2.1.4.3.1 Input

The Ordered Trouble Report File and the Old Trouble Report File.

2.1.4.3.2 Output

The New Trouble Report File.

2.1.4.3.3 Processing

The Old Trouble Tape Header is read, and the dates included are updated with today's date. The header is then written to the New Trouble Report Tape, followed by all the tape records on the Old Trouble Report Tape. A header is created and written with data corresponding to today's parameters, followed by all records in the Ordered Trouble Report File.

2.2 Form Summary Reports

At any time, summary reports may be run from data on the Perf Summary Tape, as the daily update programs incorporate the latest day's date into the previous summary. Depending on the initial choice of day-of-week and time-of-day aggregation each file's data may include anything from each trip for each day of the week (an unlikely choice) to daily aggregations aggregated over the whole week--in effect, one line of data. In general, something in between will be specified, and Figure 2-12 shows the general format and order of the output for: DOW--weekdays, and weekends; TOD--AM peak, midday, PM-peak and night. The

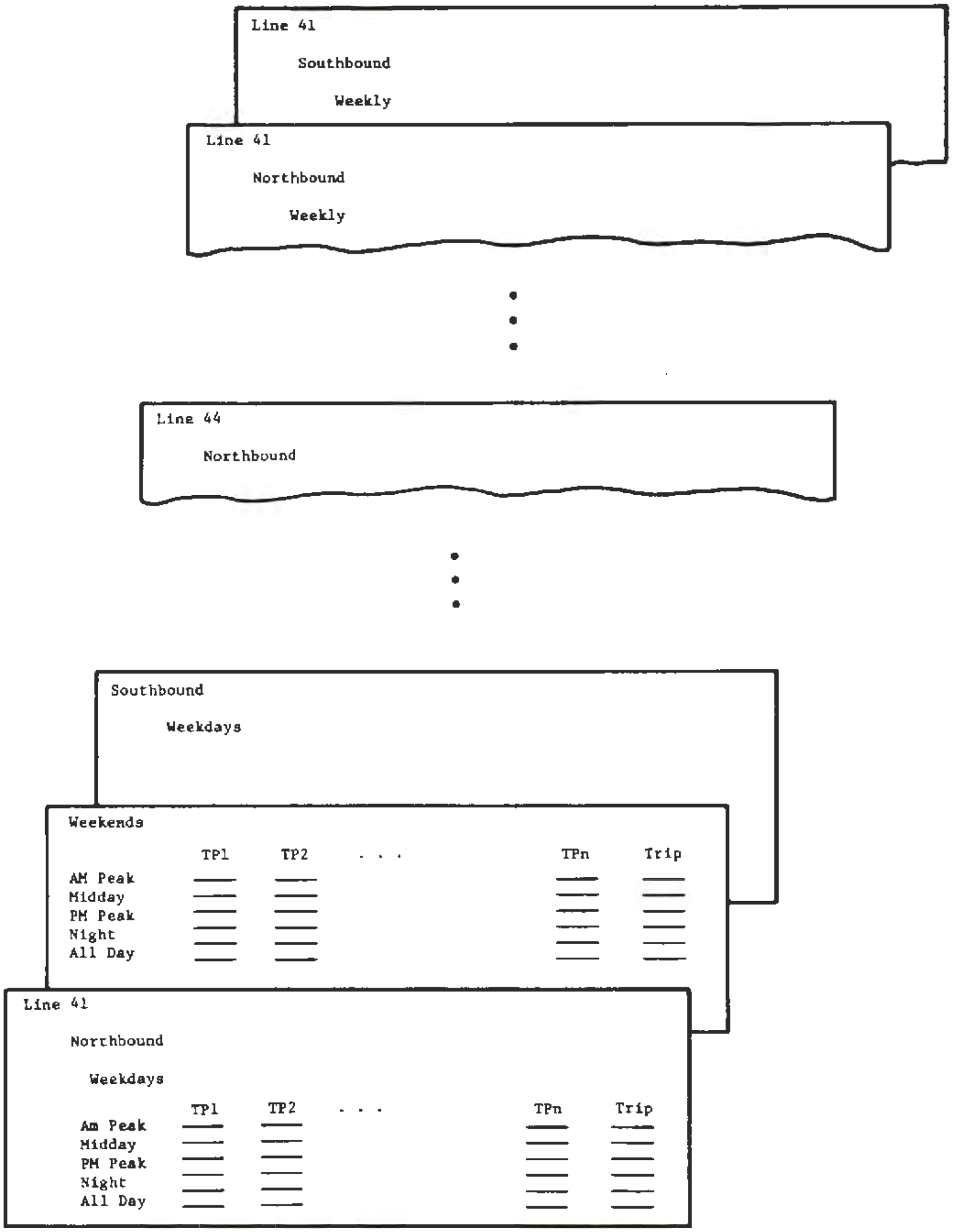


FIGURE 2-12. EXAMPLE OF SEQUENCE OF REPORT DATA

first"page" includes, for line 41, northbound, weekdays, data for each time point by time interval, and a daily total, or average, as the case may be. The next page provides the same data for the same line and direction, but for weekends. The next two pages provide the same order of information for the southbound trips, followed by the other lines, in order.

All this data can be provided merely by formatting the data in the summary files. To get a weekly aggregation for each line, direction time interval, etc. in the same order as shown, the data is sorted again, by line, direction, time-interval or trip, and day-interval. The records are now read and aggregated by time point until the line, direction, trip or time-interval changes, at which time the weekly aggregate is written. A simpler technique could be used, storing data in arrays as they are read and aggregating them properly at the end of the file; however, to accommodate the possibility that 2,000 trips may have to be handled requires inordinately large arrays. In any case, as the files will be substantially condensed by the time this processing needs to be done, the time to sort and process should be minimal.

The Schedule 15 and Ride Trip Reports contain data that can be meaningfully combined further, to form line and system totals. That is, passenger miles or standee minutes during the AM-peak for line 41 and for the entire (AVM-equipped) system are valid measures, whereas average run times or schedule deviations have meaning when referred to a particular route configuration. Therefore, these reports will be further aggregated whereas the others will stop as previously described.

3. PROGRAM LOGIC

The logical procedures required to implement the programs described in Section 2 are detailed in Appendix B. In general, a structured approach is used; that is only sequential statements (including Calls), If . . . Else, For i = n_1 to n_2, Do . . . While or Repeat . . . Until (in the former, the specified condition is tested before the included code is performed; in the latter, the test is performed afterwards) are included. A Cobol-oriented approach is used, although the statements used may not be directly implementable in standard Cobol. One such statement is Assignment By-Name, borrowed from PL/I, to indicate that all of the values of the variables in one data structure that have the same name as variables in the second data structure are replaced by the values of the second set of variables. Too, arrays (tables, in Cobol) are often used to aggregate data for all time points in successive trips until a control variable changes, with separate rows of the array being maintained for time interval aggregation, direction aggregation and line aggregation. As an update to a time interval aggregation is always accompanied by an update to each of the others, this is indicated by replacing the "row" index by *. For example:

```
calc-run-time(*,new-TP) = calc-run-time(*,new-TP) + run-time
```

adds "run-time" to "column" new-TP of all "rows" of calc-run-time. When delineating lines of text, * means the included text is comment. The symbols +, x and / refer to addition, multiplication and division; subtraction is spelled out as "minus" due to the possibility of confusion with the many hypenated variable names. Periods are used to indicate the logical end of compound statements; in more complex instances, "Ends" are also used.

In general, the meaning of the various conventions should be clear when the program descriptions are being read, at least after a few of them have been studied. Also, as much as possible, the logic of the different programs is the same, with only the computations of the pertinent measures being different. This is particularly true with all of the programs of a given type, that is those that form daily-aggregate files are very similar, while the various update and report program descriptions are virtually identical. Even among the different types of programs, a similar program sequence is used, and the determination of control breaks is based on the comparison of "keys" of the new and previous records. A key, defined at the beginning of each program description, is composed of various data that can be used to indicate that the end of a line, direction, trip, time interval, etc. has been reached, requiring some sort of summary and reinitialization processing. When the end of an input file has been reached, high-values are placed in the key, and are used to control the finish of the program.

APPENDIX A - FILE CONTENTS

The contents of the files described in Section 2 and 3 are defined here, in alphabetical order. In general, each file includes a header, a key and data. Some notational conventions are:

=	means is equivalent to
+	means and
[]	means either-or
{}	means iterations of the component enclosed
()	means the enclosed component is optional.

The iteration limits are included below and above the braces, unless they are obvious.

As the format of the data in the Aggregate Files is the same as the (Old and New) Summary Files, only the summary files are given. (The headers are slightly different, as the summary headers also provide data on aggregation limits. Also, instead of writing out the contents of the Report Print Files, examples of the output format are given.)

BASIC-OPERATING-SCHEDULE (AVM Log Tape)

```
Heading = line + schedule + direction +               A
          date-created + date effective +             B
          time-pt = numbers +                         C
          time-pt-names +                             D-F
          notes                                       G

+ Schedule ={Trip + Run + Pull-Out-Time +
            {Pull-Out-Division + Misc. +              H
             Schedule-Times + Nxclue +   }
             Next-Code + Next-Time       }

           + ( {subheading/note/blank} )

           + End-of-Direction                         I

           + End-of-Division
```

NOTE: Ordered by division line, direction, heading, schedule, end-of-division

BUS-STOP-BY-LINE

```
Header  +     {            {      data }}
            lines       bus-stop
```

```
Header  =  file-ID + date + day + weather +
           school

  data  =  bus-stop-record from tracked-bus-file
```

DAY OF WEEK (DOW) TABLE

$$\text{DOW-code} + \text{table-length} + \overset{\text{table-length}}{\underset{1}{\{}} \qquad \overset{7}{\underset{j = 1}{\{}} \text{ indicator } (j) \ \ \} \ \ \}$$

Where:

j corresponds to day-of-week:

1 = Monday, 7 = Sunday

indicator (j) = 0 or 1;

data for all days whose indicator = 1 will be aggregated

$$\text{DOW-code} = \begin{cases} \text{ALL, if want all days individually} \\ \text{INT, if want day-of-week} \end{cases}$$

LAYOVER SUMMARY

Header + $\{\text{ key + data }\}_{\text{trips}}$

header = file-ID + DOW-table + TOD-table + weather code + school-code

key = line + direction + day-int + (time-int) + (trip)

data = sched-layover + layover-dev + sample-size

NOTE: Either time-int or trip-int present, but not both

ORDERED BUS STOP

```
Header  +   {         {   record   }  }
          trips      TPs
```

Header = file-ID + date + day + weather + school

Record = same as tracked-bus record

NOTE: Ordered by line, direction, trip.

OVERLOAD SUMMARY

```
Header  +    {    key + data  }
           trips

Header =  file-ID + load-factor-threshold + DOW-table
          + TOD-table + weather-code + school-code

   key =  line + direction + day-int + time-int

                2.0
  data =  TP  +  { buses-in-load-factor-int }
                 0
```

NOTE: .1 load factor increments--20 intervals
+ 1 interval for >2.0.

