REPORT NO. FAA-RD-73-77

AIRPORT INFORMATION RETRIEVAL SYSTEM (AIRS) SYSTEM DESIGN

Manuel F. Medeiros
Julie Sussman

JULY 1973

FINAL REPORT

Prepared for
DEPARTMENT OF TRANSPORTATION
FEDERAL AVIATION ADMINISTRATION
Systems Research and Development Service
Washington DC 20591

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
FAA-RD-73-77		

4. Title and Subtitle	5. Report Date
AIRPORT INFORMATION RETRIEVAL SYSTEM (AIRS) SYSTEM DESIGN	July 1973
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Manuel F. Medeiros and Julie Sussman	DOT-TSC-FAA-73-16

9. Performing Organization Name and Address	10. Work Unit No.
Department of Transportation Transportation Systems Center Kendall Square Cambridge MA 02142	FA-306/R4111
	11. Contract or Grant No.
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address Department of Transportation Federal Aviation Administration Systems Research and Development Service Washington DC 20591	Final Report
	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This report presents the system design for a prototype air traffic flow control automation system developed for the FAA's Systems Command Center. The design was directed toward the immediate automation of airport data for use in traffic load predictions and flow control operational support. The system employed computer services offered by commercial time-sharing companies. The system was also designed to serve as a technology foundation and an experimental tool from which subsequent automation specifications could be derived. The report covers the design decisions associated with the data base, the user interface, the user language, the special processing and the numerous operational considerations. Also included are the supporting program designs for data base updating and integrity maintenance. Finally, the report presents several recommended improvements to the automation system.

17. Key Words	18. Distribution Statement
Flow Control, Air Traffic Control, Central Flow Control Facility, Automation, Information Retrieval, Airport Information, Flow Control Procedures	DOCUMENT IS AVAILABLE TO THE PUBLIC THROUGH THE NATIONAL TECHNICAL INFORMATION SERVICE, SPRINGFIELD, VIRGINIA 22151.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	188	

Form DOT F 1700.7 (8-69)

PREFACE

The automation system described in this report was conceived and developed in the Information Sciences Division of the Transportation Systems Center (TSC). The effort was sponsored by the Systems Research and Development Service of the Federal Aviation Administration (FAA). The work was directed toward the immediate automation of the Systems Command Center of the Air Traffic Service through the development of a prototype automation system employing computer services offered by commercial time-sharing companies. The prototype system was also required to serve as a technological foundation and an experimental system from which future automation specifications could be derived for subsequent phases in the System Command Center's automation plan. The prototype, known as AIRS, became operational in January 1972 and has since evolved to the level described in this system design document. The first version provided airport traffic information; later improvements included airport arrival delay predictions, advanced flow control procedures, quota flow control procedures and airport reservation data updating.

A development of this scope would not have been possible without the assistance, advice and cooperation of many people. Within the Transportation Systems Center, the authors sincerely appreciate the substantial contributions of Richard D. Wright in the development and documentation of the graphical display capabilities for the AIRS prototype automation system. Special thanks to John R. Coonan for the superior guidance and planning which assured the success of this automation endeavor. Also greatly appreciated is the technical counseling and the management support of Juan F. Bellantoni in the formulation of the prototype system.

At the FAA, the authors are deeply indebted to the Project Manager, Thomas E. Armour, for his steadfast support and his comprehensive understanding of the automation requirements, the technical approaches and the real world considerations in guiding the development of this automation system. The timeliness and efficiency of this development is due to a great extent to the outstanding coordination and cooperation of the Air Traffic Flow Control Automation Working Group, headed by Michael E. Perie. But the greatest respect and heartfelt gratitude for continuous contributions in formulating, evaluating and improving the operational design of this automation system is directed to the staff of the Systems Command Center, especially our automation contact Robert A. Christopher. This staff patiently labored through the birth and growth pains of this new automation system and unselfishly gave of their valuable time and

effort to fruitfully and comprehensively integrate the automation system into the daily functions of the Systems Command Center.

Many valuable programming ideas incorporated into the implementation of AIRS were developed with the help of Gerald Jay Sussman of M.I.T.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF TABLES

LIST OF ILLUSTRATIONS

ACRONYMS AND ABBREVIATIONS

AFCP	Advanced Flow Control Procedures
AIRS	Airport Information Retrieval System
ARO	Airport Reservation Office
ARTCC	Air Route Traffic Control Center
ATC	Air Traffic Control
CFCF	Central Flow Control Facility
FAA	Federal Aviation Administration
FSS	Flight Service Station
GA	General Aviation
GMT	Greenwich Mean Time
IFR	Instrument Flight Rules
NAS	National Airspace System
OAG	Official Airline Guide
SCC	Systems Command Center
TSC	Transportation Systems Center

1. INTRODUCTION

The Air Traffic Service of FAA has stated a critical need for automating certain operational functions indigenous to the Systems Command Center (SCC) and concerned with the nationwide monitoring of air traffic control system status and control of air traffic flows (Ref. 1). The Transportation Systems Center (TSC) is automating the most immediate of these critical needs. This document describes the TSC design and development of AIRS, an airport-oriented automation system concerned with the problems of balancing demand and capacity. AIRS is an acronym for Airport Information Retrieval System. This acronym accurately portrayed the first operational version of January 1972, which rapidly retrieved demand information covering 1000 airports. However, the AIRS of today has been greatly expanded to fill more of the SCC needs. In today's version, arrival delay predictions, allocations for Advanced Flow Control Procedures (AFCP) and flow rates for Quota Flow Procedures are readily produced upon request. Every day the latest flight schedules are routinely processed by the Airport Reservation Office (ARO) of the SCC to upgrade the accuracy of the monthly updated Official Airline Guide (OAG) data (provided by the R. H. Donnelley Corp.) which constitutes the heart of the AIRS data bank.

The critical need for automation is caused by the large number of air traffic operations within the national airspace and their complex relationships, particularly during abnormal conditions. It often happens that a single airport can disrupt the orderly movement of air traffic in large areas of the country. For example, if Chicago's O'Hare Airport were to suffer a substantial reduction in landing capacity as a result of inclement weather, the traffic converging on O'Hare would soon saturate the airspace of the Chicago Air Route Traffic Control Center (ARTCC) and spill into the adjacent ARTCC's. The presence of these delayed flights affects the movement of other flights in the congested airspace. Some means of predicting the traffic demand upon airports and the magnitude and duration of arrival delays under adverse conditions would greatly aid in the accommodation and control of the affected aircraft and would allow the ARTCC's to maintain orderly, safe, and efficient movement in their airspace. Accurate predictions of impending excessive delays will permit the impacted field facilities (ARTCC's) to adjust procedures, traffic distributions, and staff to handle the load and can be used, if necessary, to implement nationwide flow control procedures under the direction of the SCC to maintain the flow within acceptable limits.

There are also enroute traffic delay situations which have the potential for nationwide propagation. However, these are less frequent than airport-induced problems. Enroute traffic delay prediction is far greater in the complexity and scope of needed data for automation than is airport delay prediction. Indeed, enroute and airport delay interactions can not be separated in reality. But readily available data tapes of airline schedules, giving airport departure and arrival times, can serve as a first approximation for automation purposes of enroute flight times during normal enroute status. Fortunately, the bulk of airport arrival delay situations do not coincide with other enroute delays such as thunderstorm reroutes and major route restrictions. The airports most often suffer local wind, fog, and precipitation induced delays which have little effect, if any, enroute.

The automation effort is divided into three phases. The first phase is the most urgent, the development of a prototype. The second is a more expanded prototype with greater scope and capability. The third phase is a full and comprehensive automation system based upon a dedicated computer system. The three levels of automation are phased to provide beneficial empirical results for consideration in the subsequent levels. Responding to the immediate need and concentrating on the major problem situation, the airports, TSC conceived an information retrieval system (AIRS) based upon airline schedules. AIRS was scoped for timely development by delaying until a later time any detailed enroute provisions. More important, the development and implementation plan for AIRS provided maximum exchange of information through frequent interaction between the controllers and the program developers. This interaction began as early as possible with the rapid implementation of a version of AIRS which contained only the bare operational elements. Sufficient capability was provided to tabulate traffic demands and list certain flight information for any of the 1000 airports having airline traffic. The SCC experience, comments, and criticisms were then factored into the current design and the subsequent new features. The net result was the rapid evolution of this prototype toward the specific needs of SCC, and the maximum accumulation of operational experience with the associated training and the integration of automation assistance into the normal functions of the SCC. This operational experience provides a sound foundation from which the specifications for the final phase III automation can be developed. The applicability of this experience to phase III specifications is reinforced by incorporating into the AIRS prototype the latest state-of-the-art technology in software design and man-machine interface. AIRS gives the SCC staff a language

which is natural to their daily operations. The language provides for easy program control and operation even though the staff actually operates the computer through remote teletype devices. The design maximizes services and minimizes special computer training through extensive program-managed operations and built-in tolerance for and forgiving of typing or command errors. The current AIRS offers a wide scope of data processing and retrieval capability and selectivity which portends an extended useful life.

AIRS is more than the program which the SCC staff operates. It is a system which includes nightly checking programs as well as the monthly updating programs. AIRS makes extensive use of data files stored at the computer facility in Waltham, Mass. These files are retained on disc storage devices for immediate random access by AIRS. Experience has shown that computer malfunctions can damage these files and destroy the data to varying extent. Some damage is undetectable to the AIRS program and may produce erroneous traffic information and even propagate the damage throughout other undamaged data files. A series of file integrity checkout programs is run each night and reports on the status of these data files in order to detect as early as possible any damage which might have occurred during the day's operations. Some of the file damage can be and is repaired by these nightly programs after evaluating the detected damage. The AIRS program itself has some built-in file integrity checking and file repair capability too. However, only those detection and repair operations which do not adversely affect the response time are built into AIRS. The nightly programs are more thorough in their file checking. The other major part of the AIRS system is the set of monthly updating programs. Approximately two months of flight data is maintained in the AIRS central data bank at any one time. When next month's airline schedule data tape is received from R. H. Donnelley, usually one or two weeks prior to the next month, the central data bank is purged of the earliest month's data and the new data is appended to the current month. The update encompasses programs to input data from the tape, filter it for flights of interest, encode it in terms common to the data files, sort it, merge it with that part of the current month's data to be retained and finally, develop the cross-reference files vital to the efficient and speedy retrieval process.

This introduction to AIRS would not be complete if we did not discuss the computer selection. The SCC automation requirements discussed above are of a real time nature. The information needs are often spontaneous, responding to an actual or impending problem. Further, the data bank must be

updated throughout the day to maintain an acceptable level of accuracy. These two factors dictate that the automation be on-line accessible in the SCC facility, be real time in interrogate/response service and be efficient enough to handle the support and computational loads without excessive delay. Further, it is highly desirable that the update operations, customarily performed by a separate activity from the central flow control operations, be simultaneous yet non-interfering with the flow control operations. The only computer systems which meet the above requirements are time-sharing computers. The survey of available government and commercial time-sharing systems which would meet the operational and the programming needs for the AIRS prototype resulted in the selection of First Data Corporation in Waltham, Mass., which offered time-sharing service on a PDP-10 computer system, seven days per week, 16 hours per day. It is important to note that not having a dedicated time-sharing system specifically for SCC automation purposes imposes considerable constraint in the design and operation of AIRS. In reality the SCC computerized operations are only able to command a fraction (1/60) of the time-shared computer's resources during heavy use. AIRS was therefore scoped to perform within this resource limitation. Further expansion and evolution of AIRS depends upon commanding more of the computer's resources if acceptable response times are to be maintained.

This document is designed to give a detailed overview of the total AIRS system: the operational, the nightly checking and the monthly updating programs. The document is supplemented by the "AIRS User's Guide" and the "AIRS System Support Manual", both to be published shortly. (A preliminary version of the "AIRS User's Guide" has been distributed, Reference 7). The user's guide details the language and operations of AIRS from the SCC operators' point of view, and the system support manual views the data base management from an operational support perspective. The combination of these three documents provides a thorough understanding of the current version of AIRS. Further working level documentation will be provided, however, to compile a complete AIRS programmers' reference library. The working level documents are scheduled for completion later and contain complete subroutine descriptions, flow charts, and program level summaries and provide a compilation of all program source code listings. The latter material would be required by the programmers who might undertake further expansions to AIRS or incorporate major elements of AIRS into other phases of the SCC automation.

This document presents the major elements which constitute the AIRS system of today and, where fruitful, the

document develops the original system design considerations, illuminating the evolutionary factors and effects. Prior to immersion into the discussion of these major elements, a general discussion is presented which covers the background and key factors in TSC's involvement in automating the Systems Command Center. The document then proceeds to expand upon the data requirements, the AIRS language requirements and the user interface. The viewpoint then switches to examination of internal program operations such as the centralized data files, request analysis, retrieval operations and the spectrum of processing of the retrieved data for specific purposes including demand loads, delay predictions and control rates for Quota Flow. Next, the real time entry of data is examined in terms of program processing. At this point a sufficient understanding of AIRS processing is achieved to introduce the fail-safe requirements and the file management operations crucial to the security and integrity of the AIRS centralized data bank. Completing the understanding of the AIRS operational program, the document covers the operations in the automatic usage records and message features. The monthly update programs are then detailed. The final chapter presents a compilation of recommended improvements and expansions to AIRS or its successor.

2. GENERAL DISCUSSION

The Central Flow Control Facility (CFCF) was established in April 1970 to oversee the flow of aircraft among the ARTCC's. Its primary objective is the balancing of national air traffic flow to minimize delays without exceeding controller capacity, thus maintaining safe and orderly flows. This facility currently comprises the heart of the FAA's Systems Command Center. Initial operations at the CFCF concentrated on reacting to problem situations. It soon became clear that some of these problems might have been prevented if the facility could accurately forecast traffic loading. The only practical approach to such forecasting is through automation. This need, along with several others, was factored into the plan (Ref. 1) to develop full automation of the SCC.

Anticipating the requirements of the plan, the FAA's Office of Management Systems, Management Analysis Division began a pilot effort to introduce automation of airport and enroute traffic forecasting into the CFCF. This pilot development is reported in Reference 2. It became operational in September 1971 and provided typical day traffic loads for 14 high density airports and for key sectors along several major routes between selected city pairs. This pioneering effort was significant in focusing attention on the benefits of timely automation using commercial time-sharing computer resources. However, the role of continued automation of the SCC was beyond the jurisdiction of the FAA's Office of Management Systems. At this point the Transportation Systems Center was asked to assume responsibility for this pilot operation and the development of a prototype system. In October 1971 TSC accepted this responsibility. Evaluation of the operational use of the pilot programs revealed that its enroute load forecasting (Ref. 3) did not adequately cover the total traffic and therefore could not produce a useful level of accuracy. Expanding the enroute program to a useful level of coverage would require an undesirable amount of time and effort for consideration early in the prototype phase. The pilot program for airport loading, however, was of greater value to the CFCF operations. Its use resulted in frequent requests by the controllers for expansion to cover more than the present 14 airports. The TSC evaluation of the program's expandability concluded that the program was incapable of being expanded within the host time-sharing system and also that other needed upgrading was not possible within the available core and the program's language restrictions (pilot programs were written in a mix of BASIC interactive language and FORTRAN batch language). It was decided that all the available time-sharing computer

services would be surveyed and the most applicable selected. Appendix A tabulates the principal desired features used as selection criteria in this survey. Because of the final selection of a computer system other than the one used in the pilot programs, a period of parallel operation, pilot and prototype, was planned until the prototype system could assume the automation role. Work began on the prototype in October 1971 and the first operational version was implemented in mid January 1972 in the CFCF.

The prototype, AIRS, covered all (1000) airports which had scheduled air traffic which originated in and/or was destined for the United States. It had the ability to accept requests of a form natural to the controllers and to respond to requests qualified by specifications such as date, time period, aircraft types, arrivals, departures, airports of interest and airport pair traffic groupings. The flexible language and the selectivity in requesting information offered a considerable advancement over the pilot programs, which were no longer used. The pilot programs were phased out at the end of February 1972.

Many decisions were made in designing AIRS. It is appropriate to quickly review these decisions which have a profound impact on the architecture and functional blocks of the AIRS system. Further details on these decisions will be presented throughout the remaining chapters in the design areas they most affect. First and foremost, AIRS was to be a prototype system. Prototypes are usually aggressive in advancing the state of the art, experimental in that fulfilling all goals involves calculated risks, and timely with regard to producing results with breadboard or unoptimum elements. The design of AIRS strove toward these qualities with one overriding consideration, that of producing a practical, useful automation system.

The second decision was that the R. H. Donnelley airline schedule tape be the main source of information. Third, the system (like the original pilot program) would be developed and implemented on a time-sharing computer and be operated through remote terminals (teletypes and display devices). Fourth, the operation of AIRS would be such as to minimize computer training requirements by controlling as many as possible of the computer functions within the single AIRS system. A fifth decision was made to develop AIRS in the FORTRAN high level programming language and, where essential to the efficiency and capability, to use limited amounts of machine level assembly language.

The sixth decision was to provide a full range of AIRS operational capabilities for all airports in the data base.

The seventh was to make the information retrieval capability as broad, yet as specific as required to meet both the foreseen and unforeseen problem needs. This implied that AIRS could not be restricted to retrieve and produce a fixed list of reports as did the pilot programs, but must possess an internal ability for structuring the retrieval process and report formats subject to the needs of each information request. The next decision concerned the response time for AIRS' information retrieval. Most of the problems handled by the SCC are transient in nature and require that the automation be fast in order to be useful. It was therefore decided that rapid retrieval was of critical importance in the design of AIRS. Timely updating is also important, but in a conflict situation where opposing design choices can favor either speedy retrieval or fast updating, the speedy retrieval approach is to be selected.

Another decision was to design AIRS to be easily expanded. Such expansions as on-line updates, delay prediction, graphical plots and enroute approximations were envisioned. The AIRS design turned out to be upgradable enough to integrate computations for AFCP and Quota Flow Procedures, the needs for which developed after the first implementation.

The next decision was to centralize the data into one bank as opposed to maintaining separate data for each program. This produced the desirable situation of minimizing the requirements for updating, maintaining and storing the data. It was decided that simultaneous access to the centralized data bank should be provided in order to accommodate parallel operation of AIRS by several SCC users. This required that operations involving writing on the files be minimized and intermittent since two users can not write on a file at the same time (i.e. two users must take turns writing when there is a conflict).

Decisions were also made concerning exchange of comments and record-keeping to help detect operational difficulties and areas for improvement. Specifically, two-way communication through records maintained by AIRS was incorporated into the design.

The last of these important original design decisions was to minimize the input and output volume whenever practical. For example, if the flow controller desired information on the arrival traffic demand at J. F. Kennedy Airport, the input request to AIRS might have been as lengthy as: "ARRIVALS AT J. F. KENNEDY AIRPORT FOR TODAY FROM CURRENT TIME THROUGH THE NEXT FOUR HOURS" or be as short as "A JFK." This shorter form required that the

program recognize special codes and apply default day and time periods when appropriate. Since the users are air traffic controllers in need of quick assistance and not typists or computer operators, the shorter input is obviously best. The output too, can be a lengthy work of formatted art, but again, only enough output to unambiguously convey the desired information is wanted. The teletype devices used as remote input/output stations for the computer have slow printing speeds. Titles, formats, and data identifiers can consume considerable printing time in comparison to the essential data being reported. Brevity of reports will favorably improve AIRS response times. Further, optional levels of detail within these reports is another technique for minimizing outputs. The user, if he desires more data, should be able to request it after getting the initial abbreviated results. This means that AIRS must retain sufficient data to produce upon request these expanded reports following each retrieval operation (after it outputs the abbreviated data). In a later version of AIRS, the user would be given even greater control of this output, control over the content, order and range of data. This extra control permitted him to optionally set the output formats for detailed reports.

The above decisions took place in October 1971. Since then several major decisions have been added. For instance in January 1972, decisions were made to add AFCP and the ARO automation and in January 1973 to add Quota Flow Procedures to AIRS. These were major additions and will be treated in detail in the subsequent chapters. As a result of these expansions to AIRS, particularly the ARO real-time updates, several major decisions were made to insure the integrity and security of the AIRS central data bank. A period of computer malfunctions prompted serious concern when loss of automaticn for several hours followed some of the computer failures. It was decided that AIRS be equipped with internal integrity testing procedures and with file damage checking. Further, file checkout programs would be developed for automatic daily examination and repair of files.

In October 1971, the initial design decisions discussed above were completed and the program coding begun. The programs were developed and checked out during the following months and on January 12, 1972 a first version was turned over to the CFCF for operational use. The flow controllers could ask for arrival and/or departure data for any of 1000 airports known to AIRS. This would get a summary of hourly traffic counts and optionally the user could request detailed data showing individual flights. Requests for traffic could be specified with a broad range of conditions

such as retrieving just the arrival flights for an airport from other specified airports which are jet aircraft and will arrive on a particular date and time period. By the end of June 1972 the Advanced Flow Control Procedures (Ref. 4 and 5) were operationally integrated into AIRS. The automation of these procedures was substantially improved over the predecessor AFCP automation being phased out of the Kansas City ARTCC computer. The AIRS/AFCP could apply the procedures to any airport in the AIRS data bank instead of just the five fixed airports in the predecessor. The AFCP zone structures could be created or modified in real time instead of being pre-structured and fixed. The control criterion of maximum air delays was expanded to consider maximum stack (aircraft holding) limits in regulating traffic flow, and the interval for allocations used in regulating traffic flow rates was flexible from one allocation per hour to six in order that controlled traffic be better distributed.

In August of 1972, the Airport Reservations Office began updating the AIRS data base for four high density airports. This real-time update capability ppovided improved accuracy for AIRS data, not only for these four airports but for all the airports known to AIRS. The improved accuracy of the AIRS data and the evolution of new SCC flow control procedures led to the most recent expansion to AIRS. In February 1973, the Quota Flow Procedures (Ref. 6) were added to the AIRS automation system. Quota Flow is the successor to AFCP and portends a wide spectrum of flow control application. In fact, this newest addition to AIRS has already seen extensive use in the successful control of nationwide traffic during many serious airport overloads.

The above three paragraphs highlight only a few of the major events in the evolution of AIRS. Appendix B gives a more comprehensive view of the AIRS chronology, clearly showing the dynamic nature of this prototype automation development. It depicts a rapidly evolving implementation approach for automating an operational ATC facility. The approach contrasts with most of the FAA's automation history. The close relationship between TSC and FAA enabled such an approach to be successful.

Looking ahead, AIRS possesses the potential for further expansion in Phase II. In the near future limited amounts of air traffic data will be available from the ARTCC's in real time through existing teletype networks. This National Airspace System (NAS) data could be used to update progress of specific flights as they transit selected coordination fixes in each ARTCC. This data would increase accuracy of estimated arrival times. It is also possible to develop a

pseudo enroute prediction feature for AIRS which would provide approximate enroute traffic load predictions. This could be rapidly automated and be in use during the time the detailed enroute prediction ability of Phase II was being developed. A third area for AIRS expansion is in automatic communications, using the SCC's current teletype circuits. Automatic transmission of AIRS reports and flow control messages to field facilities would expedite many of the functions based upon AIRS data. The full potential of AIRS has yet to be achieved. As a prototype it has pioneered an aggressive and fruitful realm of automation. Until it is superseded by the next planned level of automation, AIRS should provide the FAA's System Command Center with a substantial foundation from which to perform the functions of centralized air traffic flow control.

3. DATA REQUIREMENTS

The success or failure of many automation systems is greatly dependent upon the balance between data required by the system and data produced by it. It can happen that the workload supporting the computer is far greater than the value of the outputs. A great deal of attention was given to the balance of inputs versus outputs in the scoping and design of AIRS. In striving for the AIRS prototype system to be both practical and useful, major trade-offs were made between the operational workload required to maintain the data bank and the potential accuracy level of the system's output. If AIRS were to be an ARTCC flight-controlling automation aid such as the NAS automation, there would be no trade-off possible because the minutest data inaccuracies would cause unacceptable errors. In the SCC application, however, there is room for trade-offs because of the predictive nature of the desired output. There are innumerable real world effects which complicate the problem of predicting the time of arrival of each aircraft at any specific place. Ideally, if one could have perfect and complete aircraft status information, the predictions would be as accurate as possible. The accuracy of this ideal case would be perfect for the near future, but would degrade as one predicted further into the future and the real world effects noticeably perturbed actual movement from the predicted aircraft movement. Since inaccuracies will exist even in the perfect data collection system, it is reasonable to consider less than perfect data collection, to see if and when the accuracy will significantly degrade below an acceptable level. The obvious benefit of reduced data collection is the associated workload reductions in maintaining the data bank. SCC trade-offs must also include practical considerations in transmitting or collecting data from the nationwide sources into this centralized data bank. Teletype networks, voice lines, data tapes and correspondence are available for transmitting different kinds of data. Let's consider what kinds of flow control information might be available through these collection techniques. The teletype and voice circuits are very similar in their message handling abilities. The data transmitted is usually of the summary, filtered or abstracted variety. Estimates of landing capacities, numbers of aircraft holding, average delays, restriction messages and the like are typical of what can be expected in support of the SCC automation system. The scope of these messages can not be expected to cover all airports each day. AIRS must therefore permanently retain some nominal information of this type and permit updates only when appropriate. Indeed, flight schedules might be updated for a few airports in this manner as ARO does. The one

exception to the abbreviated data handling ability is when there are computers directly connected to each end of the circuit. Large volumes of relatively raw data might be transmitted up to the capacity of the circuit's bandwidth. In the prototype design, this form of data entry was deemed to be sufficiently far in the future (time was required for hardware and software development) to discount the NAS or terminal computers as an immediate source of flight data even though this data would be of great value in monitoring the ATC status. Aside from this real time source of flight data, there are files of data in the form of computer readable magnetic tapes. One such source is in the form of flight plan tapes (Bulk Store), used by the ARTCC's. These tapes contain a large percentage of the flight plans for short haul (less than 600 nm) air carriers, some military and some general aviation flights. Each ARTCC prepares its own tape and the formats are currently non-standard. These 21 ARTCC tapes would be of value in determining the numerous air routes between airports; but because the AIRS enroute load prediction capability was to be delayed for a later version, this source would not be applicable until a detailed enroute expansion is available.

Fortunately, a single source of flight data in magnetic tape form was available. The R. H. Donnelley Corporation produces a monthly tape for FAA containing all the scheduled domestic and foreign flights of concern to the CFCF. The data does not contain general aviation or military flight schedules, however. The flight schedule data contains for each flight leg the associated airports, flight times, aircraft type, flight identification and dates of operation. This tape would require minimum update workload and could provide a compact source of information from which airport traffic predictions could be made. The lack of military flight schedules was considered negligible because few military flights operated from the major commercial airports of interest. The lack of general aviation (GA) data was considered a more serious omission. The only source of GA flight data (IFR only) is from the ARTCC's or Flight Service Stations (FSS). This GA information would be impossible to obtain until a direct computer-to-computer network is established, and even then the data would only be complete for 30 minutes in the future and would degrade with longer predictions. In lieu of this future source of data, the most practical type of adjustment in traffic predictions for GA flights at each airport would be to apply a GA factor (a percentage of scheduled air carrier) to the scheduled traffic in predicting the total traffic for the airport.

The last source of field data concerns correspondence and reports. A limited amount of raw data, such as extra

section flights, are known early enough to permit limited utility of this flight data in the central data bank, In general, however, the sparse volume, and the long lead times from preparation to delivery to computer entry, diminish the value of this type of information. a practical use might be to input only holiday traffic extra sections information.

The above sources of data pertain to the monitoring and predicting of air traffic status. There are, however, two other kinds of data required for SCC automation, the procedural data and the AIRS system level data. The procedural data refers to control parameter and zone structures such as those used in computing quotas for Quota Flow Control Procedures. The control data and zones must be created by the SCC and entered into AIRS in the process of developing procedures for each airport of interest. To be operationally versatile, the AIRS prototype should be capable of both editing and retaining permanently this data in the central data bank.

The last kind of data required for SCC automation is the AIRS system level data. An airport information system cannot operate efficiently without fundamental data files such as airport codes, center codes, aircraft type codes, and special definition tables (e.g. Foreign airport groupings, air taxi grouping). This data must be up-to-date and complete if the system is to have full knowledge of its data categories for encoding/decoding and retrieval operations. AIRS uses these data tables in providing the widest flexibility in retrieval capabilities and in giving the maximum freedom of operation through a natural request language.