PERF DATA BASE

```
                   end-date
File-Header  +       {      date-header +  {        {  record }}
                 start-date                trips    TPs
```

file-header = file-ID + start-date + end-date

date-header = date + day + weather + school

record = same as TP-trip record

RESEARCH TAPE

Identical to Ordered Bus Stop File

RIDE DETAILED REPORT

PASSENGER LOADING REPORT

QUEEN CITY METRO
ROUTE 43
READING ROAD
INBOUND

WEEKDAY AVERAGES FOR
DECEMBER 1978

TRIP SUMMARY

		SEGMENT 1 *				SEGMENT 2				SEGMENT 3				SEGMENT 4				SEGMENT 5				SEGMENT			
		LOCKLAND-ROSELWN				ROSELWN-CALIFORN				CALIF-CLINTON SP				CLINTON SP-LINC				LINCOLN-ROSE RD							
SCHED		RUN	TIME WITH			RUN	TIME WITH			RUN	TIME WITH			RUN	TIME WITH			RUN	TIME WITH			RUN	TIME WITH		
DEPT	BLK	TIME	# STANDEES			TIME	# STANDEES			TIME	# STANDEES			TIME	# STANDEES			TIME	# STANDEES			TIME	# STANDEES		
TIME	NO.	MIN.	1-5	6-10	11>	MIN.	1-5	6-10	11>	MIN.	1-5	6-10	11>	MIN.	1-5	6-10	11>	MIN.	1-5	6-10	11>	MIN.	1-5	6-10	11>
6:33	18	11.6	0.0	0.0	0.0	7.4	0.0	0.0	0.0	7.4	0.0	0.0	0.0	7.7	0.5	0.0	0.0	14.5	0.2	0.0	0.0				
7: 9	14	11.9	0.0	0.0	0.0	8.0	0.0	0.0	0.0	8.2	1.1	0.4	0.0	8.5	0.9	1.3	0.0	14.2	0.3	0.2	0.0				
7:33	4	12.5	0.0	0.0	0.0	8.0	0.0	0.0	0.0	7.6	0.0	0.0	0.0	9.0	0.4	0.3	1.0	14.9	0.0	0.3	0.6				
7:53	17	12.2	0.0	0.0	0.0	9.3	0.0	0.1	0.0	6.9	0.0	0.0	0.7	8.3	0.9	0.2	0.0	13.4	0.7	0.1	0.0				
8:23	11	11.0	0.0	0.0	0.0	8.6	0.1	0.0	0.0	6.6	0.0	0.0	0.0	7.0	0.0	0.0	0.0	12.7	0.0	0.0	0.0				
8:53	13	12.6	0.0	0.0	0.0	8.0	0.0	0.0	0.0	6.9	0.4	0.3	0.0	8.0	0.9	0.6	0.0	12.0	0.9	0.0	0.0				
9:12	1	12.2	0.0	0.0	0.0	8.6	0.0	0.0	0.0	6.4	0.1	0.4	0.0	7.4	0.0	0.0	0.0	12.6	0.0	0.0	0.0				
9:33	4	11.5	0.0	0.0	0.0	8.4	0.0	0.0	0.0	6.3	0.0	0.0	0.0	7.7	0.4	0.0	0.0	13.7	0.3	0.0	0.0				
10:22	11	11.5	0.0	0.0	0.0	7.2	0.0	0.0	0.0	5.7	0.0	0.0	0.0	8.2	0.4	0.0	0.0	13.7	0.1	0.0	0.0				
11: 4	2	11.0	0.0	0.0	0.0	7.9	0.0	0.0	0.0	6.1	0.3	0.0	0.0	7.9	0.3	0.3	0.0	13.2	0.2	0.0	0.0				
12: 7	11	10.0	0.0	0.0	0.0	8.5	0.0	0.0	0.0	6.5	0.0	0.0	0.0	7.0	0.3	0.3	0.0	14.3	0.7	0.6	0.0				
12:49	2	11.5	0.0	0.0	0.0	8.0	0.0	0.0	0.0	6.3	0.0	0.1	0.4	7.7	0.0	0.0	0.6	14.0	0.2	0.1	0.4				
13:18	5	10.9	0.0	0.0	0.0	8.0	0.0	0.0	0.0	6.2	0.4	0.5	0.0	8.3	0.5	0.4	0.2	14.1	0.3	0.0	0.0				
13:59	11	11.0	0.0	0.0	0.0	9.4	0.5	0.4	0.0	6.5	0.4	1.0	0.4	8.9	0.2	0.0	0.0	12.0	0.0	0.0	0.0				
14:22	14	11.1	0.0	0.0	0.0	9.9	0.3	0.1	0.0	7.4	0.7	0.0	0.0	9.3	0.0	0.0	0.0	15.6	0.0	0.0	0.0				
14:40	2	11.0	0.0	0.0	0.0	9.8	0.7	0.0	0.0	8.4	2.0	0.0	0.5	9.6	0.7	0.0	0.0	15.5	0.0	0.0	0.0				
14:50	5	12.0	0.0	0.0	0.0	11.4	1.0	0.2	0.0	7.9	1.1	1.1	2.3	9.9	0.6	0.0	1.2	15.0	0.6	1.0	0.1				
15:29	16	11.9	0.0	0.0	0.0	9.1	0.1	0.1	0.0	7.5	1.1	1.0	0.2	9.0	0.2	0.1	0.0	16.0	0.3	0.0	0.0				
15:43	11	11.2	0.0	0.0	0.0	9.2	0.0	0.0	0.0	7.3	0.4	0.0	0.0	8.5	0.0	0.0	0.0	16.2	0.0	0.0	0.0				
15:55	17	13.3	0.0	0.0	0.0	8.6	0.3	0.0	0.0	7.3	0.2	0.3	0.2	8.3	0.3	0.2	0.0	19.0	0.2	0.0	0.3				
16:19	14	12.7	0.0	0.0	0.0	9.0	0.0	0.0	0.0	7.6	0.1	0.3	0.0	8.5	0.1	0.0	0.0	17.0	0.0	0.0	0.0				
16:31	2	13.5	0.0	0.0	0.0	9.0	0.0	0.0	0.0	8.0	0.0	0.0	0.0	9.1	0.2	0.0	0.0	14.5	0.0	0.0	0.0				
21:16	11	9.0	0.0	0.0	0.0	7.2	0.0	0.0	0.7	6.4	0.0	0.0	0.5	7.0	0.0	0.0	0.7	11.4	0.0	0.0	1.2				
22:46	11	9.9	0.0	1.5	0.3	6.7	0.0	0.0	0.0	5.9	0.0	0.0	0.5	6.4	0.0	0.0	0.6	10.6	0.0	0.0	1.4				
24:16	11	9.3	0.0	0.0	0.1	5.1	0.0	0.0	0.0	6.0	0.0	0.0	0.0	5.7	0.0	0.0	0.0	11.2	0.0	0.0	0.0				

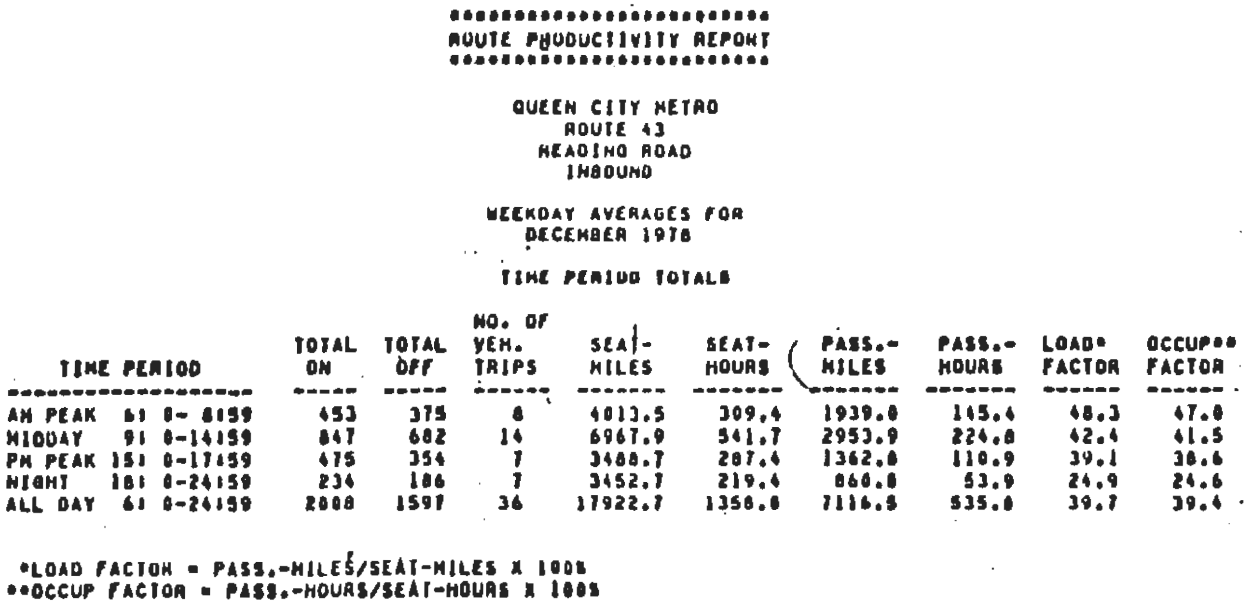
*ONLY THOSE TRIPS WITH STANDEES ARE PRINTED

SOURCE: GM

RIDE DETAILED SUMMARY

```
leader  +     {         {       key  +  data }  }
           trips      TPs


header  =  file-ID + DOW-table + TOD-table +
           weather-code + school-code

   key  =  line + direction + day-int + (time-int)
           + (trip)

  data  =  TP + psgrs-boarding-bet-TPS + psgrs-
           alighting-bet-TPs + max pasgrs-bet-TPs +
           {standee-time}  +  sample-size

             0-5  standees
             5-10 standees
             > 10 standees
```

RIDE-TRIP-REPORT



****

ROUTE PRODUCTIVITY REPORT

****

QUEEN CITY METRO
ROUTE 43
READING ROAD
INBOUND

WEEKDAY AVERAGES FOR
DECEMBER 1978

TIME PERIOD TOTALS

TIME PERIOD	TOTAL ON	TOTAL OFF	NO. OF VEH. TRIPS	SEAT-MILES	SEAT-HOURS	PASS.-MILES	PASS.-HOURS	LOAD* FACTOR	OCCUP** FACTOR
AM PEAK 6: 0- 8:59	453	375	8	4013.5	309.4	1939.0	145.4	48.3	47.0
MIDDAY 9: 0-14:59	847	682	14	6967.9	541.7	2953.9	224.8	42.4	41.5
PM PEAK 15: 0-17:59	475	354	7	3488.7	287.4	1362.8	110.9	39.1	38.6
NIGHT 18: 0-24:59	234	186	7	3452.7	219.4	860.8	53.9	24.9	24.6
ALL DAY 6: 0-24:59	2008	1597	36	17922.7	1358.0	7116.5	535.0	39.7	39.4