The above data collection workloads have ranged from practical to impossible and the contributions to accuracy from valuable to negligible. The success of the trade-offs made in scoping the data collection for AIRS can only be estimated at this time, but it appears that a good balance has been achieved. The resulting data requirements in support of AIRS are summarized in Table 3-1.

TABLE 3-1 AIRS DATA REQUIREMENTS SUMMARY

Data Description	Source/Form	Frequency	Entered by
FIELD DATA			
Flight schedules, air carrier	R.H.Donnelley (RHD)/Mag. Tape	Monthly	TSC
Extra section schedules	Airlines/Teletype & printed	aperiodic	ARO
ARO schedule updates for 4 airports (DCA, JFK, LGA, ORD)	FSS, ARTCC/ verbal, teletype, printed	Real-time daily	ARO
Airport landing capacity estimates	ARTCC, Airport/ verbal	edited as required	CFCF
Airport GA factor	CFCF, ARTCC, Airport/verbal	edited as required	CFCF
Departure delays	ARTCC, Airport/ verbal	as required	CFCF
PROCEDURAL DATA			
AFCP & Quota Flow control data	CFCF, ARTCC/ verbal	edited as required	CFCF
AFCP & Quota Flow Zone Structures	CFCF, / verbal	edited as required	CFCF
SYSTEM LEVEL DATA			
Airport Codes	RHD/mag. tape	monthly as required	TSC
Center Codes and associated airports	FAA Manuals/ printed	monthly as required	TSC
Area Codes and associated airports	CFCF/verbal	monthly as required	TSC
Aircraft type codes	RHD/mag. tape	monthly as required	TSC
Air Taxi code and associated airlines	RHD/printed	monthly as required	TSC
Airline Codes	RHD/mag. tape	monthly as required	TSC

4. CONTROLLER INTERFACE AND LANGUAGE REQUIREMENTS

In any interactive automation system there must exist a means for man to direct and control the automation activity to respond to his needs. This man/machine interface can be divided into two areas, the activation or start-up of the computer's resources and the instructions directing the application of these resources. The first area deals with connecting to the remote computer and turning on the AIRS system under the control of the computer's "monitor" (operating system). The second area deals with operations within the control of the AIRS system. In essence this second area is the control language of AIRS. There exists a wide range of freedom in designing the AIRS language. The simplest approach might employ a set of buttons. Each button tells AIRS to do a specific task. In the longest approach the program might ask an extensive string of questions designed to exactly elicit what the desired task was. There are arguments for and against both extremes and variations in between. The first section of this chapter discusses the mix of language approaches employed by AIRS and the reasons behind the mix. The second section addresses the problems of the man in dealing with the time-shared computer and also discusses the relationship of these "monitor" interactions with the AIRS system.

4.1 AIRS LANGUAGE MIX

In addition to the balance between data required by the automaticn system and data produced by it, the success or failure of the automation system is also greatly dependent upon ease of operation and control. Since the system is designed to be used and operated by ATC flow controllers, it is imperative that the interface with the computer and the request language be uncomplicated and easy to use. In the ultimate, everything should be as simple as the pushing of a special function button and presto, the desired data is reported. Well, this is certainly not possible in the prototype system. The scope of data would require an excessive number of function buttons and the available time-sharing computer systems are not capable of supporting, without considerable cost and special equipment development, a scheme of function button control. The closest approximation to push button control is the normal teletype keyboard used in conjunction with time-sharing computers. A specific function can be attributed to a single character in the keyboard. Additional functions than the number of keys on the keyboard can be easily handled by combining characters (i.e. encoding the function). The fewer number of characters composing a unique code word, the quicker the request can be input. The more functions desired, the greater the number of code words and the more difficult the memorization requirements for the operator. Obviously, mnemonic codes which have clear meaning to the operators are best. If one combines mnemonic codes for further specification of the requested data, the input becomes a string of codes. Ease of use requires that it be natural to assemble the string of codes in forming the total request. The desire for easy request assembly leads to the consideration of using a language approach which approximates controllers' jargon. The language would not only incorporate such mnemonic codes but would also permit phrases and context meanings of these codes in the request stream. The language should include freedom in the order of many of the words in each request and use of punctuation. The AIRS control language does in fact contain the full benefit of a language natural to the controllers and the conciseness of encoding.

Contrasting with this approach of coded requests is the question and answer type of automation control. For complex sets of retrieval and processing choices, it is often best to lead the operator through request formation in a stepwise dialog. This diminishes the demand upon the operator for a compound entry of complete and exact specifications from memory. The wide variety of operations in AIRS necessitates a mix between the jargon language and dialog control to best

service the controllers.

There is yet a third approach to automation control and that is the fixed format request entry. For static forms of repetitive operations such as ARO data updates, neither dialog nor jargon language inputs is essential. The price of fixed format is its own rigidity; the fixed format can not tolerate any deviations from the standard and only a limited assortment of standards is practical in most situations. This third approach was utilized in the original AFCP and ARO automations. Incorporating this same approach into the AIRS automation system would minimize the need for user retraining to continue these operations. This was the primary factor in the application of fixed format request entry for the AIRS automation of the Airport Reservation Office's data updating.

In summary, AIRS required all three approaches to controller interface in its language specification. Details of the AIRS user's language can be found in reference 7. Table 4-1 summarizes the various types of request operations available in the AIRS user's language.

One more area of discussion in the design of a user's language concerns the demands upon the users for completeness of request specification. Some of the burden of entering complete specifications can be handled by the computer. The request language can be designed to accept partial inputs, determine the appropriate default conditions and construct the complete request for processing. For example, most data requests concern the current day. For user efficiency, entry of the date in a request should only be required for other than today. The computer can insert today's date when none is entered.

TABLE 4-1 SUMMARY OF AIRS REQUEST OPERATIONS

Type of Operation	Request Selectivity
Traffic Load Summary	*Places of origin and/or destination: airports, ARTCC's, Areas (e.g. ZEUR) *Arrival and/or departure loads *Time period of interest *Date (or day) of interest *Airlines of interest *Aircraft types *Flight durations of interest *By scheduled, controlled or departure delayed times
Listings of individual flights	*Time period of interest *Sorted as desired *Columns desired (e.g. flight identification, origin departure time)
Plots of summary data	*Time period of interest *Graph data of interest (e.g. landing capacities, arrival loads, holding stacks, air delays)
Predicting arrival delays	*Destination airport *Time period of interest *Initial stack size and occurrence time
Entering Landing Capacities	*Airport of interest *Time period of interest *Normal (permanent) or today only capacities
Entering General Aviation Factors	*Airport of interest *Normal or today only
Entering Departure Delays	*Airport of interest *Time period of interest
Look up Airport Characteristics	*Airport of interest *Landing Capacities *General aviation factor *Departure delays *Time period of interest

TABLE 4-1 (continued)

Flow Control Procedures	*Airport of concern *AFCP or Quota Flow *Edit control parameters and/or zone structures *Output controlled and/or original scheduled traffic levels or delay predictions *Initial stack size and occurrence time *Overriding start and termination times for flow control period
Airport Reservation Entries	*Enter new flight schedule *Cancel flight schedule *Paper tape or interactive entry mode
Information Exchange	*Mail from TSC *Comments from users (i.e. SCC) *Format explanations

4.2 TIME-SHARING COMPUTER INTERFACE

One of the major decisions discussed in Chapter 2 stated that the operation of AIRS be such as to minimize computer training requirements for the users. Minimizing computer training simply means reducing the number of operating system level activities for the user. Since time-shared systems are multi-user in nature, it is unreasonable to expect that the operating system be tailored to suit one type of user at the expense of others. The burden of handling the operating system operations must therefore be shifted to the AIRS program as far as possible.

In most automation systems, accidental mistypes during inputs cause fatal errors and control returns to the operating system level from the automation system. Such errors mean that a restart of the automation system is required. This type of input error cannot be prevented, but it can be accommodated by a "forgiving" automation system which screens its own errors and decides what actions follow detected errors. It retains control and avoids unnecessary aborts to the operating system level, thus simplifying the "what to do now" training.

Many automation systems are composed of an assortment of completely separate programs (subsystems). One type of report might be produced by one program, a different variety might be produced by another. Each program must be started up by commands to the operating system; each requires its associated training. AIRS avoids this separate program approach and its extra training burden by integrating all of the automation programs into one multi-functional system.

Occurrence of data file errors, sometimes caused by computer malfunctions, often results in further user interaction. When a file error is detected in a typical automation approach, the system aborts what it was processing and returns to the operating system level. Attempts of the user to restart the automation system will abort at the same point. Training is not going to help here because it requires programmer attention. The system is out until fixed. This situation is prevented in the AIRS system for the most common file damage situations. The user is not left with an error message and an unusable file but instead is given a message that the detected file damage has been fixed and an instruction to restart AIRS.

There is another common situation of file damage produced when the computer malfunctions. It is really not damage in the usual sense but is the inconsistency of a partially updated file. Special training in backing up data

files could restore the file to a working level, but this is at a programmer training level. One would also lose the entries made that day. It is far better to make the AIRS system go to extra lengths to provide recovery data automatically with all critical file updates and to have AIRS test files before using them, to see if they must be restored first. This is indeed the designed capability in AIRS. AIRS can restore the integrity of any partially updated file with at worst, the only loss being the last item of data at the time of the computer malfunction.

As discussed in the previous section, the teletype or display device permits the users to interact with the computer through the keyboard. There is also another interface feature associated with some of these teletype devices, the paper tape unit. It is possible for the user to prepare paper tapes by keyboard entry when not connected to the computer. The user can then connect the teletype to the computer and enter the paper tape in place of keyboard inputs. For an operation such as flight data updates, this paper tape mode would permit considerable freedom in scheduling the typing and the data entry work cycle to meet practical operational conditions. This interface option for ARO operations has been incorporated into AIRS.

In summary, AIRS has been designed to give the user a second chance with input errors. It presents to the user a general set of control options to activate vastly different automation tasks ranging from listings to quota flow control assistance. It maintains an operational level of file integrity by recovering from partial file update damage and repairs the most common form of system damaged files. And it provides the freedom of paper tape buffering. All of these extended automation capabilities add up to the minimization of computer training and ease in operation for the ATC staff at the SCC.

5. CENTRALIZED DATA BASE

The domain of centralized data banks has been traversed in many applications from libraries to computerized information systems. There is a wealth of technology to draw upon; a great portion is independent of whether automated or manual, some is only practical with computers. One can see in the library excellent examples of efficient retrieval. A person can use the specially prepared index and cross reference files to identify the book of interest, he then uses the code number shown on the book's reference card to go to the book rack, shelf level and cubical which contains the book. A short serial search of the cubical easily locates the desired book. One can see in this example the combination of techniques which aid speedy retrieval. First the use of index or cross reference files which insure rapid access to any part of the data bank. Other techniques used involve grouping or categorizing (e.g. subject and author index files), codes which point to the area of location and sorts such as alphabetized ordering. The actual process of retrieval involves serial searching, binary searching (e.g. in a sorted card file one can narrow down the area of search by spanning successively smaller numbers of cards approaching the desired card) and random accessing.

Looking into computerized information retrieval systems one sees other applicable technology. The computer system adds a new dimension to accessing by introducing hash coding, the process of using the name of the desired item in a transformation equation to identify the storage location (address) of the item. Computers also allow blocks of data to be scattered throughout an area of storage yet be serially retrieved by a chaining technique which links each block to another.

The problem of designing a central data base is the development of the best mix of these techniques to produce the most efficient file structures to fulfill the design objectives. The AIRS design objectives which influence the data base structure are:

1. Provide fastest access and data retrieval possible within the time-sharing computer restrictions.

2. Provide efficient updating capability but not at the sacrifice of retrieval speed.

3. Provide full range of operations for all airports in the data base.

The design tradeoffs must also consider the time-sharing computer's external storage (disc) capacities, access and transfer rates, and the share of service expected during normal computer loading. Add to this the share of internal storage (core) and buffering capacities for transferring data to and from the storage devices and the problem becomes more complicated.

This chapter presents the AIRS tradeoffs, the selected goals and the effects of computer system restrictions in the design and structure of its centralized data bank. The discussicn will cover the files used during the operation of AIRS. Discussed in later chapters will be three closely related areas: the support files, the damage recovery aspects of each applicable file and the non-centralized (temporary) files. This chapter is subdivided into four sections. The first addresses the major file, the flight schedules, and its associated index files. The second section treats the file of the airport parameters vital to traffic predictions. The next section discusses the flow control files needed for application of the AFCP and Quota Flow Procedures. And the last describes the other data files required by AIRS.

5.1 FLIGHT SCHEDULES

This section describes the main data base of AIRS - the flight schedules. We will explain what information it contains, how the information is represented, how it is organized, how it is accessed, and how it is updated.

CONTENT

The flight schedules contain:

1. All scheduled flights (except helicopters) originating in or departing from the U.S. A "flight" is actually a non-stop flight leg. These scheduled flights are obtained monthly from Official Airline Guide data.
2. Unscheduled flights (extra sections, general aviation, etc.) entered by the ARO for major airports. The fact that the ARO normally only makes entries for certain airports is due to FAA operating procedures, not to restrictions in AIRS. There are only a few airports for which the FAA requires reservations. AIRS treats all airports alike, however, and is equally happy to accept entries for any one. In fact, the ARO does enter extra sections at holiday times for many more than their standard reservation airports.
3. The ARO also cancels flights for the major airports (as above, there is no restriction within AIRS to particular airports). Any flight can be cancelled, regardless of how it got into the data base (whether from monthly schedules or ARO entries).

The monthly schedules contain about 25,000 flights, and the data base usually contains two months worth to provide continuity. ARO entries add about 10,000 a month, bringing the typical data base size to about 60,000 flights.

A "flight" in the data base consists of the following basic data:

1. Flight ID (usually an airline and flight number)
2. Origin airport
3. Destination airport
4. Planned time of departure (in GMT)
5. Planned time of arrival (in GMT)
6. Effective date
7. Discontinued date
8. Days of the week on which it departs
9. Days of the week on which it arrives
10. Aircraft type (equipment type)

11. User class (whether scheduled air carrier, general aviation, taxi, extra section)

It may also contain temporary information for the current day giving different departure and arrival times as a result of SCC-entered departure delays or flow control restrictions.