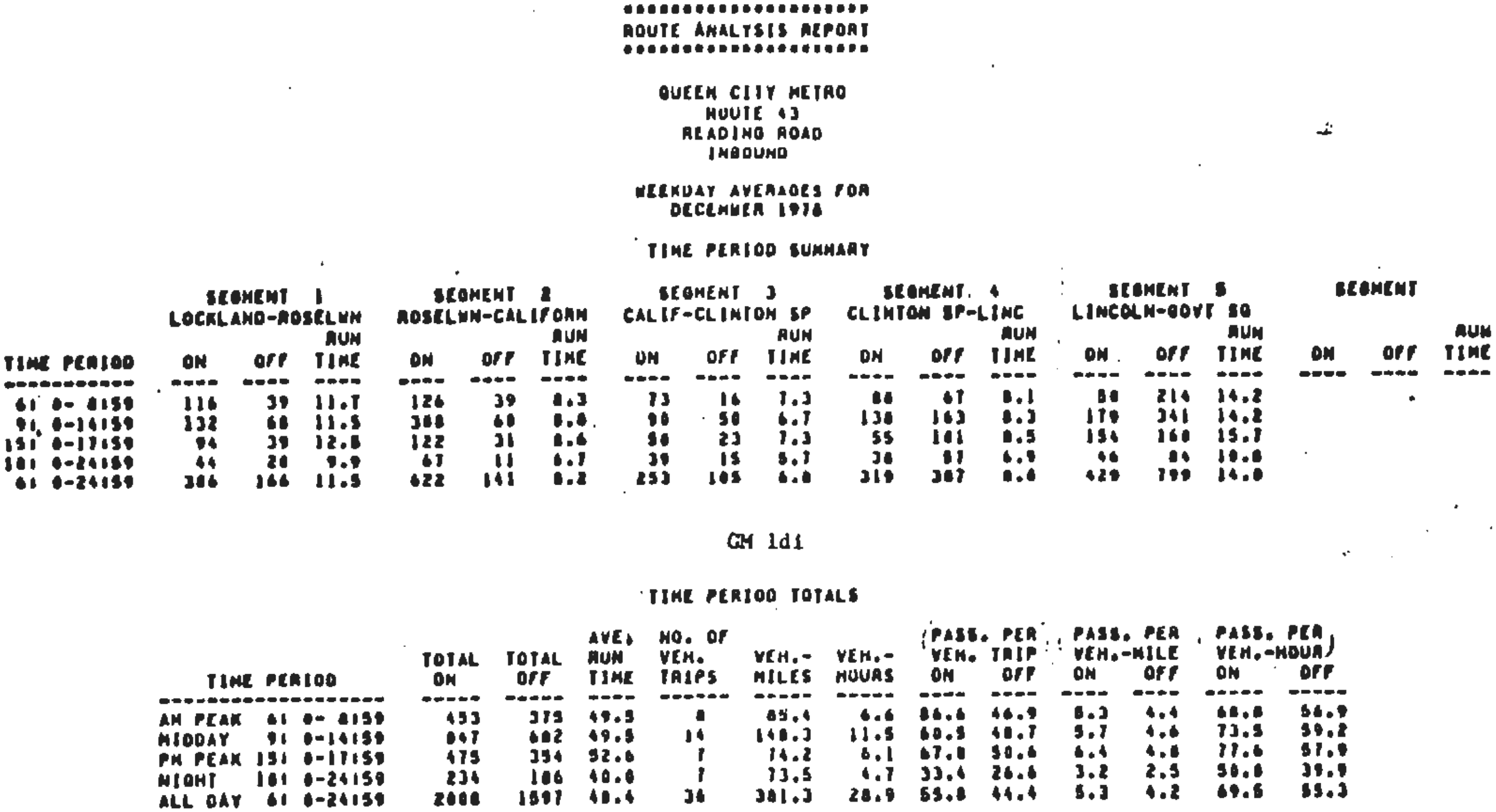
*LOAD FACTOR = PASS.-MILES/SEAT-MILES X 100%
**OCCUP FACTOR = PASS.-HOURS/SEAT-HOURS X 100%

SOURCE: GM

RIDE TRIP SUMMARY

```
header  +   { key  +  data }
           trips

    header  =  file-ID + DOW-table + TOD-table + weather-
               code + school-code

       key  =  line + direction + day-int + (time-int) +
               (trip)

      data  =  psgr-trips + psgr-mis + load-factor +
               standee-mis + sample-size
```

RUN REPORT



****
ROUTE ANALYSIS REPORT
****

QUEEN CITY METRO
ROUTE 43
READING ROAD
INBOUND

WEEKDAY AVERAGES FOR
DECEMBER 1978

TIME PERIOD SUMMARY

	SEGMENT 1			SEGMENT 2			SEGMENT 3			SEGMENT 4			SEGMENT 5			SEGMENT		
	LOCKLAND-ROSELWN			ROSELWN-CALIFORN			CALIF-CLINTON SP			CLINTON SP-LINC			LINCOLN-GOVT SQ					
TIME PERIOD	ON	OFF	RUN TIME	ON	OFF	RUN TIME	ON	OFF	RUN TIME	ON	OFF	RUN TIME	ON	OFF	RUN TIME	ON	OFF	RUN TIME
6: 0- 8:59	116	39	11.7	126	39	8.3	73	16	7.3	88	67	8.1	50	214	14.2			
9: 0-14:59	132	60	11.5	308	60	8.8	90	50	6.7	138	163	8.3	179	341	14.2			
15: 0-17:59	94	39	12.8	122	31	8.6	50	23	7.3	55	101	8.5	154	160	15.7			
18: 0-24:59	44	28	9.9	67	11	6.7	39	15	5.7	38	57	6.9	46	84	10.8			
6: 0-24:59	386	166	11.5	622	141	8.2	253	105	6.8	319	387	8.6	429	799	14.0			

GM 1di

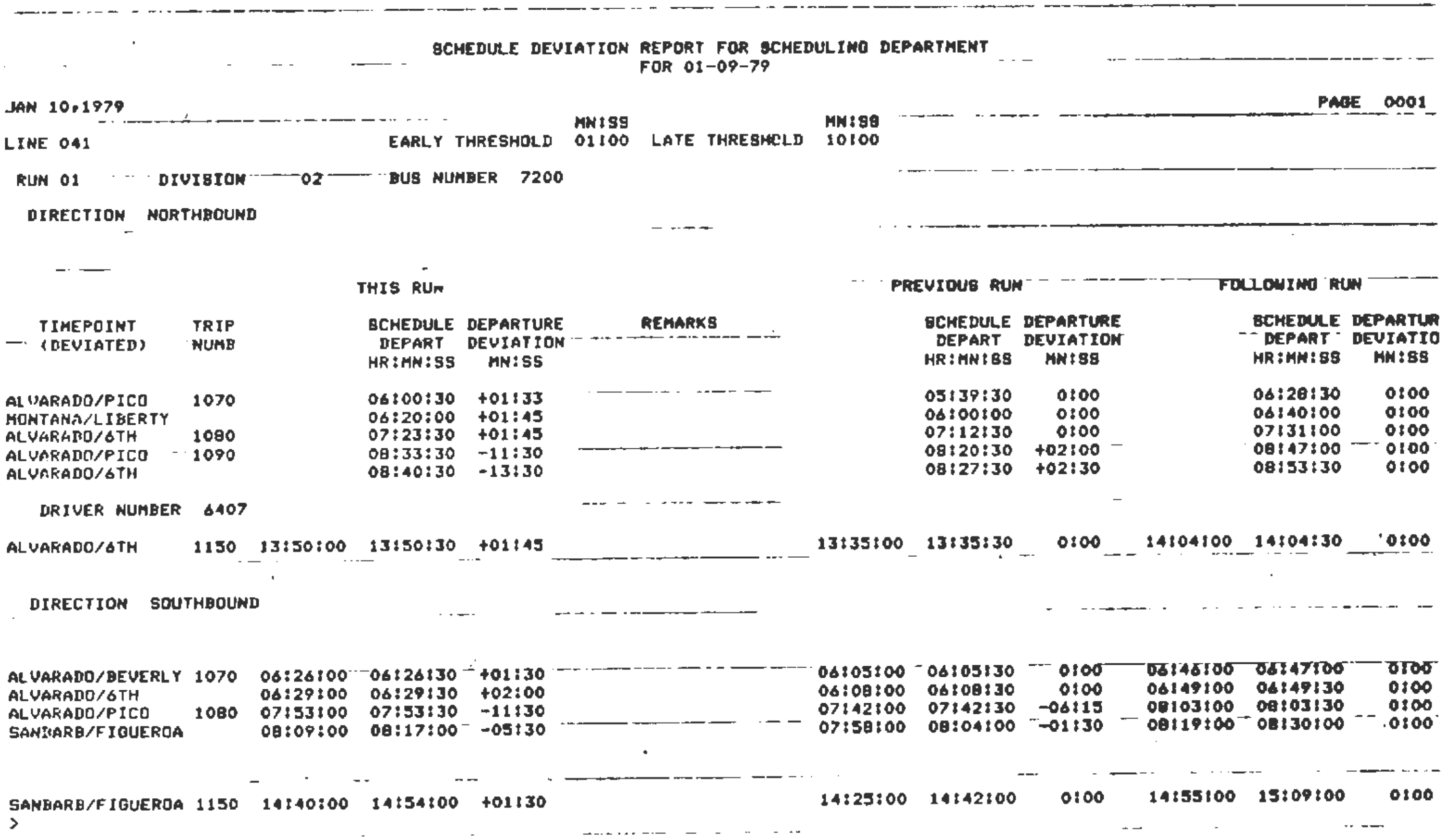
TIME PERIOD TOTALS

TIME PERIOD	TOTAL ON	TOTAL OFF	AVE. RUN TIME	NO. OF VEH. TRIPS	VEH.-MILES	VEH.-HOURS	(PASS. PER VEH. TRIP ON	OFF	PASS. PER VEH.-MILE ON	OFF	PASS. PER VEH.-HOUR) ON	OFF
AM PEAK 6: 0- 8:59	453	375	49.5	8	85.4	6.6	56.6	46.9	5.3	4.4	68.8	56.9
MIDDAY 9: 0-14:59	847	682	49.5	14	148.3	11.5	60.5	48.7	5.7	4.6	73.5	59.2
PM PEAK 15: 0-17:59	475	354	52.6	7	74.2	6.1	67.8	50.6	6.4	4.8	77.6	57.9
NIGHT 18: 0-24:59	234	186	40.0	7	73.5	4.7	33.4	26.6	3.2	2.5	50.0	39.9
ALL DAY 6: 0-24:59	2008	1597	48.4	36	381.3	28.9	55.8	44.4	5.3	4.2	69.6	55.3

SOURCE: GM

RUN SUMMARY

```
header  +   {     {   key  +  data }  }
           trip  TPs

header  =  file-ID + DOW-table + TOD-table +
           weather-code + school-code

   key  =  line + direction + day-int + (time-int)
           +(trip)

  data  =  TP + shed-run-time + run-time-dev +
           (run-time-dev-shed-dev) + sample-size
```