FORM

All flight schedules are stored on a single disk file. In earlier systems, such as the pilot terminal model (Ref. 2) and the original AFCP system (Refs. 4 and 5), the schedules for each airport were on separate files, making it simple to deal with the flights for each airport. However, these systems only handled a few airports -- 14 in the pilot model and 5 in the AFCP program. One of the decisions in designing AIRS was to avoid making any advance assumptions as to what airports would be of interest to flow controllers, and thus we chose to make information on all airports accessible. AIRS has in its data base over 1000 airports, and it would not be feasible to keep a separate file for each.

Flight schedules on this file are stored in a standard format. Each flight takes up the same number of words, which include space for all the data the flight can contain (see above). Since all flights have room for the temporary delay data, flow control procedures or other delays can be applied to any airport. Although this delay data is seldom used, allocating space for it in the standard format doesn't require much space and simplifies processing. If the flight format only contained the basic flight data, how would temporary delays be handled? One method would be to put the temporary data somewhere else (perhaps on a separate file) and have a pointer to this data in the basic flight entry. This would make later processing inefficient, since to decide anything about a flight we would have to follow its pointer to get the necessary data; jumping around during disk access is much slower than reading sequentially from one area. Another method would be to move the flight entry to the end of the file, where there's room, and append the added data to it. This would complicate both access and processing, since entries would have different lengths and formats. Also, since flights are accessed via indexes (as we will see later), moving a flight to a new location requires changing the indexes which point to it. Since delays are imposed on many flights at a time, it would take a lot of work to do this re-indexing.

Minimizing the number of words used per flight can improve both the abilities of AIRS and its response time. Some data processing in AIRS, such as sorting flights for listing and computing flow control restrictions, requires working with large numbers of flights at once. But lack of internal (in core) storage space limits the number that can be worked with simultaneously. Obviously, the fewer words per flight the more flights that can be stored internally. A more important gain is in response time, since disk access is the principal cause of slow response. Fewer words per flight means fewer disk accesses to read the same flights, hence shcrter run times. The gains are not only in reading the schedules, but also in writing and reading scratch files used to store flights temporarily during processing.

Two techniques are used to reduce the size of a flight entry - packing and encoding. Most of the data in a flight entry do not require a full 36-bit word. We thus pack as many items into each word as will fit. The PDP-10 machine language includes instructions which make it easy to manipulate (store and retrieve) arbitrary "bytes" consisting of any number of bits at any position in a word. Further compaction is achieved by encoding some of the data so that it requires less space. For example, representing an airport by a number instead of by its three-character name cuts the amount of space it requires in half.

ACCESS

Flight retrieval would be excessively slow if we had to sequentially search the entire schedule file. The sequential reading alone (ignoring any processing done on the flights) would take several minutes. Random access is available, but in order to make use of it we must know where we want to access. We can find the desired flights more efficiently by indexing the flights according to data (keys) by which they will be retrieved. For example, they could be indexed by origin, airline, planned time of arrival, etc. Then to retrieve flights satisfying some indexed condition, we would look up that condition in the index and search just the flights listed there. The schedules can be indexed by any number of keys. Then if a request specified more than one indexed condition, we would look up both and combine the information from the two to get our list of flights to retrieve. This might narrow down the search still further. For example, let's assume the schedules were indexed by origin and destination. If a request were for JFK departures to Miami, we would look up JFK in the origin index and MIA in the destination index, then search only the flights which appeared on both lists.

It might seem advantageous to index the schedules by all possible keys - origin, destination, airline, planned time of departure, planned time of arrival, aircraft type, etc. Then almcst any condition appearing in a request could be used, via the associated index, to reduce the schedule search. By combining the information from all these indexes, we should get a maximum narrowing down of the search, since we would look up only the flights satisfying all the indexed conditions. However, there are two reasons not to go overboard on the indexing, but rather to judiciously choose the best keys to index by. First of all, the work required to manipulate the indexes may outweigh the advantage they are designed to achieve. Merging indexes to get a narrowed down list is in itself a time-consuming process. Since the indexes are permanent, they will be on disk files; thus accessing many indexes means doing a lot of disk accesses. Also, some indexes would be very long, requiring many accesses (for example, in an hour-of-departure index, each hour would list about 2000 flights). Secondly, indexes add work to data base updating caused by ARO flight entries. If flights are to be accessed only via indexes, then in order to be accessible, any flights added to the schedule file must also be added to all appropriate indexes. Obviously, the more indexes we use, the more work it will take to add a flight, and the slower schedule updates will be. Although retrieval speed is more important than update speed, updating must not be made so slow as to be impractical. We must therefore choose only a few keys to index by.

There are three major factors in choosing a key to index by:

1. It should be frequently used to access flights; otherwise we're carrying the overhead of updating the index but using it infrequently.
2. It should provide a good narrowing down of the search, otherwise we're doing the work of manipulating the index for very little benefit.
3. The indexing data should be constant; otherwise a request which changed the flight data would also require the index to be changed (the flight would no longer be listed in the right place). This increases the workload of making a change to the schedules.

We chose to index the schedules by origin and destination airports. This choice satisfies the three factors above.

1. Since AIRS is airport-oriented, all requests for traffic information must specify an airport of origin or destination. (Actually, a center or defined area may be used instead. This will be discussed below.)
2. Since there are over 1000 airports in the AIRS data base, indexing by airport narrows down the file considerably.

3. No requests in AIRS result in changing a flight's origin or destination.

Another retrieval condition required in every request (though it may be implicit) is a time period. However, arrival and departure times are not useful to index by since they violate factors 2 and 3 above. Since most requests are for at least half a day, time periods provide little or no narrowing of the search. They are also non-constant, since imposition of flow controls or departure delays may temporarily affect them. There is no point in indexing by factors such as aircraft type which are rarely used in retrieval.

We stated above that every request for traffic must specify an airport. Actually, it must specify a place, which can be an airport, a center (ARTCC), or a defined area (such as Europe). Centers are defined as groups of airports and areas as groups of airports and/or centers. AIRS originally translated groups such as centers into their constituents. Thus every request did have (implicitly) airports in it. The indexes for these airports were looked up and combined to get a resultant list just as if the airport names had been in the original request instead of the center name. However, since centers contained many airports and often several centers appeared in the same request, the index manipulations for these requests turned out to be so slow as to give unacceptably poor response. We thus index the schedules by origin and destination centers as well as by airports. Areas are still handled by substituting the constituents and combining the indexes. It makes sense to index by center, since center definitions are permanent. Instead of doing the same time-consuming airport combinations each time the center is used, it is economical to just do them once, when forming the index.

ORGANIZATION

Although the indexes enable us to access only the relevant parts of the schedule file, the disk access is still much less efficient than it could be. Even when randomly accessing specific words of a file, the entire disk block containing the words is transferred into a buffer. If the next words requested are from the same disk block, the system recognizes that they are already in its buffer and doesn't access the disk again. Thus the disk is accessed only when the words to be read are in a different block from the words last read. Our access would therefore be more efficient if the flights we wanted were grouped together. We thus sort the schedule file by airport (since it is indexed by airport). Actually, it is sorted by destination

and subsorted by origin. This means that access by destination is more efficient than access by origin, since the flights for a destination are together while the flights for an origin are scattered among various destinations. We made destination retrieval more efficient since most flow control requests (for arrival delay predictions, flow control procedures, etc.) are concerned with arrivals. We could have made both types of retrieval equally efficient by having two copies of the schedules, one sorted each way. This is unfeasible, however, because:

1. The data base is so large that two copies would strain the available storage;
2. ARO entries would have to be made on both copies and added to both sets of indexes, doubling the amount of work;
3. Data changes (such as delays) would have to be entered on both copies.

The sorting is done once a month when the new Airline Guide schedules are obtained. Entries made by the ARO throughout the month are not in the sorted order. ARO entries will be discussed later in this section.

CROSS-REFERENCES

How do you look up an airport or center in a cross-reference (index), and what do you find when you do? There are two cross-reference files, one for airports and one for centers. Each airport (and center - everything we say about the airport indexes also holds true for the centers) has associated with it two lists - one list of pointers to flights originating there, and one of pointers to flights destined for it. The pointers are the relative positions of the flights in the schedule file rather than their absolute addresses. That is, a pointer value of 17 means the 17th flight, not the flight at word 17. The use of relative pointers is possible because the schedule file is fixed format, so given a relative position it is easy to calculate the corresponding address. It is desirable to use relative rather than absolute pointers because it is then possible to reformat the schedule file, changing the number of words per flight, without affecting the cross-references. Such reformatting has indeed been necessary several times during the evolution of AIRS. Since the flight schedules (except for ARO entries, a small percentage) are sorted by airport, the cross-references will have many consecutive pointers. We thus reduce their size by storing pointer ranges rather than individual pointers. For example, instead of listing the numbers 23 through 30, we would just store the pair: 23,30.

Since the pointer lists vary in size, there is no inherent way to know where to find the list for any particular airport. We thus have a fixed format index on the start of the cross-reference file which gives, for each airport, the address and length of each of its two (origin and destination) pointer lists. The same number used to represent the airport on the schedule file is used to calculate the address of the airport's information in the fixed index. (AIRS translates from the airport names to these numbers via the dictionary, discussed in 8.1.) Centers, though not stored on the flight schedules, are also assigned numeric values (via the dictionary) which are used to calculate their positions in the fixed index of the center cross-reference file.

The lists of pointer ranges are kept sorted in increasing order. It is good to have them sorted when they are being used, since:

> If they are being used directly to retrieve flights, there will be just a single pass through the schedule file, thus getting the advantages of sequential access (discussed earlier);

and

> If they are being combined to take care of multiple request conditions, the combination operations are much simpler and more efficient with sorted lists (see 9.1).

It is obviously more efficient to store them sorted in the first place than to sort them each time they are used. It would be difficult to sort them at use time anyway, since the lists may be arbitrarily long and may not fit in core.

ARO ENTRIES

The new flight schedules typed into AIRS by the ARO are added to the central schedule file and, in order to be accessible, to the cross-references. How is this accomplished?

New flights are appended to the end of the schedule file; no attempt is made to arrange them. We saw earlier that the Airline Guide flight schedules are sorted by airport in order to speed access to them. It isn't possible to maintain this sort when adding flights, since to put them in the area corresponding to their origin/destination combination would require pushing all subsequent flights down to make room -- a gigantic undertaking when you're talking about tens of thousands of flights. Since AIRS isn't dependent on the sort, we neen't consider such extremes, but can simply put the flights on the end of the file in the order in which they are entered. If the ARO groups the flights by airport when entering them, better

access times are maintained than if they allow them to be fragmented.

In order to be accessible, a new flight must be added to the cross-references for its origin and destination airports and centers (if any). Since the new flight is at the end of the file, its relative position is greater than that of any other flight, so adding its pointer to the end of a cross-reference list automatically maintains the sorted order of the list. Cross-reference lists do not initially have empty space on their ends for adding pointers. Expansion space can't be provided in advance, since it is not known what airports will need expansion or how much space they will need. Instead, when a list needs room to expand, it is copied onto another area of the file which has enough space, and allocated some extra space. If enough additions are made to use up this space the list is moved again. Since the pointers are stored in ranges, such that any sequence of consecutive flights requires only two pointer words, we see another advantage of grouping the flights before entering them, namely that the cross-references will grow less, thus being easier to manipulate and requiring less frequent moving around.

The space vacated by a cross-reference list that has been moved shcuild be reusable by other lists that need space. Each cross-reference file thus has associated with it a free-space file, which keeps track of space available on the cross-reference file. Areas that have been vacated are recorded on this file, and areas to move to are chosen by looking on this file. Of course, if no adequate space is available, the list is moved to the end of the cross-reference file, extending it.

SUMMARY

Figure 5-1 shows the structure and interrelationships of the schedule file, the cross-reference files, and the free-space files. Two of the items shown have not been discussed yet:

1. Recovery data:
   During the processing of an ARO entry or cancellation, there are times when the data base is vulnerable; if the program were interrupted (as by a computer crash) at certain instants, the files might be left in an inconsistent and possibly unusable state. AIRS thus stores enough data before performing the dangerous file modifications so that it can recover after an interruption and restore the integrity of the files. The nature of this recovery data and procedure will be discussed in detail later (13.6).

2. Expansion room:
   In order to avoid a certain type of damage, the files are kept physically longer than is needed for their data. Thus modifications that append to the files are appending logically (to the data), not physically (to the file). This will be explained in 13.2.

To summarize the file contents,
   The schedule file contains fixed format flight schedules.
   The terminal cross-reference file contains
      variable-length lists of pointers to flights (on the schedule file) for each origin and destination terminal (airport). It also contains a fixed format index, addressed by terminal, to locate these lists.
   The terminal free-space file keeps track of the end of the terminal cross-reference file (so it can be appended to) and free areas (location and number of words) on the terminal cross-reference file.
   The center cross-reference file is analogous to the terminal cross-reference file, but for ARTCCs.
   The center free-space file is analogous to the terminal free-space file.

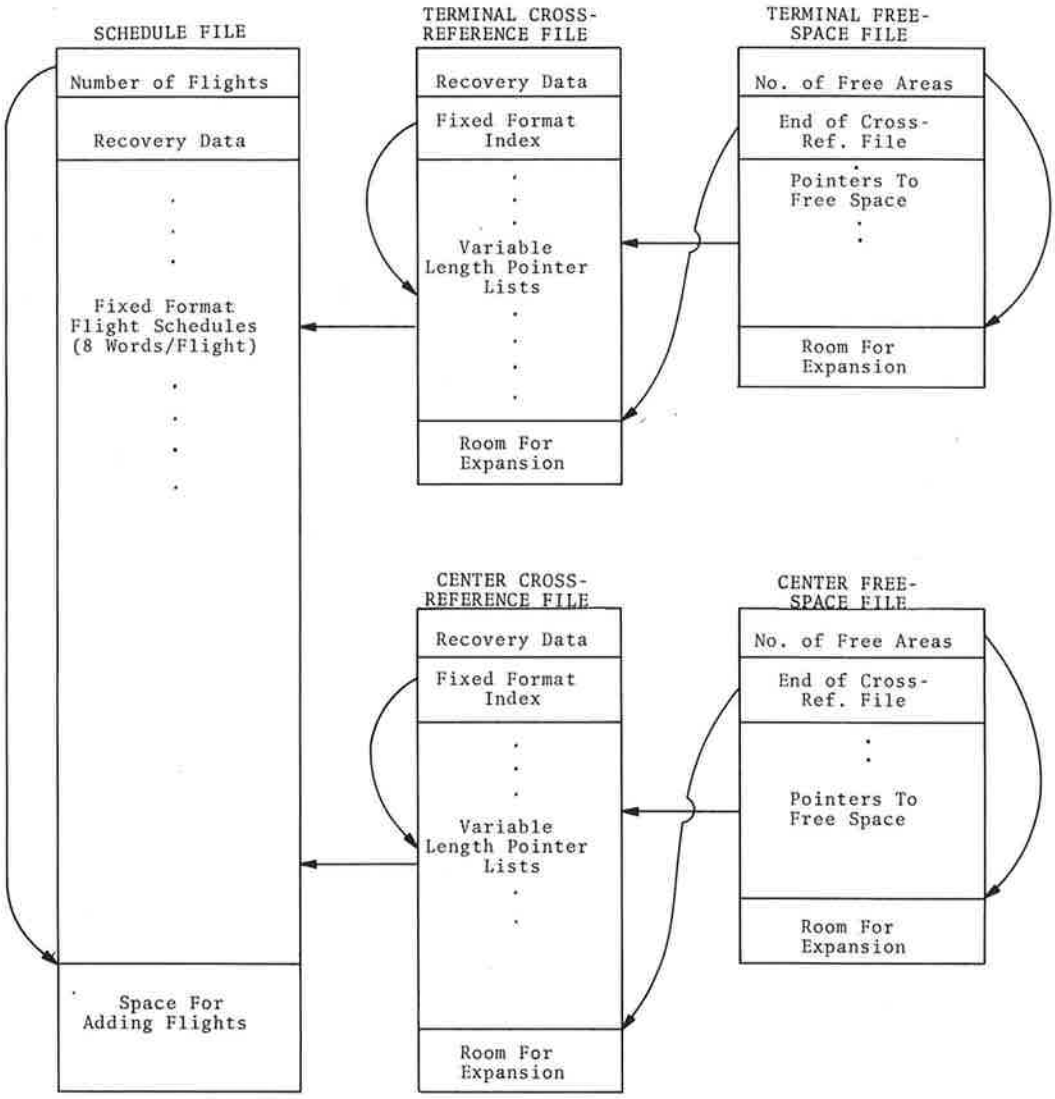


FIGURE 5-1

FLIGHT SCHEDULES AND RELATED FILES

5.2 AIRPORT DATA

In addition to flight schedules, AIRS must have certain data on airports. This section describes the airport data file, which contains some of this data. The rest will be discussed in section 5.3.

Four types of information are stored on the airport data file:

1. ARTCC
   In order to make flow control allocations, AIRS must be able to find out what center an airport is in. Every domestic airport has an associated center.

2. General Aviation Factor
   Unscheduled traffic (general aviation, etc.) must be taken into account in order to make meaningful predictions. A factor, a percentage to apply to scheduled traffic, is used to estimate unscheduled traffic. The factor, which can be entered or modified by the user for any airport, is a single value which applies at all times on all days for that airport. A temporary value can also be entered to apply for the current day only.

3. Landing Capacities
   Landing capacities are necessary in order to do arrival delay predictions or flow control. Landing rates, by hour, can be entered or modified by the user for any airport. The same landing rates apply regardless of day of the week, but a temporary set of rates can be entered to apply for the current day only.

4. Departure Delays
   Departure delays can be applied by airport by hour of the day and apply to the current day only.

Further discussions of the meaning and use of this information will be found in chapters 9 through 12.

The simplest file structure would be to have a standard block set up for each airport to hold all of the above data. The data for a particular airport could easily be accessed by using the numeric value associated with the airport to calculate the address of its data block (as was done in section 5.1 for the cross-reference files). However, this would be extremely wasteful of storage space; although every airport has a center, very few will ever have general aviation factors, landing rates, or departure delays entered for them. We therefore have small standard blocks for all

airports, accessed via the airport's number, containing only the center and a pointer (address) to a larger block containing the rest of the information, if any. The longer data block is only created if and when data is entered for the airport (see chapter 11.1), and contains space for all the data that can be entered. Storage is also conserved (hence access times reduced) by packing the data. Both general aviation factors (today and normal) are stored in a single wcrd, and all the data for each hour (today and normal landing rates, departure delay) are packed into a single word.

The file also contains information on any temporary data entered and when it applies, so that AIRS can reset it automatically the next day (see chapter 11.1). Figure 5-2 schematically shows the airport data file.

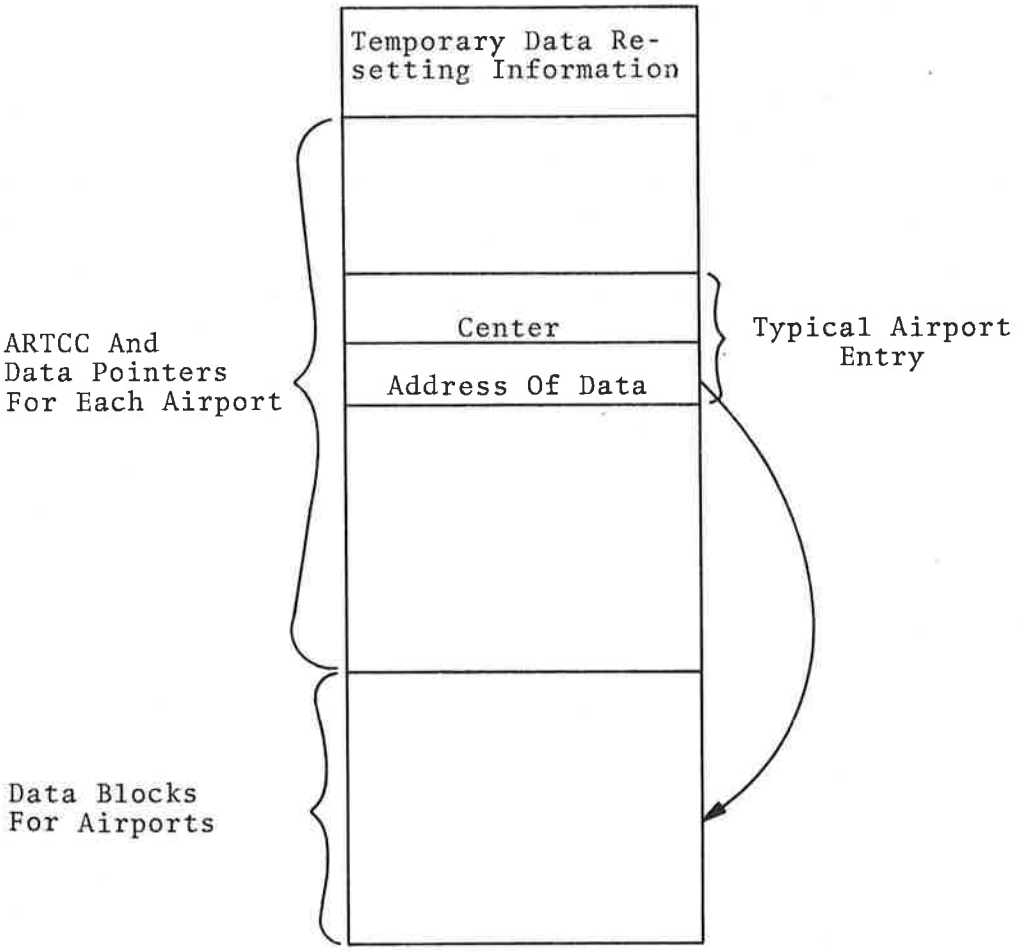


FIGURE 5-2

AIRPORT DATA FILE

5.3 FLOW CONTROL DATA

The incorporation of flow control procedures into the AIRS system requires that certain control and traffic zoning data be maintained in the centralized data bank. Because flow control procedures can be tailored to each airport of interest, the supporting data files must be maintained by airport. These files could have been combined into one file as organized in the previous section, 5.2. However, the separate approach was selected to facilitate multiple flow control procedures for a single airport. There are two types of flow control files possible for each airport, the first contains the control information and the second contains the zone structure. Except for a single default control file, these files are created and maintained by the CFCF in accordance with the flow control needs (see Chapter 11.2). It is therefore possible for hundreds of these files to exist in the central data bank as flow control procedures encompass more airports. Currently, fewer than a dozen airports are involved. The purpose of the default control file is to initialize the control parameters when the CFCF is developing the flow control procedures for another airport.

The control files contain the following information for the applicable airport:

1. The maximum air delay permitted during flow control periods.

2. The maximum stack (airborne aircraft holding in airport's area) permitted during flow control periods.

3. The interval of time used in computing the allocations and quota flow rates.

4. The choice of whether or not to incorporate the general aviation traffic using the GA factor and, if used, during which periods of time to employ it (i.e. for airports with ARO updates, the GA factor need not apply during the period in which ARO operations apply).

5. The choice concerning the applicability of flow control procedures to airborne domestic and/or foreign traffic.

6. The margins for take-off and landing times, to be used in determining aircraft flight status (e.g. an aircraft might be considered ineligible for flow

control ground delay when it is ten minutes from take-off instead of on take-off).

The zone files contain airport grouping information which is used in formulating the flow control allocations. A zone file has the following data:

1. The number of zones in the file.

2. The control status of each zone.

3. A list of departure centers and airports comprising each zone (i.e. the places of flight origin).

4. Within each zone, titled subsets of airport groupings for tabulation of allocations or quotas for that zone.

The structures of these two types of files are simple. The amount and range of information within each is very limited. The control file is a fixed list of control parameters. The zone file is a mix of fixed data blocks and one general area of storage used for the place lists. This general area is accessed by using a pointer and number (of entries) found in the fixed data block associated with the zone number (zones are referenced by number). The use of the files simply involves reading and storing in core all of the information they contain.

5.4 OTHER DATA FILES

AIRS has three other central data files containing tables which enable it to know about days and dates and to translate back and forth between words in the user language and their internal representations. These are as follows:

DATE FILE

This file contains the names of the months, and associates with each month the number of days in it, the first day in it, and the year in which the information is true. AIRS automatically keeps this information up to date. AIRS simply reads this file when it starts up and holds the data in an array in core. The table is used for:

- translating from the internal representation of a month (a number from 1 to 12) to its name for use in output - the number is used simply to index the name in the array
- figuring out the day of the week a given date falls on, and vice versa
- calculating a date having some relationship to a given date (e.g. calculate tomorrow's date)

The manipulation of dates and the updating of this file are discussed further in chapter 16.

TRANSLATION FILE

This file contains four tables: the dictionary, the airport table, the aircraft type table, and the center table.

DICTIONARY: This table is described fully in section 8.1. Briefly, it is a hash table in which each word in the AIRS language is defined, providing the translation from user language to internal representation. For example, looking up an airport name gives the number used to calculate the airport's location in the cross-reference file (5.1) and in the airport data file (5.2). Actually, the dictionary on this file does not contain all the entries listed in table 8-1. It only contains airports, centers, airlines, aircraft types, areas (groups of places), aircraft type groups, and airline groups. When AIRS is run, it reads in the dictionary and copies it onto a scratch file. It then enters the rest of the items on this copy. This is necessary because the values of some types of words contain addresses within the AIRS program. A change in AIRS can thus change these values.

By setting up a private dictionary and entering these values in it at run time, each copy of AIRS makes sure it has a correct (for itself) dictionary. The dictionary (on the scratch file) is read by randomly accessing just the words desired.

AIRPORT TABLE: This is just a list of airport codes (such as JFK). The number used to represent an airport (on the schedule file, and for all internal use) serves as an index to its name in this table. The table is simply read into an array in core, where it is used to look up airport names for output.

AIRCRAFT TYPE TABLE: This is a list of aircraft type codes, indexed by the numbers used internally (and on the schedule file) to represent aircraft types. Like the airport table, it is read into an array in core and used to get codes for output.

CENTER TABLE: This is a list of center (ARTCC) codes, indexed by the numbers used to represent centers. It is read into an array in core and used to look up center names for output.

GROUP DEFINITION FILE

There are words in the AIRS language standing for groups of aircraft types, airlines, and places. AIRS handles these by substituting for them the members of the group. The group definition table has, for each group: the group name, followed by the number of members in the group, followed by the members of the group. These definitions are packed end-to-end; the definition of the group name in the dictionary supplies the index of the group's definition in the table. The group table is read into an array in core whenever a request containing a group is processed. It is not kept in core throughout the run because it is space-consuming and infrequently needed.

Some of the programs which support AIRS also make use of the date and translation files (see chapter 18).

6. AIRS PROGRAM OVERVIEW

Chapter 4 described the ways the user communicates with AIRS (via a combination of dialog, fixed format requests, and a free-form request language), and table 4-1 summarized the activities AIRS can be asked to perform. Figure 6-1 shows the overall flow of AIRS. After AIRS is started up and has initialized its files and data, it enters the basic cycle of accepting a free-form request from the user, analyzing it, and performing the requested activity. The cycle continues until the user requests that AIRS quit. Two of the activities have extensive user control cycles of their own: These are diagrammed in figure 6-2. The flow control activity has two dialog (question-and-answer) cycles: one in which the user edits zones and control parameters, and one in which he controls the flow control reports and activities. The normal AIRS request cycle is not resumed until the user so specifies. The airport reservation activity goes through a request cycle similar to the overall AIRS cycle. It accepts fixed format requests, analyzes them, and performs the requested activities. The ARO cycle continues until the user requests that AIRS quit or that control return to the normal AIRS cycle.

Of course there are other user interactions and control flows not shown in 6-1 and 6-2, but these show the major loops of AIRS activity control. The various AIRS activities will be discussed in the following chapters.

Initialization

Descriptions of the initialization operations are scattered throughout this document. These include:

- Typing message, if any (15.1) and announcing existence of mail (15.2);
- Reading and updating, if necessary, date information (5.4, 16);
- Reading translation tables and forming a run-time dictionary (5.4);
- Resetting temporary data in centralized files if necessary (11.1).