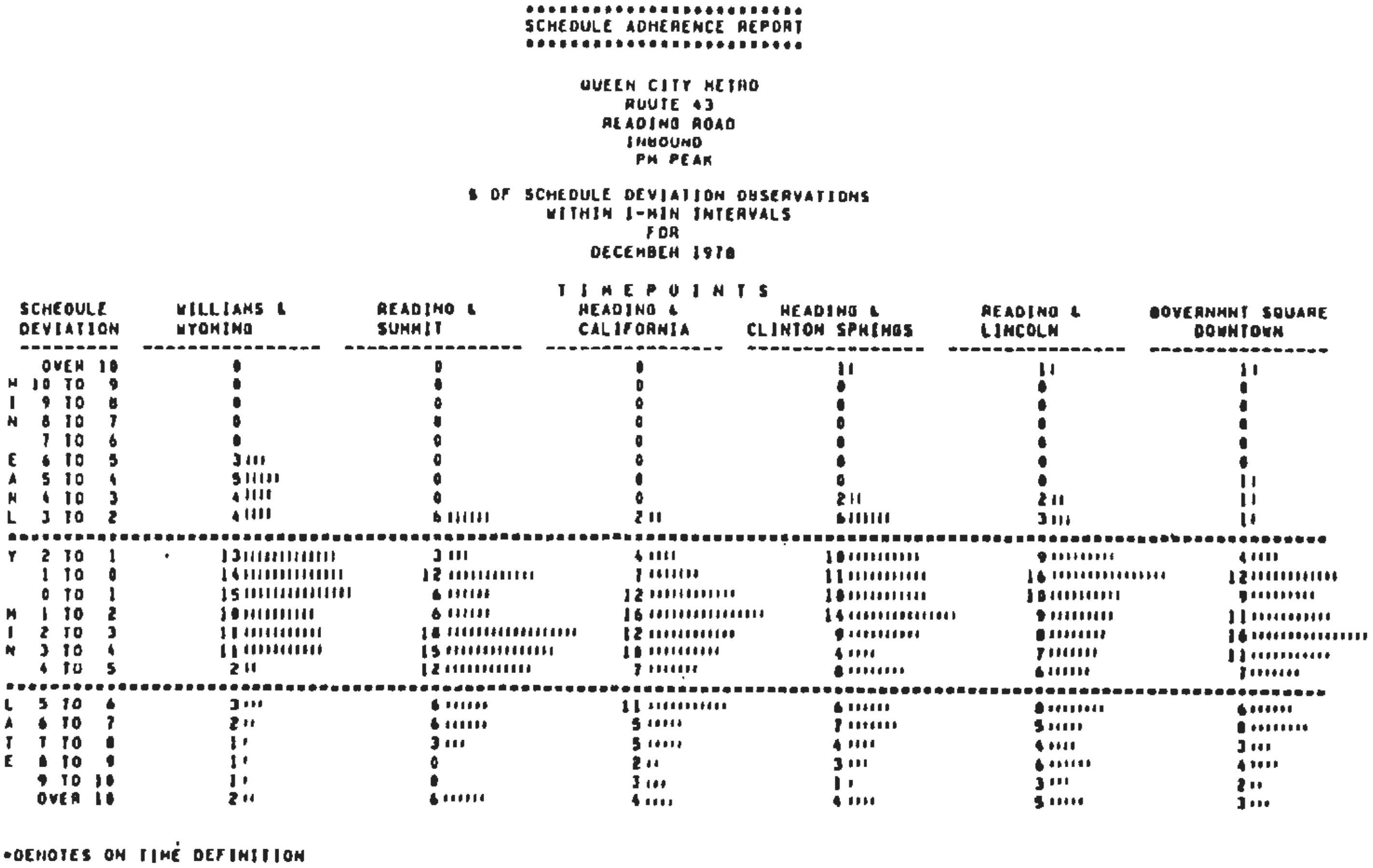


```
                         SCHEDULE DEVIATION REPORT FOR SCHEDULING DEPARTMENT
                                             FOR 01-09-79

JAN 10,1979                                                                                          PAGE  0001
                                         MN:SS                   MN:SS
LINE 041                 EARLY THRESHOLD 01:00  LATE THRESHOLD   10:00

 RUN 01      DIVISION    02     BUS NUMBER  7200

  DIRECTION  NORTHBOUND


                                  THIS RUN                                       PREVIOUS RUN                        FOLLOWING RUN

   TIMEPOINT      TRIP                SCHEDULE DEPARTURE       REMARKS                  SCHEDULE DEPARTURE                 SCHEDULE DEPARTUR
   (DEVIATED)     NUMB                 DEPART  DEVIATION                                 DEPART  DEVIATION                  DEPART  DEVIATIO
                                      HR:MN:SS   MN:SS                                  HR:MN:SS   MN:SS                   HR:MN:SS   MN:SS

ALVARADO/PICO     1070                06:00:30  +01:33                                  05:39:30    0:00                   06:28:30    0:00
MONTANA/LIBERTY                       06:20:00  +01:45                                  06:00:00    0:00                   06:40:00    0:00
ALVARADO/6TH      1080                07:23:30  +01:45                                  07:12:30    0:00                   07:31:00    0:00
ALVARADO/PICO     1090                08:33:30  -11:30                                  08:20:30  +02:00                   08:47:00    0:00
ALVARADO/6TH                          08:40:30  -13:30                                  08:27:30  +02:30                   08:53:30    0:00

   DRIVER NUMBER  6407

ALVARADO/6TH      1150   13:50:00     13:50:30  +01:45                        13:35:00  13:35:30    0:00       14:04:00    14:04:30   '0:00


  DIRECTION  SOUTHBOUND


ALVARADO/BEVERLY 1070    06:26:00     06:26:30  +01:30                        06:05:00  06:05:30    0:00       06:46:00    06:47:00    0:00
ALVARADO/6TH             06:29:00     06:29:30  +02:00                        06:08:00  06:08:30    0:00       06:49:00    06:49:30    0:00
ALVARADO/PICO    1080    07:53:00     07:53:30  -11:30                        07:42:00  07:42:30  -06:15       08:03:00    08:03:30    0:00
SANBARB/FIGUEROA         08:09:00     08:17:00  -05:30                        07:58:00  08:04:00  -01:30       08:19:00    08:30:00    0:00


SANBARB/FIGUEROA 1150    14:40:00     14:54:00  +01:30                        14:25:00  14:42:00    0:00       14:55:00    15:09:00    0:00
>
```

SOURCE: GOULD, Inc.

SCHEDULE-DEVIATION-DAILY REPORT
OVERLOAD-DAILY-SIMILAR

SCHEDULED DEVIATION SUMMARY REPORT
OVERLOAD SUMMARY SIMILAR



```
                                   ************************
                                   SCHEDULE ADHERENCE REPORT
                                   ************************

                                       QUEEN CITY METRO
                                          ROUTE 43
                                        READING ROAD
                                          INBOUND
                                          PM PEAK

                              % OF SCHEDULE DEVIATION OBSERVATIONS
                                   WITHIN 1-MIN INTERVALS
                                            FOR
                                       DECEMBER 1978

                                       T I M E P O I N T S
 SCHEDULE     WILLIAMS &         READING &         READING &          READING &          READING &         GOVERNMNT SQUARE
 DEVIATION    WYOMING            SUMMIT            CALIFORNIA      CLINTON SPRINGS       LINCOLN              DOWNTOWN
 ---------    ---------------   ---------------   ---------------   ---------------   ---------------   ---------------
   OVER 10        0                 0                 0                 1I                1I                1I
 M 10 TO  9       0                 0                 0                 0                 0                 0
 I  9 TO  8       0                 0                 0                 0                 0                 0
 N  8 TO  7       0                 0                 0                 0                 0                 0
    7 TO  6       0                 0                 0                 0                 0                 0
 E  6 TO  5       3III              0                 0                 0                 0                 0
 A  5 TO  4       5IIIII            0                 0                 0                 0                 1I
 R  4 TO  3       4IIII             0                 0                 2II               2II               1I
 L  3 TO  2       4IIII             6IIIIII           2II               6IIIIII           3III              1I
***************************************************************************************************************************
 Y  2 TO  1   *  13IIIIIIIIIIIII    3III              4IIII            10IIIIIIIIII        9IIIIIIIII        4IIII
    1 TO  0      14IIIIIIIIIIIIII  12IIIIIIIIIIII     7IIIIIII         11IIIIIIIIIII      16IIIIIIIIIIIIIIII 12IIIIIIIIIIII
    0 TO  1      15IIIIIIIIIIIIIII  6IIIIII          12IIIIIIIIIIII    10IIIIIIIIII       10IIIIIIIIII        9IIIIIIIII
 M  1 TO  2      10IIIIIIIIII       6IIIIII          16IIIIIIIIIIIIIIII 14IIIIIIIIIIIIII    9IIIIIIIII       11IIIIIIIIIII
 I  2 TO  3      11IIIIIIIIIII     18IIIIIIIIIIIIIIIIII 12IIIIIIIIIIII  9IIIIIIIII         8IIIIIIII        16IIIIIIIIIIIIIIII
 N  3 TO  4      11IIIIIIIIIII     15IIIIIIIIIIIIIII 10IIIIIIIIII       4IIII             7IIIIIII         11IIIIIIIIIII
    4 TO  5       2II              12IIIIIIIIIIII     7IIIIIII          8IIIIIIII         6IIIIII           7IIIIIII
***************************************************************************************************************************
 L  5 TO  6       3III              6IIIIII          11IIIIIIIIIII      6IIIIII           8IIIIIIII         6IIIIII
 A  6 TO  7       2II               6IIIIII           5IIIII            7IIIIIII          5IIIII            8IIIIIIII
 T  7 TO  8       1I                3III              5IIIII            4IIII             4IIII             3III
 E  8 TO  9       1I                0                 2II               3III              6IIIIII           4IIII
    9 TO 10       1I                0                 3III              1I                3III              2II
   OVER 10        2II               6IIIIII           4IIII             4IIII             5IIIII            3III

*DENOTES ON TIME DEFINITION
```

SOURCE: GM

SCHEDULE DEVIATION SUMMARY

$$\text{header} \; + \; \underset{\text{trips}}{\{} \quad \underset{\text{TPs}}{\{} \quad \text{key} \; + \; \text{data} \; \} \; \}$$

$$\text{header} \; = \; \text{file-ID} + \text{DOW-table} + \text{TOD-table}$$

$$\text{key} \; = \; \text{line} + \text{direction} + \text{day-int} + \text{time-int}$$

$$\text{data} \; = \; \text{TP} + \underset{t\,=\,-t_1}{\overset{t\,=\,+t_2}{\{}} \; \text{buses-in-sched-dev-int} \; \} \; + \; \text{sample-size}$$

NOTE: 1 minute sched. dev. intervals

$$\left.\begin{array}{lcl} -t_1 & = & -10 \text{ min.} \\ \\ +t_2 & = & +10 \text{ min.} \end{array}\right\} \quad 22 \text{ intervals}$$

SECTION 15 SUMMARY

header + { key + data }

```
header  =  file-ID + DOW-table + TOD-table

   key  =  line + direction + day-int +
           time-int

  data  =  psgrs-boarded + bus-trip-distance +
           psgr-mi + bus-trip-time + psgr-min +
           capacity-mi + seat-mi + bus-trips
```

NOTE: DOW-table & TOD-table specified by UMTA

Weekdays, Sat., Sun.
AM-peak, Mid-day, PM-peak, Night

SIGNPOST-TABLE-1 (SIGNTB) (AVM Log Tape)

x-coord + y-coord + distance +
east-code + line +
east-chain + west-chain +
north-chain + south-chain

NOTE:

- 1 entry per signpost (917)
- "chain" values tell how many records between this SP and the next one in each direction
- ordered by north-code, then east-code

SIGNPOST-TABLE-2 (SNGTBL) (AVM Log Tape)

number-east-codes-for-this-north-code +
SIGNTB-pointer-to-start-of-north-codes

NOTES:

- o 255 entries-one for each possible north code. Their order defines a given north code.
- o This table used as a key into SIGNTB.

SUMMARY TAPE

```
Header  =  ID + start-date + end-date
        +  (Section-15-Summary)
        +  (Run-Summary)
        +  (Ride-Detailed-Summary)
        +  (Ride-Trip-Summary)
        +  (Sched-Dev-Summary)
        +  (Overload-Summary)
        +  (Layover-Summary)
```

TIME OF DAY (TOD) TABLE

$$\text{TOD-code} + \text{table-length} + \overset{\text{table-length}}{\underset{i = 1}{\{ \text{ end-time } \}}}$$

$$\text{TOD-code} = \begin{cases} \text{ALL, if each trip is to be included separately} \\ \\ \text{INT, if trips are to be aggregated over time intervals} \end{cases}$$

TP-TRIP

```
header  +   {          {         key + data  } }
           trips      TPs
```

```
header = file-ID + date + day + weather + school

  key  = line + direction + trip

  data = run + bus + TP + time + sched-dev +
         psgrs-on-between-TPs + psgrs-off-bet-TPs +
         psgrs-aboard + max-psgrs-bet-TPs +
         max-psgr-bus-stop
```

TRACKED BUS (AVM Log Tape-Type 3 File)

block header + {[bus-stop | bus-assnment/pullout |
dispatcher-action]}

block header = AVM-block-start-time + date + day +
weather + {line + sched + AVM-config}
lines

bus-stop = time + line + run + bus + trip +
direction + origin + destination +
stop + TP + sched-dev + sched-adj +
shed-hywy + hdwy dev + psgr-aboard +
psgrs-boarding + pasgrs-alighting +
total-boardings + total-alighting +
psgrs-on-leader + load-diff + run-time +
psgr-wait + psgr-mi + psgr-trip-time +
psgr-trip-delay + tactical-code +
trouble-report +
SPM-position +1^{3} {tactics} + other-flags