Request Analysis and AIRS Activities

Chapter 8 explains request analysis (for both the main AIRS cycle and the ARO cycle) and for each type of request, tells what chapters explain the associated processing (these chapter references are in 8.3 under "Phrase Processing" and at the end of 8.4). For several activities, however, these referenced chapters are not really the whole story. Chapter 7 describes how certain activities which need flight data invoke the retrieval activity by making an internal (i.e.

generated by and within AIRS) traffic load request. Thus in these cases, the formulation and processing (through request analysis, chapter 8, and retrieval, chapter 9) of this internal request precede the main data processing operations described for the activities.

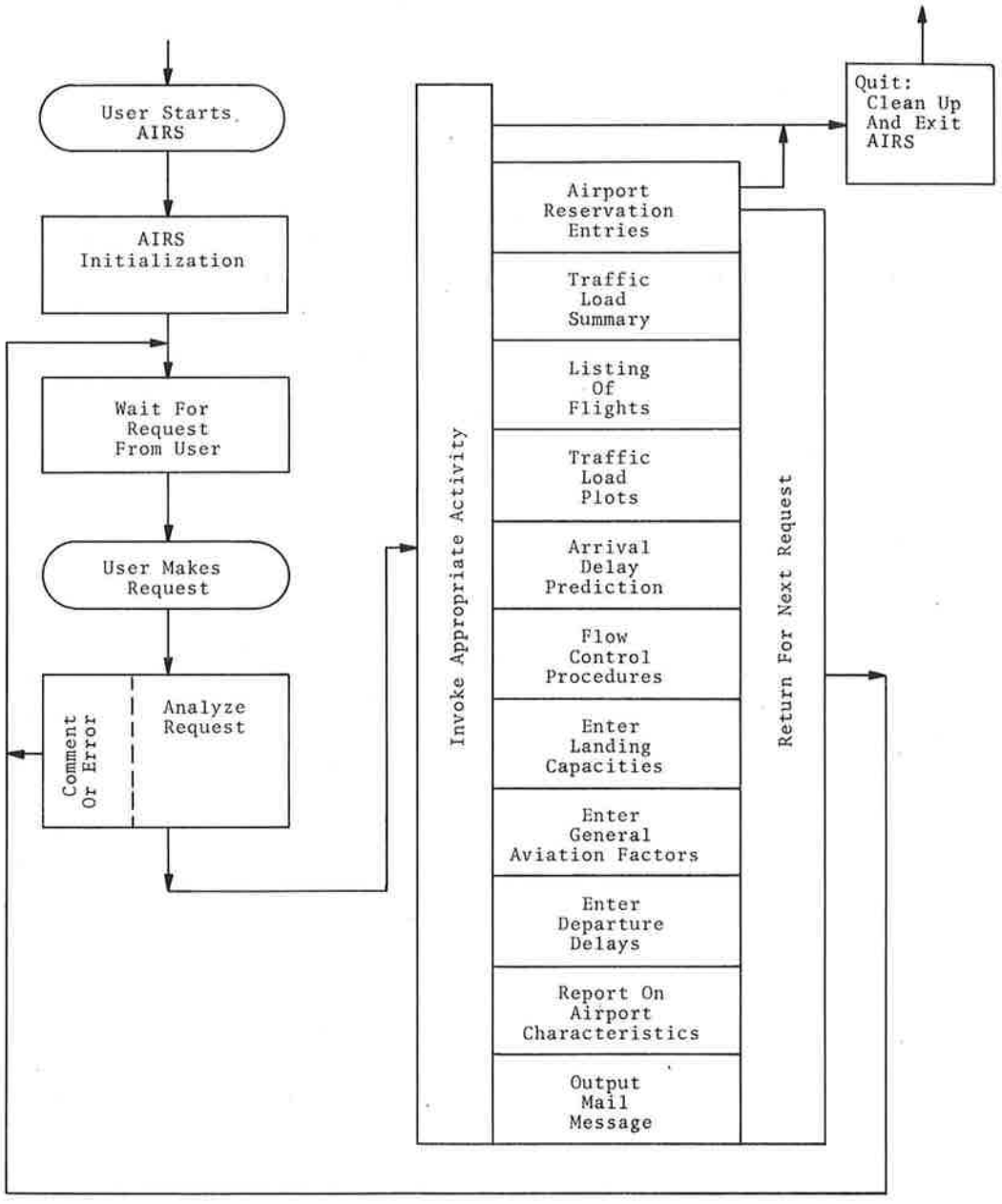


FIGURE 6-1

OVERALL AIRS REQUEST FLOW

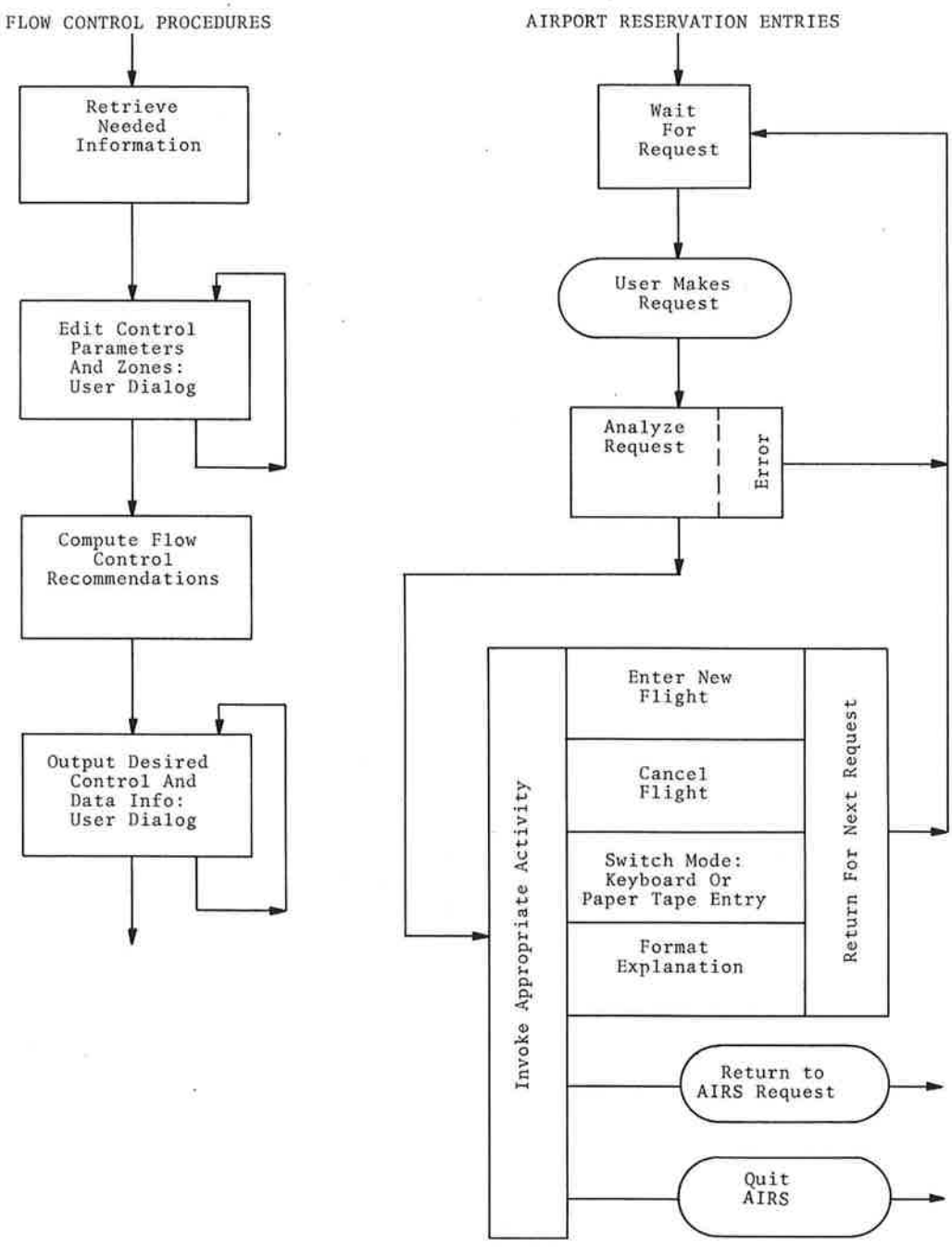


FIGURE 6-2

AIRS ACTIVITY REQUEST FLOWS

7. REQUEST FORMULATION

The kinds of requests that can be made to AIRS are summarized in table 4-1, and the "words" used in formulating them are summarized in table 8-1. (For a complete description of the AIRS request language, see the AIRS User's Guide, Ref.7.) The overall cycle of AIRS operation is:

1. Informs the user it's ready for a request
2. Reads in the characters typed by the user
3. Analyzes the request (see chapter 8)
4. Carries out the requested action (see later chapters)
5. Back to step 1

However, not all requests processed by AIRS are typed in by the user. Some are formulated by AIRS itself in order to get the information it needs to carry out a user request. In particular, retrieval of traffic loads, the fundamental AIRS operaton, is invoked both explicitly (by a user request for traffic loads) and implicitly (by a user request, such as a quota flow request, which leads AIRS to request traffic loads).

AIRS formulates its request just as if it were an AIRS user. It does the equivalent of typing it in by putting the characters making up its request in the same place they would be if a user had typed them in. It then calls the appropriate parts of AIRS to analyze and process this request. Once this AIRS-generated request has been processed, the original processing continues, using the retrieved data.

The fcllowing requests to AIRS cause internal request generation:

1. Arrival delay prediction or flow control procedures

In order to carry out these requests, AIRS must know the arrival demand for the airport in question. The user's request must specify a single airport of interest and cannot specify a date or place any conditions (like the ones possible in a simple traffic load request) on the traffic. In order to process a request of this type, AIRS formulates a request for the arrivals at the given airport for the entire current operational day.

2. ARO schedule updates

Reservations: Under some circumstances, AIRS may be required to check a flight being entered to be sure it has not already been entered. It does this by formulating a request for the scheduled departures from the given origin to the given destination on the given date during the given

hour (it checks the whole hour in which the flight departs). It can then check through the retrieved flights for the particular one being entered by comparing flight IDs.

Cancellations: In order to cancel a flight in the data base, AIRS must first find it. A cancellation request specifies either departure or arrival information and may optionally include the other airport. For an arrival cancellation, AIRS requests the scheduled arrivals at the given destination (from the origin, if given) on the given date during the flight's hour of arrival. For a departure cancellation, AIRS requests the scheduled departures from the given origin (to the destination, if given) on the given date during the flight's hour of departure. It can then check through the retrieved flights, as above, to locate the particular one to cancel.

3. Departure delay entry

Departure delays can be entered for the current day by specifying the airport at which they apply and the hours during which they apply. These delays are entered into each affected flight in the data base. Thus AIRS makes a request for the scheduled departures from the given airport for the entire current operational day. It requests the whole day's traffic since the user's request can specify any number of hours and associated delays.

8. REQUEST ANALYSIS

AIRS accepts many types of requests, each having a great deal of flexibility in its form and content. Chapter 6 showed the two main request cycles in AIRS (the ARO cycle and the regular AIRS cycle) and the types of requests each handles. Within each cycle, AIRS never knows in advance what type of request to expect. This chapter explains how AIRS analyzes requests. The methods used are very flexible, allowing easy modifications and additions to request syntax.

The basic process (for the regular AIRS cycle, not ARO) is as follows. The request, which starts out as just a stream of characters, is scanned to locate the various "words" in it. Each word (except for numbers) is looked up in the "dictionary," which defines it by giving a category (e.g. classifying it as an airport code) and a specific value (e.g. the number used to index information for the airport). The request is thus transformed from a character stream to a list of category-value pairs. This form of the request is then examined for recognizable phrases. The recognition of a phrase leads to information about the request contents being stored in a standard form and determines how the rest of the request is analyzed. Finally, the standard form is examined and any default information is filled in. It is the standard form that is used later to actually perform the requested action.

The following sections describe the various aspects of request analysis mentioned above, as well as the analysis of ARO requests. We have tried to not only describe how things are done, but also why they are done that way. Limitations of this implementation as well as its good points will be discussed. Data processing in response to the analyzed request is discussed in subsequent chapters.

8.1 THE DICTIONARY

AIRS must have a large vocabulary including, among other things, all airport codes for scheduled traffic (currently over 1000), key words such as "LIST", and names of months. What should its dictionary consist of, and how should it be organized?

ORGANIZATION OF THE DICTIONARY

One approach would be to have separate lists of each type of word - that is, a list of airports, a list of key words, a list of months, and so on. Then to identify a word, we would read through all the lists until we found it. We would know what it was by knowing which list we found it in. This is obviously inefficient; How would you like it if you had to skim a whole dictionary to find a particular word? We could take a lesson from people and put all the words in a single list in alphabetical order. Then we could find a word by doing a binary search, like people do, taking ever-decreasing jumps back and forth until we zero in on our word. But with only one list, we don't know what a word is by virtue of where we found it, so we must store a definition along with the word. A third approach, still using a single list with definitions, is to use hash coding rather than alphabetical order. Hash coding means computing an address (in a table) directly from the word in question, so it can be found immediately. (We will not go into algorithms for hash coding). Of course, this same address computation is used when the table is formed. This is very efficient (in time), since except in the case when several words hash to the same address and further search is required, we find the word immediately. This is the approach AIRS takes.

The only drawback to a hash table is that it must have extra space in it to avoid being cluttered (having many words hash to the same space, thus longer lookup times). This is not a consideration in AIRS since the dictionary is already so big (requiring room for definitions as well as words, and having close to 2000 words) that it can't fit in core and has to be stored on the disk. A little extra space on the disk is no problem.

DICTIONARY DEFINITIONS

It is not sufficient for the definition of a word to just tell what type of word it is; that is, whether it is an airport, a key word, etc. In many cases, in order to process the request we need more specific knowledge about the word, a defining value. For example, let's say a request asks for J747s. We know from the dictionary that this is an aircraft

type, so we must test flights to see if their type is J747. But on the master schedule aircraft types are encoded as numbers to take up less space, so we must translate J747 to its code number in order to do the comparison. The dictionary should provide this code number as the value of J747.