TABLE FORMAT E. TROUBLE REPORT (CS-10)

```
                                   TROUBLE REPORT # 999
                                                                DISP.     NO.     99
VEHICLE:     3108    DIV: 99  CALL (LOCATION: WILSH/WSTRN  STATUS:  E83BL    TIME:    1159A

LINE/RUN:   176-12
OPERATOR ID:    XXXX   CH# XX TRBL  LOCATION: XXXXXXXXXXX TRBL TIME:  XXXXX   CLRD:     XXXXX

ACCIDENT        :     EMERGENCY? :    OPERATIONS? :   MAINTENANCE PROBLEMS?     :

# INJURIES      :     SAS ALARM   :   BLOCKADE     :  AIR COND :  FAREBOX :      ENGINE
TRTD AT SCNE    :     ASSAULT-OPER:   CANCELLATION:   AIR LEAK :  HEADSIGN:       HOT     :
REFERRED AID    :     ASSAULT-PSGR:   DETOUR       :  AIR PRESS:  HEATER  :       POWR    :
TO HOSPITAL     :     ROBBERY     :   LATE         :  AVM DSPLY:  HORN    :       STALL   :
AMBULANCE?      :                     OUT LATE     :  AVM OTHER:  LIGHTS  :      XMISSION
WHICH HOSP?           MISCELLANY? :   OVERLOAD     :  BATTERY  :   HD/TL  :       CLTCH   :
:                                     PADDLEBOARD :   BELLOWS  :   INSIDE :       LEAK    :
                      ALTERATION  :   TRANSFERS    :  BRAKES   :   MARKER :       SHIFT   :
WITNESSES?      :     DISTURBANCE :   ZONE CHECKS :   DEFROSTRS:   MIRROR :       SLIPS   :
PHOTOS?         :     INFORMATION :                   DRECTIONL:   OIL LEAK       RADIO
BUS MOVING?     :     INTOX-PSGR      ILLNESS?     :  DIRTY BUS:   OIL PRES         NO RX :
BUS DAMAGE      :     LOST ARTICLE:                   DOOR-ENTR:   TIRES  :         NO TX :
                      LOST PSGR   :   PASSENGER    :  DOOR-EXIT:   WATER LK         BEEPS :
                      MISSILES    :   OPERATOR     :               WIPERS :
                      VANDALISM   :   REFERRED AID:   OTHER MAINT:

                           .
                           .
                           .
                        Lines 23-40
                           .
                           .
                           .


SUMMARY:                                              REPORTED     BY        TO      TIME
                                                      XXXXXXXX  XXXXXXXX   XXXXX    XXXXX

                           .
                           .
                        Lines 42-52
                           .
                           .
                           .


SERVICE DELAY:     XXXX
NEW BUS NO.     XXXX  LOCATION:      XXXXXXXXXXXX  DIR:    XXXX    TIME:     XXXXX
    REPLACE?    X     TURNBACK?  X  DHB?  X    RELAY?  X
COPIES:         :         :        :       :        :       .       :      :     .
```

SOURCE: COULD, Inc.

TROUBLE REPORT (AVM Log Tape-Type 2 File)

$$\text{block-pointer} + \overset{n}{\underset{1}{\{}} \quad \text{data} \}$$

$$\text{block-pointer} = \overset{n}{\underset{i=1}{\{}} \quad \text{block-number-of-}i^{th}\text{-trouble report} \}$$

$$\text{data} = \text{trouble-report-text}$$

VEHICLE-LOCATION-AND-STATUS (CLST) (AVM Log Tape)

VLST 1 = vehicle + line + run + bus-type

VLST 2 = for-internal-use-only

VLST 3 = $^{6}_{1}\{\text{schedule-deviation-thresholds}\}$ + computed-sched-deviation + $^{1}_{3}\{\text{display-sched-thresholds}\}$

VLST 4 = bus + line + run + veh-ID + bus-type + status

NOTES:

- Random route + 41 + 44 + 89 + 83
- 1 entry per vehicle (220 total)
- vehicle-ID not the same as vehicle-number

APPENDIX B - DETAILED LOGIC

The processes required to form the files defined in Appendix A are detailed here. The numbering system corresponds to that used in Section 2. For example, B.1.3.1, "Form FSTRIP Files" is described in Section 2.1.3.1 and refers to the process with the same name, numbered 1.3.1, shown in the "balloon chart" of Figure 2-5.

B.1 Form AVM Files.

Pertinent logical procedures comprising this major module are described below.

B.1.1 Gould Fortran Programs.

As this module has been separately developed and documented by Gould, details of the logical processes are not given here.

B.1.2 Select and Sort.

```
Do While records exist

   Read tracked-bus Into temp.
   Select (record-type):

      Case(block-header):
         If this is first block-header read Then
            Write bus-stop-header From temp.

      Case(bus-stop-record):
         Write bus-stop-by-line(line) From temp.

   Endselect.

Enddo.

For all lines
   Sort bus-stop-by-line(line) By direction, trip, time.

Concatenate bus-stop-header with
   bus-stop-by-line(all-lines) to form ordered-bus-stop file.
```

B.1.3 Update perf tapes.

The detailed processes comprising this module are described below.

B.1.3.1 Form TP-Trip File

Program Variables:

```
    header
      id
      date
      day
      school
      weather

    new
       key
          line             * other data from   *
          direction        * ordered-bus-stop  *
          trip             *  file not used    *
       data1
          time
          run
          bus
          TP
          adj-sched-dev
          psgrs-aboard
       data2
          stop
          sched-adjust
          psgrs-boarding
          psgrs-alighting
          total-on
          total-off

    prev:  same as above

    out
       key
       data1
       calc
          sched-dev
          psgrs-on
          psgrs-off
          max-psgrs
          max-psgr-stop

    save
       total-on        *  to compute total psgrs  *
       total-off       *    on/off between TPs    *

    max
       max-psgrs
       max-psgrs-stop
```

```
Main:

   Call process-header.
   Call initialize.
   Call process-data.


Process-Header:

   Read ordered-bus-stop Into header.
   id = TP-trip-id.
   Write TP-trip From header.


Initialize:

   Read ordered-bus-stop into new
      at end Call error1.
   prev = new.
   Call trip-begin.
   save = prev By-Name.


Process-Data:

   Do While new-key ¬= high-values

      Do While new-key = prev-key

        prev = new.
        Call stop-update.
        If prev-TP ¬= 0
           Call TP-update.
        Read ordered-bus-stop into new
           at end new-key = high-values.

      Call trip-end.
      Call trip-begin.
   Stop-Update:

      If prev-psgrs-aboard > max-psgrs
         max-psgrs = prev-psgrs-aboard
         max-psgr-stop = prev-stop.


   TP-Update:

      psgrs-on = prev-total-on minus save-total-on.
      psgrs-off = prev-total-off minus save-total-off.
*  Compute psgrs aboard when bus leaves stop  *
      sched-dev = adj-sched-dev minus sched-adj.
      psgrs-aboard = psgrs-aboard + psgrs-boarding minus
                     psgrs-alighting.
      out = prev By-Name.
      calc = max By-Name.
      save = prev By-Name.
      Write TP-trip From out.
      max = 0.


*  First and last bus stop data saved (for trip, layover
     times), even if they're not time points  *

   Trip-Begin:

      out = 0.
      max = 0.
      out = prev By-Name.
      out-TP = begin-code.
      Write TP-trip From out.


   Trip-End:
      out = prev By-Name.
      out-TP = end-code.
      Write TP-trip From out.
      If new-key ¬= high-values
         prev = new
         Read ordered-bus-stop Into new.
```

B.1.3.2 Form New Summary Tape.

The detailed processes comprising this module are described below.

B.1.3.2.1 Information common to Aggregate programs.

Input Variables (from TP-trip file):

```
new
   key
      line
      direction
      trip
   data1
      run
      bus
      TP
   data2
      time
      sched-adj
      adj-sched-dev
   data3
      psgrs-on
      psgrs-off
      psgrs-aboard
      max-psgrs
      max-psgrs-stop

prev:  same as new
```

Header variables:

```
header1                       header2
   id                            id
   begin-date                    date
   end-date                      day
   DOW-code                      weather
   TOD-code                      school
   weather-code
   school-code
```

Process-Header:

```
        If **-summary exists Read summary Into header1.
*  First day of new summary tape  *
        Else Read selection-criteria Into header1.
        Read TP-trip Into header2.
        If weather-code = all or weather = weather-code
           If school-code = all or school = school-code
              header2-id = **-aggregate-id
              Write **-aggregate From header2
              Return.                 *  Continue processing   *

        Else terminate processing.    * Selection criteria not *
                                      *  satisfied by today's  *
                                      *  conditions            *
```

```
     Time-Int-Table(new-data2,new-key):

*  Determine which time interval a given trip start time is in;
     the trip-table gives a "phantom start time" for each bus
     that does not start its trip at the beginning of the line  *

      time-check = trip-table(new-key).

*  If new-trip is in table Then
      time-check = phantom-start-time.
   Else time-check = 0.                    *

     If time-check = 0 Then
        sched-time = new-time + sched-dev
        time-int = TOD-table(sched-time).
     Else time-int = TOD-table(time-check).


 *  TOD-table chooses the proper TOD-interval based on the
       originally-chosen aggregation intervals  *
```

B.1.3.2.1.1 Form Daily-Section-15-Aggregate File

```
   Program Variables:

      out
         key
            line
            direction
            day-int
            time-int
         data
            psgrs-boarded
            bus-miles
            psgr-miles
            bus-mins
            psgr-mins
            capacity-miles
            seat-miles
            bus-trips    * sample size *

      calc(4)
         data (same as above)

*     calc(i,j)                                                *
*        i = 1 - time interval or trip aggregation             *
*            2 - direction aggregation                         *
*            3 - line aggregation                              *
*            4 - system aggregation                            *


   Main:

      Call process-header.
      Call initialize.
      Call process-data.


   Initialize:

      calc = 0.
      Read TP-trip Into prev At end Call err1.
      Read TP-trip Into new At end Call err2.
      Call aggregate-begin.
   Process-Data:

      Do While new-key ¬= high-values
         Do While new-key = prev-key

            Call compute.
            prev = new.
            Read TP-trip Into new At end new-key = high values.

         Call trip-end.
      Call system-control-break.
      End.


   Aggregate-Begin:

*  Initialize out-key data      *
      out = 0.
      out-day-int = day-table(header-day).
      out-time-int = time-int-table(new-data2,new-key).
      out-line = new-line.
      out-direction = new-direction.


   Compute:

      calc-psgrs-boarded(*) = calc-psgrs-boarded(*) + psgrs-on(new-TP).
      avg-psgrs = (psgrs-aboard(prev-TP) + psgrs-aboard(new-TP)) / 2.
      miles = distance-table(prev-TP,new-TP).
      mins = new-time minus prev-time.
      calc-bus-mins(*) = calc-bus-mins(*) + mins.
      calc-bus-miles(*) = calc-bus-miles(*) + miles.
      calc-psgr-miles(*) = calc-psgr-miles(*) + (avg-psgrs x miles).
      calc-psgr-mins(*) = calc-psgr-mins(*) + (avg-psgrs x mins).
      seats = seat-table(bus).
      capacity = seats.
      calc-capacity-miles(*) = calc-capacity-miles(*) + (capacity x miles).
      calc-seat-miles(*) = calc-seat-miles(*) + (seats x miles).
   Trip-End:

*  When new record is from different trip, line or direction,   *
*       perform necessary summaries                              *
      calc-bus-trips(*) = calc-bus-trips(*) + 1.
      time-int = time-int-table(new-data2,new-key).
      If new-line ¬= prev-line
         Call line-control-break.
      Else If new-direction ¬= prev-direction
              Call direction-control-break.
           Else If time-int ¬= out-time-int
                 Call time-control-break.
*   Otherwise, perform no additional processing   *

*   Have first record in new trip--read another   *
      If new-key ¬= high-values
         Call aggregate-begin
         prev = new
         Read TP-trip Into new At end Call err3.


   Line-Control-Break:
      Call direction-control-break.
      out-direction = line-agg-code.
      Call output(3).

   Direction-Control-Break:
      Call time-control-break.
      out-time-int = dir-agg-time-code.
      Call output(2).


   Time-Control-Break:
      Call output(1).


   System-Control-Break:
      out-line = system-agg-code.
      Call output(4).


   Output(i):

*   Write output   *
      out = calc(i) By-Name.
      Write Section-15-aggregate From out.
      calc(i) = 0.
```

B.1.3.2.1.2 Form Daily-Run-Aggregate File

```
   Program Variables:

      out                               *  layover output format  *
         key                            *    same as input        *
            line
            direction
            day-int
            (time-int)
            (trip)
         data
            TP
            sched-run-time
            run-time-dev
            adj-run-time-dev
            run-time-dev-var
            sample-size


      calc(2,trip-agg-code)
         data (same as above)

*     calc(i,j)                                                *
*        i = 1 - time interval or trip aggregation             *
*            2 - direction aggregation                         *

      save
         time
         sched-dev
         adj-sched-dev


   Main:

      Call process-header.
      Call initialize.
      Call process-data.


   Initialize:

      calc = 0.
      Read TP-trip Into prev At end Call err1.
      Read TP-trip Into new At end Call err2.
      Call aggregate-begin.
   Process-Data:

      Do While new-key ¬= high-values
         Do While new-key = prev-key

            Call compute.
            prev = new.
            Read TP-trip Into new At end new-key = high values.

         Call trip-end.
      End.


   Aggregate-Begin:

*  Initialize out-key data      *
      out = 0.
      out-day-int = day-table(header-day).
      If TOD-code ¬= all
         out-time-int = time-int-table(new-data2,new-key).
      Else out-trip = new-trip.
      out-line = new-line.
      out-direction = new-direction.

   Compute:

*   Save 'first bus stop' data from trip totals    *
      If prev-TP = begin-code
         save = prev By-Name
         Write layover from prev.

*   Check for 'last bus stop'   *
      If new-TP = end-code
         Write layover From new
         Return.

*   Compute times and add to previously saved ones   *
      run-time = new-time minus prev-time.
      sched-time =  (new-time + new-sched-dev) minus
             (prev-time + prev-sched-dev).
      adj-sched-time = (new-time + new-adj-sched-dev) minus
             (prev-time + prev-adj-sched-dev).
      dev = sched-time minus run-time.
      adj-dev = adj-sched-time minus run-time.
      calc-sched-run-time(*,new-TP) = calc-sched-run-time(*,new-TP) +
             sched-time.
      calc-run-time-dev(*,new-TP) = calc-run-time-dev(*,new-TP) + dev.
      calc-adj-run-time-dev(*,new-TP) = calc-adj-run-time-dev(*,new-TP) +
             adj-dev.
      calc-run-time-dev-var(*,new-TP) = calc-run-time-dev-var(*,new-TP) +
               dev**2.
      calc-sample-size(*,new-TP) = calc-sample-size(*,new-TP) + 1.
   Trip-End:

*  When new record is from different trip, line or direction,    *
*       perform necessary summaries                               *

      run-time = prev-time minus save-time.
      sched-time = (prev-time + prev-sched-dev) minus
             (save-time + save-sched-dev).
      adj-sched-time = (prev-time + prev-adj-sched-dev) minus
             (save-time + save-adj-sched-dev).
      dev = sched-time minus run-time.
      adj-dev = adj-sched-time minus run-time.
      calc-sched-run-time(*,trip-agg-code) =
           calc-sched-run-time(*,trip-agg-code) + sched-time.
      calc-run-time-dev(*,trip-agg-code) =
           calc-run-time-dev(*,trip-agg-code) + dev.
      calc-adj-run-time-dev(*,trip-agg-code) =
           calc-adj-run-time-dev(*,trip-agg-code) + adj-dev.
      calc-run-time-dev-var(*,trip-agg-code) =
           calc-run-time-dev-var(*,trip-agg-code) + dev**2.
      calc-sample-size(*,trip-agg-code) =
           calc-sample-size(*,trip-agg-code) + 1.
      time-int = time-int-table(new-data2,new-key).

      If new-line ¬ = prev-line or new-direction ¬= prev-direction
         Call direction-control-break.
      Else If time-int ¬= out-time-int or
            TOD-code = all
            Call time-control-break.
*   Otherwise, perform no further processing   *

*   Have first record in new trip--read another   *
      If new-key ¬= high-values
         prev = new
         Call aggregate-begin
         Read TP-trip Into new At end Call err3.

  Direction-Control-Break:
     Call time-control-break.
     If TOD-code = all out-trip = dir-agg-trip-code.
     Else out-time-int = dir-agg-time-code.
     Call output(2).

  Time-Control-Break:
     Call output(1).
   Output(i):

*    Form averages and output   *
       For j = 1 to trip-agg-code
          If calc-sample-size(i,j) > 0
             calc-sched-run-time(i,j) = calc-sched-run-time(i,j) /
                calc-sample-size(i,j)
             calc-run-time-dev-var(i,j) = (calc-run-time-dev-var(i,j)
                minus calc-run-time-dev(i,j)**2) / calc-sample-size(i,j)
             calc-run-time-dev(i,j) = calc-run-time-dev(i,j) /
                calc-sample-size(i,j)
             out = calc(i,j) By-Name
             Write run-aggregate from out.
       calc(i,*) = 0.
```

B.1.3.2.1.3 Form Daily-Ride-Aggregate Files

```
   Program Variables:

      out1                 out            out2
         key                  key            key
            line                             data
            direction                           psgr-trips
            day-int                             psgr-miles
            (time-int)                          load-factor
            (trip)                              standee-mins
                                                st-min-per-st
         data                                   bus-trips  * sample-size*
            TP
            psgrs-on
            psgrs-off
            max-psgrs
            max-psgr-stop
            standees
            standee-time(3)
            mins-wi-standees
            sample-size

      calc1(2,trip-agg-code)                     calc2(4)
         data (same as above)                       data (same as above)
                                                       + standees

                                                 save
                                                    load

*     calc(i,j)                                                *
*        i = 1 - time interval or trip aggregation             *
*            2 - direction aggregation                         *
*            3 - line aggregation                              *
*            4 - system aggregation                            *


   Main:

      Call process-header.
      Call initialize.
      Call process-data.


   Initialize:

      calc1, calc2, save = 0.
      Read TP-trip Into prev At end Call err1.
      Read TP-trip Into new At end Call err2.
      Call aggregate-begin.
   Process-Data:

      Do While new-key ¬= high-values
         Do While new-key = prev-key

            Call compute.
            prev = new.
            Read TP-trip Into new At end new-key = high values.

         Call trip-end.
      Call system-control-break.
      End.


   Aggregate-Begin:

      seats = seat-table(bus).
      st-min2, min2 = 0.
*  Initialize out-key data      *
      out, out1, out2 = 0.
      out-day-int = day-table(header-day).
      If TCD-code ¬= all
         out-time-int = time-int-table(new-data2,new-key).
      Else out-trip = new-trip.
      out-line = new-line.
      out-direction = new-direction.
   Compute:

*   Detailed Ridership Computations   *
      calc1-psgrs-on(*,new-TP) = calc1-psgrs-on(*,new-TP) +
                                 new-psgrs-on.
      calc1-psgrs-off(*,new-TP) = calc1-psgrs-off(*,new-TP) +
                                  new-psgrs-off.
      If new-max-psgrs > calc1-max-psgrs(*,new-TP)
         calc1-max-psgrs(*,new-TP) = new-max-psgrs
         calc1-max-stop(*,new-TP) = new-max-stop.
      calc1-sample-size(*,new-TP) = calc1-sample-size(*,new-TP) + 1.

*   Trip Ridership Computations (others at end)   *
      miles = distance-table(prev-TP,new-TP).
      mins = new-time minus prev-time.
      avg-psgrs = (prev-psgrs-aboard + new-psgrs-aboard) / 2.
      save-load(*) = save-load(*) + avg-psgrs.
      calc2-psgr-miles(*) = calc2-psgr-miles(*)  +  (avg-psgrs x miles).
      calc2-psgr-mins(*) = calc2-psgr-mins(*) +  (avg-psgrs x mins).
      calc2-psgr-trips(*) = calc2-psgr-trips(*) +  psgrs-on.

*   Compute Standee Data (trip-agg-code saves trip sums)   *
      If (prev-psgrs-aboard or new-psgrs-aboard) > seats
         Call compute-standees
         calc1-standees(*,new-TP) = calc1-standees(*,new-TP) + standees
*  Trip standees calculated at control break  *

         calc1-mins-wi-standees(*,new-TP) = calc1-mins-wi-standees(*,new-TP)
                                            mins-wi-standees
         calc1-mins-wi-standees(*,trip-agg-code) =
              calc1-mins-wi-standees(*,trip-agg-code) + mins-wi-standees
         calc2-standee-mins(*) = calc2-standee-mins(*) + mins-wi-standees

*  Accumulate passenger-minutes-per-standee data  *

         st-min2 = st-min2 + standee-mins
         min2 = min2 + mins-wi-standees
         If new-psgrs-aboard < seats
            calc2-standees(*) = calc2-standees(*) + (st-min2 / min2)
            st-min2, min2 =0
         Endif

*   Separate standee-mins according to number of standees   *
         If standee-mins < 5
            calc1-standee-time(*,new-TP,1)= calc1-standee-time(*,new-TP,1) +
                                             standee-mins
            calc1-standee-time(*,trip-agg-code,1) =
                 calc1-standee-time(*,trip-agg-code,1) + standee-mins.
         Else If standees > 10
                 calc1-standee-time(*,new-TP,3) =
                    calc1-standee-time(*,new-TP,3) + standee-mins
                 calc1-standee-time(*,trip-agg-code,3) =
                    calc1-standee-time(*,trip-agg-code,3) + standee-mins.
              Else calc1-standee-time(*,new-TP,2) =
                      calc1-standee-time(*,new-TP,2) + standee-mins
                   calc1-standee-time(*,trip-agg-code,2) =
                      calc1-standee-time(*,trip-agg-code,2) + standee-mins.
      Endif.
```

```
  Compute-Standees:

     pmin = min(prev-psgrs-aboard,new-psgrs-aboard).
     pmax = max(prev-psgrs-aboard,new-psgrs-aboard).
     psgr-diff = pmax minus pmin.
     time-diff = new-time minus prev-time.

     If (pmin and pmax) > seats
        time-over = time-diff.
        standee-mins = ((psgr-diff/2.) + (pmin minus seats)) x
                       time-over.
     Else excess-psgrs = pmax minus seats
          ratio-over = excess-psgrs / psgr-diff
          time-over = time-diff x ratio-over
          standee-mins = (excess-psgrs x time-over) / 2.

     If time-diff > 0
        standees = standee-mins / time-diff.
     Else standees = 0.

     mins-wi-standees = time-over.


   Trip-End:

*  When new record is from different trip, line or direction,
        perform necessary summaries                                 *
      calc2-bus-trips(*) = calc2-bus-trips(*) + 1.
      time-int = time-int-table(new-data2,new-key).
      If new-line ¬= prev-line
         Call line-control-break.
      Else If new-direction ¬= prev-direction
              Call direction-control-break.
           Else If time-int ¬= out-time-int or
                 time-code = all
                 Call time-control-break.
*   Otherwise, perform no further processing   *

*   Have first record in new trip--read another   *
      If new-key ¬= high-values
         Call aggregate-begin
         prev = new
         Read TP-trip Into new At end Call err3.


   Line-Control-Break:
      Call direction-control-break.
      out-direction = line-agg-code.
      Call output(3).
   Direction-Control-Break:
      Call time-control-break.
      If TOD-code = all out-trip = dir-agg-trip-code.
      Else out-time-int = dir-agg-time-code.
      Call output(2).


   Time-Control-Break:
      Call output(1).
      save = 0.


   System-Control-Bread:
      out-line = system-agg-code.
      Call output(4).


   Output(i):

*   Write out data   *

*    Output-2 data  *

      avg-load = save-load(i) / calc2-sample-size(i).
      calc2-load-factor(i) = avg-load / seats.
      If calc2-standees(i) > 0 Then
         calc2-st-min-per-st(i) = calc2-standee-mins(i) / calc2-standees(i).
      out2 = calc2 By-Name.
      calc1-standees(i,trip-agg-code) = calc2-standees(i).
      Write ride-trip-aggregate from out2.
      calc2(i) = 0.

*    Output-1 data  *

      If i <= 2     * for ride-detailed file, only output for  *
                    *  time & direction control breaks (line   *
                    *  and system totals meaningless)          *
         For j = 1 to trip-agg-code
            If calc1-sample-size(i,j) > 0 Then
               out1 = calc1(i,j) By-Name
               Write ride-detail-aggregate from out.
         calc1(i,*) = 0.
```