Thus the dictionary contains a two-part definition for each word - a category and a value. The value is not always an encoding value, as it is in the above example for aircraft types. For each category the values are whatever is appropriate for that category.

The following table shows the categories of words in the dictionary, what words are assigned to each, and what values are used to further define the words. Although AIRS must understand numbers, they needn't be in the dictionary since they can be directly recognized and evaluated.

TABLE 8-1 - THE DICTIONARY

Category	Words	Value
airport	3-letter codes for all airports in the AIRS schedules - e.g. JFK	number used as code for the airport in the schedules - also used to index information about the airport
center	3-letter codes for all U.S. ARTCCs - e.g. ZNY	number used to index information about the center
area	Words defined to stand for groups of airports &/or centers Currently contains foreign groups, such as ZEUR (Europe)	index to group's members in the group definition table
airline	2-letter airline codes for all scheduled traffic - e.g. TW	the airline itself (same as the word)
airline group	Words defined to stand for groups of airlines - currently only TAXI, standing for commuter airlines	index to group's members in the group definition table
aircraft type	Codes for all aircraft types of scheduled flights. All start with J, P, or T - e.g. J727	number used as code for the type in the schedules
aircraft type group	Words defined to stand for groups of aircraft types. Currently three are defined - J(jets), P(props), and T(turbo-props)	index to group's members in the group definition table
month	first 3 letters of month name followed by a period e.g. APR.	the corresponding number from 1 to 12

TABLE 8-1 continued

Category	Words	Value
day	first 3 letters of day name followed by a period e.g. TUE.	a number from 1 to 7 starting with Sunday=1
direction	A (for arrivals) or D (for departures)	A=1, D=2
request-type key words	words which identify the type of request e.g. LIST, ?, QFLOW, TEST	the word itself, or a synonymous word if any
other key words	other key words used in requests but not classifying them - e.g. STACK, -, FROM LNDG	the word itself, or a synonymous word if any
relation	words standing for arithmetic relations - e.g. L (less than), G (greater than)	pointer to a program which tests the relation
flight data selectors, type 1	characteristics of flights that can be used in arithmetic relations (see above) and in listing requests - includes words for planned time of departure, etc.	pointer to a program which selects the data
flight data selectors, type 2	a) characteristics of flights that can't be used in relations, just in listing requests - e.g. ORIG (origin),...	pointer to a program which selects the data
	b) words used in listing requests to stand for different flight data selectors dependent on context - e.g. AIRP (airport)	a code number

VOCABULARY RESTRICTIONS

There are two restrictions on words in the AIRS vocabulary:

1. They cannot be longer than five characters:
This is not much of a hardship, since most words AIRS should know fall within this limit anyway. Airport codes are three letters, aircraft type codes are at most four, etc. The only problem is in inventing key words; it is sometimes hard to think up short enough words with mnemonic value.

The reason for the five-character limit is that that is the number of characters that can be stored in one word (computer word) on the PDP-10. It would be ridiculously wasteful to allocate two computer words everywhere AIRS words are being stored, since only a handful out of thousands would use the extra space. It would introduce non-uniformity and therefore added complexity in handling if AIRS words took different amounts of space. Although this could be done, it was not deemed to be worth the work. It is desirable in any case to keep words concise to minimize the typing required to make a request.

2. A word cannot have two meanings:
At first glance, this seems obvious. However, there are cases in which it would be nice to allow two. For example, AIRS originally used as its names for months the first three letters of the month name. But it turned out that several of these were airport codes, so we now require a period after the month abbreviations to make them unique. This again keeps the dictionary and request handling simple and uniform, since we know that looking up a word gives a simple definition in a known form. Another reason for single meanings, as we will see in section 8.3, is that the syntax analysis can't handle ambiguity.

8.2 WORD RECOGNITION

Once a request has been formulated (see chapter 7) and determined not to be a comment (see chapter 15.5), it is broken up into "words," where a word is one of the following types of character strings:

1. Number: a string of digits - e.g. 1230
2. Word: a string starting with a letter, containing letters and/or digits, and possibly ending with a period.
3. Special symbol: any character other than a letter, digit, or blank

For example, the request "A ORD J727,J707 1200- 2000MAR.20" (727s and 707s arriving at O'Hare between 1200 and 2000 GMT on March 20) would break up into the words:

"A" "ORD" "J727" "," "J707" "1200" "-" "2000" "MAR." "20"

Although blanks serve to delimit words, they are not necessary if there is no ambiguity. For example, in the above request, "2000" and "MAR." are recognized as separate words because a word starting with digits can only contain digits, so the "M" must signal the start of a new word.

Each "word" is checked as follows:

1. Numbers: Since numbers are used in AIRS to denote times, dates, landing rates, etc., none of which can be bigger than 2400, numbers greater than 2400 are rejected as erroneous.
2. Words: Since AIRS requires that a word fit into a single computer word, words longer than five characters are rejected as erroneous. Words up to five characters are looked up in the dictionary. If a word isn't found, it is rejected as an error.
3. Special symbols: Special symbols, like words (2), are looked up in the dictonary. If they are not found, however, they are simply ignored. Thus characters such as comma, which have no special meaning in AIRS and are therefore not in the dictionary, will still act as word delimiters.

Once the words have been identified, they are stored along with their definitions, as found in the dictionary. Each word is defined by a code categorizing it and a value unique to it. Although numbers are not in the dictionary, they are easily assigned a category (number) and value (their integer value). Thus the above request would be transformed into a table as follows:

Word	Category	Value
"A"	a direction	The code number standing for arrivals
"ORD"	an airport	the number used to encode ORD in the schedule and to access information about ORD
"J727"	an aircraft type	the number used to encode J727 in the schedule
"J707"	an aircraft type	The number used to encode J707 in the schedule
"1200"	A number	1200
"-"	a key word	"-"
"2000"	A number	2000
"MAR."	A month	3 - i.e.the third month
"20"	A number	20

8.3 SYNTAX ANALYSIS

OVERVIEW

The category-value form of the request obtained in the last section is now analyzed as follows. The beginning of the request is compared to all possible patterns until one is found that matches. The recognized request phrase is processed as appropriate, then the part of the request immediately following it is analyzed in the same way. This process continues until the entire request has been analyzed. If at some point no pattern can be found that matches the request, the request is declared erroneous.

PHRASES

In the above, a pattern is a sequence (any length) of category-value pairs. The value may be unspecified, so that any member of the category will match. For example, one pattern for specifying a time period would be: any number, followed by the key word "TO", followed by any number. This is a three-word pattern in which the first and last words need only be the right category (number), but the middle word must have a particular value ("TO") as well as category (key word). The request phrase "1800 TO 2200", for example, would match this pattern.

FLOW OF CONTROL

Recognition of a phrase determines the sequence of patterns to be tried in analyzing the next part of the request. These may repeat patterns already tested, or may be entirely different. For example, a demand request can contain any of its possible phrases in any position, so in analyzing a demand request we repeatedly try the same sequence of patterns. On the other hand, if we recognize (via the first phrase) that a request is for flight listings, or for entering landing capacities, we must look for phrases specific to these types of requests.

DISCUSSION OF APPROACH

This phrase-oriented approach has the advantage of clarity, since phrases are seen as a whole, hence ease of syntax modification. This is important in an evolutionary system such as AIRS. Since the first version of AIRS was implemented, many phrases have been added or modified; even

new types of requests have been added. This approach also has a disadvantage - it tends to give insufficient error diagnostics to the user. If no legal pattern matches the next part of the request, all we know is that it doesn't match. Other approaches which might have had better diagnositics were rejected because they sacrificed program clarity.

AMBIGUITY

It was pointed out in section 8.1 that a word cannot have two meanings. If it could, the syntax analyzer would have to decide between them; AIRS can't ask the user which one he meant, since the request might have been formulated by part of AIRS (see chapter 7), not by the human user. But the syntax analyzer can't handle ambiguity. Since it processes a phrase as soon as it recognizes it and never looks at the next part of the request until it has understood the current part, it can't recognize that there is more than one possible interpretation and look ahead to decide which one makes sense. Even it it could, there might be cases in which it couldn't distinguish between the interpretations, since both might be meaningful.

PHRASE PROCESSING

We said that when a phrase is recognized, it is processed; that is, information obtained from it is stored in standard locations. Flags may be set or data items recorded for use in executing the request. When a phrase that determines the type of request is recognized, a switch is set to indicate what AIRS should do to process the request after it is completely analyzed. The storage locations (if any) used to hold the standard information for that type of request are initialized, and the request analysis continues. In the following, we will show for each type of request the standard items set as a result of phrases that can appear in that request type. We will also indicate (in brackets) the action taken after request analysis for each request type.

DEMAND [traffic load retrieval - see 9, 10.1]
(note: this type of request is recognized by not being any of the below types, not by recognition of any phrase)

1. Places (airports, centers, areas) are divided into two categories:

a. Places of interest - the first place in the request or, if a parenthesized group of places appears first, all places in the group. These are the places the arrival/departure flags, times, etc. (below) refer to. I.e. For arrivals, these are the destinations; for departures, these are the origins.

b. Other, or restricting, places - those appearing after the first group. These determine the flights' "other end" - i.e. for arrivals, the origin; for departures, the destination.

Areas are looked up in the group definition table and replaced by their constituent airports and centers. The places are further divided, depending cn whether they are to be included or excluded (whether a minus sign precedes them). They are thus stored in one of four lists:

a. places of interest to include
b. places of interest to exclude
c. other places to include
d. other places to exclude

2. time period (possibly one-ended), if any, is stored (only one can be appear)

3. a single day and/or date can be stored

4. Aircraft types are stored in one of two lists: one if they are to be excluded (were preceded by a minus sign) and the other if they are to be included (were not preceded by a minus sign). Aircraft type groups appearing in the request are looked up in the group definition table and replaced by their constituent types. These are then treated as if they had appeared individually in the request.

5. Airlines, like aircraft types, are stored in one of two lists: one for airlines to include and one for airlines to exclude. Airline groups are expanded and stored like aircraft type groups.

6. Flags are set to indicate the appearance of keywords which:
   a. select arrivals and/or departures
   b. ask AIRS to ignore certain or all departure delays
   c. speed up processing by locking out ARO or not storing retrieved flights

7. Restrictions on time enroute, departure time, or arrival time can be stored. This involves storing the flight data selector, arithmetic relation, and number making up the condition.

LIST [list flights - see 10.4]

stores time period, if any

SORT [sort for listing - no processing action; has no effect unless it appears as part of a listing request - see 10.4 for effect]

stores the flight data selectors to sort by in separate lists for arrivals and departures

INFO [format for listing - no processing action - see 10.4 for effect]

for each specifier in the request:
- sets up format specification to be used when typing data
- stores code numbers identifying what flight data to report - separate lists for arrivals and departures
- stores column headers to put on report - separate lists cf headers for arrval and departure reports

PLOT [plot demand data - see 10.5]

stores time period, if one is given

TEST [arrival delay predictions - see 10.2] and
AFCP,QFLOW [flow control procedures - see 10.3]

sets time period, if any
- for delay prediction, used as report time period
- for flow control, used as override time for implementation

sets stack size, if any (only one can be given)
sets associated stack time, if any
stores airport (only one can appear)
sets flags indicating what key words appeared in request
- for example: TEST, QFLOW, AFCP, EDIT, XARO, CONT, ...

ENTER LNDG [enter landing capacities - see 11.1]
sets flag for today or normal values - initialized to today value, set to normal if "NORM" appears
stores each airport along with the rates to entered for it for each hour

? LNDG [report landing capacities - see 11.1]
stores time period of report, if any
stores list of airports specified, if any

ENTER GENAV [enter GA factor - see 11.1]
sets flag for today or normal values - initialized to today value, set to normal if "NORM" appears
stores each airport that appears along with the associated factor; if no factor given, sets it to reset the factor to normal

? GENAV [report GA factors - see 11.1]
stores airports listed, if any, in an array

ENTER DELAY [enter departure delays - see 11.1]
stores each airport along with the delay to be entered for it for each hour

? DELAY [report departure delays - see 11.1]

stores time period of report, if given
stores list of airports specified, if any

ARO. [switches to ARO input mode for subsequent requests - see 8.4]

no other request phrases

MAIL [print messages from TSC - see 15.2]

no other request phrases

QUIT [causes AIRS to terminate]

no other request phrases

Recall that the phrases being recognized are series of category-value pairs. When we say that a request item is stored, we generally mean the value of the item. In the case of places in a demand request, both the category and value are stored, since AIRS needs to distinguish between

centers and airports so as to know which cross-reference file to use (see 5.1, 9.1). See table 8-1 for the meanings of the values of various request words.

8.4 ARO REQUESTS

ARO requests are analyzed separately from all other AIRS requests. A command ("ARO.") to AIRS puts the program into ARO mode. The ARO request analysis and processing then takes over until commanded by an "AIRS" request to go back to the regular AIRS section. ARO request analysis is separate from the regular AIRS for several reasons.

1. Requests contain flight IDs, which are up to seven characters. The AIRS mechanism, as we have seen, allows only five-character words, since it uses only one computer word to store one word.
2. AIRS request analysis expects to find each "word" in its dictionary. Any word not there causes an error. However, flight IDs cannot be in the dictionary, since new ones are being entered all the time.
3. The date/time format in ARO requests consists of a string of five or six digits; the first one or two are the date (no month) and the rest is the time. Even if AIRS could handle the six-character field (which it can't), it would treat it as a single number.
4. The ARO request formats were inherited from an earlier system, so we had to deal with predetermined keywords (RA, CXD, CXA, GA, AN, AT, AQ). Though by luck it turns out these terms are unique in AIRS, the two-letter codes potentially conflict with airline names and the three-letter words with airport codes, so they have not been put into the AIRS dictionary, since it can't have a word doubly defined.

Requests are scanned to locate the beginning and end of each "word." Spaces are the only recognized delimiters. The first word is then checked against a list of command keywords to determine the kind of request. Some requests, such as "TAPE", consist simply of the key word, so there is no further analysis, and the request is simply processed. Others, such as "RA", need further analysis. The reservation and cancellation requests have fixed formats; that is, certain fields must appear in a certain order (the only optional field is the aircraft type). Thus rather than trying to decide what each "word" is, then deduce the meaning from the order and relationships of the words, as AIRS does, ARO knows what the word in each position should be. It analyzes each word on the assumption that it is what it should be, considering it an error if it isn't legitimate. For example, the flight ID can be any (seven or fewer) characters, but the user class must be one of the four legitimate codes. The airports (and aircraft type, if any) must be legitimate codes, as determined by looking them up in the regular AIRS dictionary. Date/time entries must contain possible dates (less than 32) and times (less than 2400).