B.1.3.2.1.4 Form Daily-Sched-Dev-Aggregate File

```
   Program Variables:

      out
         key
            line
            direction
            day-int
            time-int
         data
            TP
            hist(22)               * calc-hist(i,j,k) *
            sample-size

      calc(2,trip-agg-code)
         data (same as above)      * note: there is a  *
                                   *  calc-hist(i,j,k) *

*     calc(i,j)                                          *
*        i = 1 - time interval or trip aggregation       *
*            2 - direction aggregation                   *
*        j = TP                                          *
*        k = histogram interval                          *


   Main:

      Call process-header.
      Call initialize.
      Call process-data.

   Initialize:

      calc = 0.
      Read TP-trip Into prev At end Call err1.
      Read TP-trip Into new At end Call err2.
      Call aggregate-begin.

   Process-Data:

      Do While new-key ¬= high-values
         Do While new-key = prev-key

            Call compute.
            prev = new.
            Read TP-trip Into new At end new-key = high values.

         Call trip-end.
      End.
   Aggregate-Begin:

*  Initialize out-key data      *
      out = 0.
      out-day-int = day-table(header-day).
      out-time-int = time-int-table(new-data2,new-key).
      out-line = new-line.
      out-direction = new-direction.

   Compute:

      calc-sample-size(*,new-TP) = calc-sample-size(*,new-TP) + 1.
*   Find histogram interval for sched-dev   *
      If new-adj-sched-dev < -10
         calc-hist(*,new-TP,1) = calc-hist(*,new-TP,1) + 1.

      Else If new-adj-sched-dev > +10
              calc-hist(*,new-TP,22) = calc-hist(*,new-TP,22) + 1.

           Else index = integer(new-adj-sched-dev) + 12
                calc-hist(*,new-TP,index) = calc-hist(*,new-TP,index) + 1.


   Trip-End:
*  When new record is from different trip, line or direction,  *
*       perform necessary summaries                             *

      time-int = time-int-table(new-data2,new-key).
      If new-line ¬= prev-line or new-direction ¬= prev-direction
         Call direction-control-break.
      Else If time-int ¬= out-time-int
              Call time-control-break.
*   Otherwise, perform no processing   *

*   Have first record in new trip--read another  *
      If new-key ¬= high-values
         Call aggregate-begin
         prev = new
         Read TP-trip Into new At end Call err3.


   Direction-Control-Break:
      Call time-control-break.
      out-time-int = dir-agg-time-code.
      Call output(2).


   Time-Control-Break:
      Call output(1).
```

```
Output(i):

   For j = 1 to trip-agg-code
      If calc-sample-size(i,j) > 0
         calc-hist(i,j,*) = calc-hist(i,j,*) / calc-sample-size
         out = calc(i,j) By-Name
         Write sched-dev-aggregate from out.
   calc(i,*) = 0.
```

B.1.3.2.1.5 Form Daily-Overload-Aggregate File

```
   Program Variables:

      out
         key
            line
            direction
            day-int
            time-int
         data
            TP
            hist(10)
            sample-size

      calc(2,trip-agg-code)
         data (same as above)

*     calc(i,j)                                                *
*        i = 1 - time interval or trip aggregation             *
*            2 - direction aggregation                         *


   Main:

      Call process-header.
      Call initialize.
      Call process-data.


   Initialize:

      calc = 0.
      Read TP-trip Into prev At end Call err1.
      Read TP-trip Into new At end Call err2.
      Call aggregate-begin.


   Process-Data:

      Do While new-key ¬= high-values
         Do While new-key = prev-key

            Call compute.
            prev = new.
            Read TP-trip Into new At end new-key = high values.

         Call trip-end.
      End.
   Aggregate-Begin:

*  Initialize out-key data      *
      out = 0.
      capacity = seat-table(bus).
      If time-code = all
         out-time-int = time-int-table(new-data2,new-key).
      Else out-trip = new-trip.
      out-day = day-table(header-day).
      out-line = new-line.
      out-direction = new-direction.


   Compute:

*   Find load factor between time points    *
      avg-load = (prev-psgrs-aboard +
                          new-psgrs-aboard) / 2.
      load-factor = avg-load / capacity.
      calc-sample-size(*,new-TP) = calc-sample-size(*,new-TP) + 1.

*   Find histogram interval  *
      index = load-factor-table(load-factor).
      calc-hist(*,new-TP,index) = calc-hist(*,new-TP,index) + 1.


   Trip-End:

*  When new record is from different trip, line or direction,  *
*        output averages                                        *
      If new-line ¬= prev-line or new-direction ¬= prev-direction
         Call direction-control-break.
      Else Call time-control-break.

*   Have first record in new trip--read another    *
      If new-key ¬= high-values
         Call aggregate-begin
         prev = new
         Read TP-trip Into new At end Call err3.


   Direction-Control-Break:
      Call time-control-break.
      out-time-int = dir-agg-time-code.
      Call output(2).


   Time-Control-Break:
      Call output(1).
   Output(i):

*   Write output   *
      If calc-sample-size(i) > 0
         calc-hist(i,j,*) = calc-hist(i,j,*) / calc-sample-size(i,j)
         out = calc(i,j) By-Name
         Write overload-aggregate from out.
      calc(i,*) = 0.
```

B.1.3.2.1.6 Form Daily-Layover-Aggregate File

```
   Program Variables:

      Modification of usual input record definition:

      new
         key
            line                     * file now sorted in same  *
            direction                *  order as key            *
            run
            trip
         data
            bus
            TP
            .
            .
            .

       prev (same as above)

      out
         key
            line
            direction
            day-int
            (time-int)
            (trip)
         data
            sched-layover
            layover-dev
            sample-size

      calc(2)
         data (same as above)

*     calc(i)                                                 *
*        i = 1 - time interval or trip aggregation            *
*            2 - direction aggregation                        *


   Main:

      Call process-header.
      Call initialize.
      Call process-data.


   Initialize:

      calc = 0.
      Read TP-trip Into prev At end Call err1.
      Read TP-trip Into new At end Call err2.
      Call aggregate-begin.
   Process-Data:

      Do While new-key ¬= high-values
         Do While new-key = prev-key

            prev = new.
            Read TP-trip Into new At end new-key = high values.

         Call trip-end.
      End.


   Aggregate-Begin:

*  Initialize out-key data       *
      out = 0.
      out-day-int = day-table(header-day).
      If TCD-code ¬= all
         out-time-int = time-int-table(new-data2,new-key).
      Else out-trip = new-trip.
      out-line = new-line.
      out-direction = new-direction.



   Trip-End:

*  When new record is from different trip, line, run or   *
*       direction, perform necessary summaries             *

      If new-run = prev-run
         layover =  new-time minus prev-time
         calc-sched-layover(*) = calc-sched-layover(*) +
            ((new-time + new-adj-sched-dev) minus
             (prev-time + prev-adj-sched-dev))
         calc-layover-dev(*) = calc-layover-dev(*) +
            (laycver minus calc-sched-layover(*))
         calc-sample-size(*) = calc-sample-size(*) + 1.

      time-int = time-int-table(new-data2,new-key).
      If new-line ¬= prev-line or new-direction ¬= prev-direction
         Call direction-control-break.
      Else If time-int ¬= out-time-int or
                 new-run ¬= prev-run or
                 TCD-code = all
              Call time-control-break.
*   Otherwise, perform no further processing   *

*   Have first record in new trip--read another   *
      If new-key ¬= high-values
         Call aggregate-begin
         prev = new
         Read TP-trip Into new At end Call err3.
   Direction-Control-Break:
      Call time-control-break.
      If TOD-code = all out-trip = dir-agg-trip-code.
      Else out-time-int = dir-agg-time-code.
      Call output(2).


   Time-Control-Break:
      Call output(1).


   Output(i):

*    Form averages and output    *
      If calc-sample-size(i) > 0
         calc-sched-layover(i) = calc-sched-layover(i) /
            sample-size(i)
         calc-layover-dev(i) = calc-layover-dev(i) /
            sample-size(i)
         out = calc(i) By-Name.
         Write layover-aggregate from out.
      calc(i) = 0.
```

B.1.3.2.2 Update

The two input files (aggregate and old-summary) and the output file (new-summary) have the same format.

```
  Main:

     Call process-header.
     Call initialize.
     Call process-data.


  Initialize:

     Read aggregate Into aggregate At end Call err1.
     Read old-summary Into summary At end Call err2.


  Process-Data:
     Do While aggregate-key ¬= high-values and
              summary-key ¬= high-values

        If summary-key < aggregate-key
           Write new-summary From summary
           Read old-summary Into summary
              At end summary-key = high-values.

        Else If summary-key = aggregate-key
                 Call update
                 Read aggregate Into aggregate
                    At end aggregate-key = high-values.

             Else Write new-summary From aggregate
                  Read aggregate Into aggregate
                     At end aggregate-key = high-values.


Update:

   For i = 1 to max-summary-elements

      If summary-element(i) is a sum
         summary-element(i) = summary-element(i) +
                  agg-element(i).

      Else If summary-element(i) is an average or percentage
              old-size = summary-sample-size(i)
              old-avg = summary-avg(i)
              Call average(old-size,old-avg,
                  agg-sample-size(i),agg-avg(i),
                  summary-sample-size(i),summary-avg(i))
              new-avg = summary-avg(i).
```

```
*  When a variance is updated, previous step will have updated
   the average (new) pertaining to that variable  *

               Else If summary-element(i) is a variance
                            old-var = summary-var(i)
                       Call Variance(old-size,old-avg,old-var,
                            agg-sample-size(i),agg-avg(i),agg-var(i),
                            summary-sample-size(i),new-avg,
                            summary-var(i)).
                    Else If summary-element(i) is an extreme
                            If agg-element(i) is more-extreme-than
                                  summary-element(i)
                               summary-element(i) = agg-element(i).


   Average(old-size,old-avg,addl-size,addl-avg,new-size,new-avg):

 *  Method to update an existing average (old) when another
    average (addl) becomes available) *

      new-size = old-size + addl-size.
      If new-size > 0 Then
         x12 = old-avg minus addl-avg
         n23 = addl-size / new-size
         new-avg = old-avg minus (n23 x x12).

      Else Call error-handling.


   Variance(old-size,old-avg,old-var,addl-size,addl-avg,addl-var,
      new-size,new-avg,new-var):

 *  Method to update an existing variance (old) when
    another (addl) becomes available *

      new-size = old-size + addl-size.
      If new-size > 0 Then
         x13sg = old-avg**2 minus new-avg**2
         x12sq = old-avg**2 minus addl-avg**2
         v12 = old-var minus addl-var
         n23 = addl-size / new-size
         new-var = old-var + x13sq minus (n23 x (v12 + x12sq)).

      Else Call error-handling.
```

B.1.3.3 Form New-Perf-Data-Base Tape.

```
Read old-perf-data-base header.
Read TP-trip header.

Form new-perf-data-base header From
   old-perf-data-base header and TP-trip header.
Write new-perf-data-base header.

Do While old-perf-data-base records exist
   Read old-perf-data-base Into temp.
   Write new-perf-data-base From temp.
Enddo.

Do While TP-trip records exist
   Read TP-trip Into temp.
   Write new-perf-data-base From temp.
Enddo.
```

B.1.3.4 Form Daily Exception Reports.

The detailed processes comprising this module are described below.

B.1.3.4.1 Form Daily Schedule Deviation Report

```
   Program Variables:

      save(3,trip-agg-code)
         TP
         time
         adj-sched-dev
         passengers-aboard

      limit
         early
         late


Main:

   Call process-header.
   Call initialize.
   Call process-data.


Initialize:

   Read selection-criteria into limit.
   Read TP-trip into new
      at end Call error1.
   prev = new.
   Call read-three-trips.
   If new-key = high-values Call error2.
   Call first-trip.


Process-Data:

   Do While new-key ¬= high-values

*  Trip changes handled in read-trip  *
         Do While new-line = prev-line
              and new-direction = prev-direction

            Call check-tolerances.
            save(1,*) = save(2,*).
            save(2,*) = save(3,*).
            prev = new.
            Call read-trip(3).

         Call last-trip.
```

```
Read-Three-Trips:

   limit-count = 0.
   save = 0.
   Call read-trip(1).      * assumes at least 3 trips in *
   Call read-trip(2).      * a given line & direction    *
   Call read-trip(3).


Read-Trip(i):

   Do While new-key = prev-key

      prev = new.
      If prev-TP ¬= begin-code and prev-TP ¬= end-code
         save(i,prev-TP) = prev By-Name.
      Read TP-trip Into new
         at end new-key = high-values.


Check-Tolerances:

   For i = 1 to trip-agg-code

      If save-sched-dev(2,i) < early or
         save-sched-dev(2,i) > late

         Write prev-key, save(2,i), save(1,i), save(3,i)
         limit-count = limit-count + 1.


First-Trip:

   For i = 1 to trip-agg-code

      If save-sched-dev(1,i) < early or
            save-sched-dev(1,i) > late
         Write prev-key, save(1), 'first-trip', save(2)
         limit-count = limit-count + 1.


Last-Trip:

   Call check-tolerances.
   For i = 1 to trip-agg-code

      If save-sched-dev(3,i) < early or
            save-sched-dev(3,i) > late
         Write prev-key, save(3,i), save(2,1), 'last-trip'
         limit-count = limit-count +1.
   Write prev-line, prev-direction, limit-count.
   If new-key ¬= high-values
      prev = new.
      Call read-three-trips
      Call first-trip.
```

B.1.3.4.2 Form Daily Overload Report

```
          Program Variables:

             save(3,trip-agg-code)
                TP
                time
                adj-sched-dev
                max-psgrs
                max-psgr-stop
                excess-psgrs

             overload-percent


       Main:

          Call process-header.
          Call initialize.
          Call process-data.


       Initialize:

          Read selection-criteria Into overload-percent.
          Read TP-trip into new
             at end Call error1.
          prev = new.
          Call read-three-trips.
          If new-key = high-values Call error2.
          Call first-trip.


       Process-Data:

          Do While new-key ¬= high-values

*  Trip changes handled in read-trip  *
             Do While new-line = prev-line
                  and new-direction = prev-direction

                Call check-tolerances.
                save(1,*) = save(2,*).
                save(2,*) = save(3,*).
                prev = new.
                Call read-trip(3).

             Call last-trip.
```

```
Read-Three-Trips:

   limit-count = 0.
   save = 0.
   Call read-trip(1).
   Call read-trip(2).
   Call read-trip(3).


Read-Trip(i):

   Do While new-key = prev-key

      prev = new.
      capacity = seat-table(prev-bus).
      load-limit = overload-percent x capacity.
      If prev-TP ¬= begin-code and prev-TP ¬= end-code
         save(i,prev-TP) = prev By-Name
         If prev-max-psgrs > load-limit
            save-excess-psgrs = prev-max-psgrs minus load-limit.
      Read TP-trip Into new
         at end new-key = high-values.


Check-Tolerances:

   For i = 1 to trip-agg-code

      If save-excess-psgrs(2,i) > 0
         Write prev-key, save(2,i), save(1,i), save(3,i)
         limit-count = limit-count + 1.


First-Trip:

   For i = 1 to trip-agg-code

      If save-excess-psgrs(1,i) > load-limit
         Write prev-key, save(1), 'first-trip', save(2).
         limit-count = limit-count + 1.


Last-Trip:

   Call check-tolerances.
   For i = 1 to trip-agg-code

      If save-sched-dev(3,i) < early or
             save-sched-dev(3,i) > late
          Write prev-key,save(3,i), save(2,i), 'last-trip'
          limit-count = limit-count + 1.
   Write prev-line, prev-direction, limit-count.
   If new-key ¬= high-values
      prev = new.
      Call read-three-trips
      Call first-trip.
```

B.2 Common Report Program Contents (reports 2.1 through 2.6)

Input file structure is identical to the output file stucture of given summary file:

```
new
   key
      key1
         line
         direction
         day-int
      key2
         (time-int)        * either time-int or trip   *
         (trip)            * will be present; not both *
   data  (varies with file)

prev: same as above.

calc
   data (same as above)
```

There are two outputs:

```
out-print:  print line
out-week:  summarized weekly data, same
           format as input.  It is accumulated
           when the DCW outputs are being printed;
           then is later read and printed.
```

Output data line structure includes same data (except possibly for sample-size).

'data' consists of four types:

1. sums (includes sample sizes)
2. averages or percentages
3. variances
4. extremes

Therefore, additional data of the same type can be incorporated by:

1. adding addl-sums to old-sums
2. Calling 'average' (described later), with the old and addl averages (or percentages) and sample sizes
3. Calling "variance (described later), with the old and addl averages, variances and sample sizes.
4. comparing old and addl extremes

The format of the summary files are given here; the data fields that are averages or percentages are indicated by 'a/p'; those that are variances are indicated by 'v'; those that are extremes are indicated by 'e'; the remaining ones are sums.

Section-15-Aggregate File

```
   Program Variables:

      out
         key
            line
            direction
            day-int
            time-int
         data
            psgrs-boarded
            bus-miles
            psgr-miles
            bus-mins
            psrg-mins
            capacity-miles
            seat-miles
            bus-trips    * sample size  *
```

Daily-Run-Aggregate File

```
   Program Variables:

      out1
         key
            line
            direction
            day-int
            (time-int)
            (trip)
         data
            TP
   a/p      sched-run-time
   a/p      run-time-dev
   a/p      adj-run-time-dev
   v        run-time-dev-var
            sample-size
```

```
Daily-Ride-Aggregate Files

   Program Variables:

      out1                 out            out2
         key                  key            key
            line                             data
            direction                           psgr-trips
            day-int                             psgr-miles
            (time-int)                 a/p      load-factor
            (trip)                              standee-mins
                                                st-min-per-st
         data                                   bus-trips  * sample size *
            TP
            psgrs-on
            psgrs-off
   e        max-psgrs
   e        max-psgr-stop
            standees
            standee-time(3)
            mins-wi-standees
            sample-size

Daily-Sched-Dev-Aggregate File

   Program Variables:

      out
         key
            line
            direction
            day-int
            time-int
         data
            TP
   a/p      hist(22)
            sample-size
```

Daily-Overload-Aggregate File

```
Program Variables:

   out
      key
         line
         direction
         day-int
         time-int
      data
         TP
a/p      hist(10)
         sample-size
```

Daily-Layover-Aggregate File

```
Program Variables:

   out
      key
         line
         direction
         day-int
         (time-int)
         (trip)
      data
a/p      sched-layover
a/p      layover-dev
         sample-size
```

Main:

```
   Call process-header.
   Call initialize(summary).
   Call process-data(summary).
   Call process-week.
   Call initialize(week).
   Call process-data(week).
```

```
Initialize(file):

   Reset out-print, out-week (to zeros or blanks).
   calc = 0.
   Read file into new.
   prev = new.
   Call print-header(new-key).


Process-Data(file):

   Do While new-key ¬= high-values
      Do While new-key = prev-key

         Move prev-data Into proper column of out-print.
         prev = new.
         Read file Into new At end new-key = high-values.

      Call print-line.


Print-Line:

   increment print-line-counter.
   If print-line-counter > page-size
      Call print-header(prev-key).
   Print out-print.
   Reset out-print.
   If new-key ¬= high-values
      If new-key1 ¬= prev-key1
         Call print-header(new-key).
   prev = new.


Print-Header(key):

*  Prints formatted key items and other related data (e.g.,
      time point names as well as numbers  *
   Print key-header.
   print-line-counter = 0.
   Process-week:

*  If there is more than one day-of-week category,
     also want a weekly aggregate, in same format as
     for individual DOW 'pages'  *
   If header-DOW-code = all Return.
   Call sort-for-day.
   Call initialize(summary).

   Do While new-key2 ¬= high-values
      Do While new-key2 = prev-key2

         Call update.
         prev = new.
         Read summary into new At end new-key2 = high-values.

      Call write-week.


   Write-Week:

      out-week-key = prev-key.
      out-week-day-int = day-agg-code.
      out-week = calc By-Name.
      Write week From out-week.
      calc = 0.
      prev = new.


   Sort-For-Day:

*  Reorder 'summary' so can add successive records (until
      time-int/trip changes) to get weekly totals  *
   Rewind summary.
   If TOD-code = all
      Sort by line, direction, trip, TP, day-int.
   Else Sort by line, direction, time-int, TP, day-int.


   Update:

      For i = 1 to max-data-elements

         If prev-data-element(i) is a sum
            calc-data-element(i) = calc-data-element(i) +
               prev-data-element(i).
         Else If prev-data-element(i) is an average or percentage
                 old-size = calc-sample-size(i)
                 old-avg = calc-avg(i)
                 Call average(old-size,old-avg,prev-sample-size(i),
                      prev-avg(i),calc-sample-size(i),calc-avg(i))
                 new-avg = calc-avg(i).
```

```
             Else If prev data element(i) is a variance
                     old-var = calc-var(i)
                     Call variance(old-size,old-avg,old-var,
                        prev-sample-size(i),prev-avg(i),prev-var(i)
                        calc-sample-size(i),new-avg,calc-var(i)).
                  Else If prev-data-element(i) is an extreme
                          If prev-data-element(i) more-extreme-than
                                calc-data-element(i)
                             calc-data-element(i) = prev-data-element(i).


  Average(old-size,old-avg,addl-size,addl-avg,new-size,new-avg):

*  Method to update an existing average (old) when
   another average (addl) beccmes availatle)  *

     new-size = cld-size + addl-size.
     If new-size > 0 Then
        x12 = old-avg minus addl-avg
        n23 = addl-size / new-size
        new-avg = old-avg minus n23 x x12.

     Else Call error-handling.

  Variance(old-size,old-avg,old-var,addl-size,addl-avg,addl-var,
     new-size,new-avg,new-var):

*  Method to update an existing variance (old) when
   another (addl) becomes available *

     new-size = cld-size + addl-size.
     If new-size > 0 Then
        x13sq = old-avg**2 minus new-avg**2
        x12sq = old-avg**2 minus addl-avg**2
        v12 = old-var minus addl-var
        n23 = addl-size / new-size
        new-var = old-var + x13sq minus (n23 x (v12 + x12sq)).

     Else Call error-handling.
```