Thus the ARO request analysis uses the AIRS dictionary for terms known throughout the system (airport and aircraft type codes) and its own word lists for other words (command keywords and user classes).

The kinds of requests recognized in ARO mode (and the information contained in each) are:

reservation [enter a flight schedule - see 12]
- flight ID
- user class
- origin airport
- destination airport
- departure date (not including month) and time
- estimated time enroute
- aircraft type (optional) - if omitted, make-believe type "NONE" is substituted

cancellation [cancel a flight schedule - see 12]

either:
- flight ID
- origin
- departure date and time
- destination (optional)

or:
- flight ID
- destination
- arrival date and time
- origin (optional)

The optional airports are ignored if they are not recognized.

TAPE [switch to paper tape input - see 12]

KEY [switch to regular keyboard input - see 12]

HELP [explain request formats - see 15.3]

AIRS [return to regular AIRS input mode - 8.2]

QUIT [causes AIRS to terminate]

8.5 CONSISTENCY AND SUFFICIENCY

After a request has been completely recognized, it may need to be checked for consistency and sufficiency of data specification.
Examples of the types of situations discussed below are given in table 8-2.

CONSISTENCY

If a request contains inconsistent data, it must be rejected.

SUFFICIENCY

If a request is incomplete (is missing information which AIRS needs in order to process it), AIRS will complete it with reasonable values if it can, or reject it if it can't.

Required Information

A free-form request may be completely recognized (8.3) even though it is missing essential data. Such a request will be rejected at this stage.

Optional Information

Default values are filled in for optional, omitted items. Subsequent processing stages are ignorant of whether the information came from the user or from AIRS.

OTHER ASSUMPTIONS

There are other assumptions made by AIRS to cover unspecified information, but which are not really request completion. These are of two types:

Unspecifiable

Certain information cannot be specified in a request; AIRS always decides the values itself.

Optional and Unnecessary

Some optional information is not needed to process the request, so values are not filled in. The processing sections of the program take different actions depending on whether or not this information was supplied.

TABLE 8-2 - CONSISTENCY AND SUFFICIENCY EXAMPLES

CONSISTENCY

Dates in demand requests must be legal. Feb. 29, for example, would be rejected except in a leap year.

If both a date and a day of the week are included in a demand request, they must be consistent. That is, the given date must fall on the given day.

A time period in a LIST or PLOT request must fall within the time range for which data is available - i.e. the time period of the last request which retrieved flights.

SUFFICIENCY - REQUIRED INFO

A demand request must include a place name (everything else is optional!).

If a demand request is not for the current day (specifies a date or day) it must specify a time period.

A request for flow control or arrival delay prediction must include an airport.

SUFFICIENCY - OPTIONAL INFO

If no sort specified in LIST request, arrivals sorted by arrival time and departures by departure time.

If no time period given in LIST or PLOT request, time period of available data assumed.

demand request:

If no date or day specified, current day assumed.

If no times specified (and request is for current day) a five-hour period starting with the current hour is assumed.

If neither arrivals nor departures is specified, both are assumed.

If no times given for landing rate report (? LNDG), report covers entire operational day.

If no times given for arrival delay prediction (TEST), report covers entire operational day.

TABLE 8-2 continued

UNSPECIFIABLE INFO

Flight reservations and cancellations include date but not month. AIRS chooses the month on the assumption that the request is for a date between previous day and three weeks in the future. A date not in that range is rejected.

OPTIONAL INFO

If no airports listed in landing rate request, all airports having capacities are reported.

If no times given in flow control request, AIRS computes and recommends times.

9. FLIGHT RETRIEVAL

This chapter describes how AIRS retrieves the flights satisfying a demand request. We assume that the request has already been analyzed and default conditions filled in, so that we now have a complete specification of the desired flights (see chapter 8.3). That is, we know:

1. the date and day of the week
2. the time period
3. whether arrivals, departures, or both
4. the airports &/or centers of origin &/or destination
5. restrictions as to aircraft type and airline (if any)
6. whether to ignore departure delays, flow control delays, or both
7. conditions on the time enroute, departure time, or arrival time (if any)

The entire retrieval process from here on is done separately for arrivals and for departures. AIRS uses the flight specifications listed above to build its representation of the meaning of the request. This representation has two parts: a list of pointers to flights having the right origin/destination combinations (section 9.1), and a program to filter flights to select those satisfying the rest of the conditions (section 9.2). It then uses the pointers to (randomly) access the possible flights, and one by one applies the filtering program to them. It counts up all flights that pass the test, and remembers them so that further processing can be done on them if desired (see chapter 10, processing retrieved data).

9.1 FLIGHT RETRIEVAL POINTERS

THE IDEA

To read the entire flight schedule file, testing each flight tc see if it satisifed the request conditions, would be prohibitively slow due to the large number of disk accesses required. We narrow down the search by first figuring out where on the file the flights having the requested origin/destination combinations are; then we test only those flights to see if they meet the rest of the request criteria.

We discussed in chapter 5.1 the indexes (or cross-references) to the schedules by airport and by center. We can look up any airport or center in the index file and find pointers to the flights originating or arriving there. Chapter 5.1 also explained why we decided to use airports and centers to index the schedules rather than, for example, aircraft type or arrival time. The methods of retrieval described here would apply no matter what we chose to index by. The general idea is to use the request conditions having associated cross-references to find pointers to the flights satisfying those conditions, then to retrieve just those flights and test them for the rest of the request conditions. This section will explain how we use the cross-references to form a list of pointers to the appropriate flights.

THE PROCEDURE

We saw in chapter 8.3 that the places named in a request are divided into four groups according to their meaning in the request:

group 1: places of interest to include
group 2: places of interest to exclude
group 3: restricting places to include
group 4: restricting places to exclude

For example, if we had a request for the traffic departing all airports in the Miami ARTCC except Miami International and Tampa International destined for any airport in the New York ARTCC except Kennedy International and La Guardia, the groups would be:

group 1: Miami ARTCC
group 2: Miami International, Tampa International
group 3: New York ARTCC
group 4: Kennedy International, La Guardia

For the sake of simplifying the discussion, we will assume we are processing a departure request. The same discussion would hold for arrivals if we just interchanged

the words depart/arrive and origin/destination. Thus places of interest are origins and restricting places are destinations, so the flights we want are the ones whose

- origin is one of the places of interest to include but not one of the places of interest to exclude
- destination is one of the restricting places to include but not one of the restricting places to exclude.

We form a list of pointers to these flights as follows:

In the index files, look up each place in group 1 to get pointers to the flights originating there. Combine these all into a single list, L1. Thus L1 has pointers to all flights with origins we want to include.

Look up each place in group 2 to get pointers to the flights originating there. Combine these into a single list, L2. Then L2 has pointers to all flights with origins we want to exclude.

The flights with legitimate origins are those that appear on L1 but not on L2. To get these, we form L3 by complementing L2 - that is, L3 has exactly those flights not on L2. Then L3 points to all flights we do not want to exclude. Thus the flights we want are those that are on both L1 and L3 - that is, the ones we were asked to include and were not asked to exclude. We therefore form L4, the list of all flights on both L1 and L3. L4 now points to all flights with legitimate origins.

Similarly, we use the group 3 and group 4 places to get a list, L5, of all flights with legitimate destinations.

Finally we get a list of all flights having the requested origin/destination combinations by taking those flights that are on both L4 (flights with requested origins) and L5 (flights with requested destinations).

This final list is the one used to retrieve flight schedules. Note that in the simple case of a request for traffic for a single airport, the above procedure reduces to just looking up the pointers for that airport.

POINTER MANIPULATION

Three basic operations are used in manipulating the pointer lists:

- complement (forming a list of all things not on a list)
- union (forming a list, without duplication, of all things on either of two lists)
- intersection (forming a list of all things on both of two lists)

In particular, we took the intersection of:

the union of the origins to include
the complement of the union of the origins to exclude
the union of the destinations to include
the complement of the union of the destinations to exclude

The operations operate on one or two lists of pointer ranges in increasing order, as found on the cross-reference files, and produce a list in the same form (sorted ranges) on a scratch file. If more than two lists must be combined, the first two are processed to form an intermediate answer on a scratch file, then the third is combined with that answer, and so on until they are all incorporated. Scratch files must be used since the lists cannot be assumed to fit in core.

Keeping the lists sorted enables AIRS to be very efficient in manipulating them. For example, suppose you have to find the intersection of two unordered lists of numbers (forget about ranges, for the moment). In order to decide whether an item on the first list is on the second list, you must search the second list until you either find it (in which case it belongs in the answer) or exhaust the second list (in which case it is not part of the answer). You thus search the second list once for each item on the first list. On each search you examine either the whole list (if the item is not found) or (on the average) half the list (if the item is found). Considerably less searching is required if the lists are ordered, since you can stop looking as soon as you find a number larger than the one you're searching for. Then when searching for the next number, you can start from where you left off on the previous search, instead of from the beginning. You are thus making only one pass through each list. This can further be thought of as going through the two lists in parallel, moving a finger down each list. At each step, you compare the two numbers pointed at. If they are the same, that number belongs in the answer (the intersection) and you move on to the next pair of numbers. Otherwise, the smaller can be rejected (it can't appear anywhere later on the other list) and the finger is moved down to examine the next number on that list. Similar reasoning applies to the other operations.

AIRS operates as just described, processing the lists in a single pass. The comparisons are slightly more complex for the lists of ranges AIRS deals with than for simple lists of numbers, since the ranges must be checked for overlap, not equality. In the case of intersection, the overlap is the part that goes into the answer list; in the case of union, overlapping ranges get combined into a single range.

The efficiency of the operations, the fact that each list is read exactly once and sequentially, is particularly important because the lists are being processed directly from the disk file. A nice bonus of the above method is that the final answer list is sorted, so when it is used to retrieve flights, the schedule file is accessed in one sequential pass, with no jumping back and forth. As discussed in 5.1, this is the most efficient way of accessing the file.

9.2 FILTERING

We now have pointers to those flights having the right origin/destination pairs. These are the flights that will be accessed. This section discusses how the rest of the request conditions are applied to the accessed flights. The conditions are used to write a filtering program which tests a flight and decides whether it satisifes the request. Because this approach is unusual, we will first try to give the rationale for it. We will then explain how the programs are written and run, and give some idea of what is contained in the programs for AIRS flight retrieval. Finally we will comment on other uses of this program mechanism throughout AIRS.

Representation of meaning as a program

Certainly the standard form specification of the desired flights (formed in chapter 8 and reviewed in the introduction to this chapter) contains all the meaning of the request. Items appearing in the request are classified not only by their inherent meaning (e.g. a number) but also by their role (meaning) in the specific request (e.g. end time of period of interest). At this stage we could certainly go directly to the retrieval process. The flights would be accessed and a program would decide whether they satisfied the request conditions. Most of the conditions which can appear in a request (such as restriction by airline) appear relatively rarely. The program that tests flights, however, being a fixed program, must take all possible conditions into account. It must first test whether an optional condition does exist, then if so, apply it to the flight. The program is run for each flight accessed, so these tests are performed for each flight. Decisions on whether a condition (such as airline) is applicable, however, depend only on the request, not on individual flights. Why not make these decisions just once, when the request is analyzed, then forget about them? A person would certainly not follow a fixed, all-encompassing procedure when doing manual data selection, but would tailor his procedure to fit each particular activity. For example, a travel agent using the Airline Guide would probably not pay attention to a flight's airline (unless an airline were on strike) when compiling a list of possible flights for a client. Although he is capable of making decisions based on airline, he would not normally ask himself, "Do I have to check the airline today?" each time he moved his eyes to the next flight on the list.

Thus we would like to tailor the decision procedure to the specific request, assembling just the appropriate tests,

making the decisions on what is relevant just once. But putting together a procedure is another way of saying writing a program. AIRS writes a flight filtering program which, given a flight, tells whether it passes. This tailored program is AIRS's representation of the meaning of the request. It is used in deciding what flights satisfy the request.

Decisions which are made only once can afford to be more complex than decisions which must be repeated over and over. Thus we could even include some optimization in this program-writing. In assembling the program, we could choose the order of tests to be performed depending on the request contents so as to narrow down flights with fewer tests. For example, if a request restricts retrieval to one aircraft type and covers a 24-hour period, it might be good to test for aircraft type first. However, if it restricts retrieval to twenty aircraft types and a one-hour time period, testing the time period first might be better. Such order optimization is not done in AIRS's program-writing, but it could be.

Writing and Running Programs

How can a FORTRAN program write and run a program? Programs can only manipulate data! Programs are programs and data is data and never the twain shall meet (apologies to Rudyard Kipling). Certainly AIRS cannot write a FORTRAN program, which would have to be compiled, loaded, and executed, none of which could be done within AIRS. Nor could it write a program in any external language, which would require control of the compiler or interpreter for that language. AIRS programs are written in a special, internal language designed for the purpose, and AIRS contains an interpreter for that language. (Note: The language and interpreter are more closely related to LISP, reference 9, than to any other standard language, and the ideas for them were derived from LISP.) The elements of the language are just data items to the rest of AIRS, which writes a program by putting the elements into an array (the program-writing area). It can then call the interpreter, passing it a pointer to the start of the program (the index in the array where it starts).

Programs are made up of operators and operands, written in prefix form. For example,

(AND (GREATER PTD 1200) (LESS PTD 1500))

is an expression which, if evaluated (interpreted) with respect to some flight, would have value true if the flight's planned time of departure fell between 1200 and 1500, and false otherwise. "AND" is the operator in the

top-level expression, and it has two operands (in this case), each of which is an expression having an operator and two operands, etc. The interpreter can perform a variety of "primitive" operations in terms of which all programs are written. These include, for example: logical operators (and, or,...); arithmetic operators (plus,minus,...); relations (greater than, less than,...); operators to extract each item from a flight record (airline, planned time of departure,...); operators to store items in a flight record; and many more.

The interpreter (hence the language) is recursive - that is, it calls itself. While trying to evaluate an expression, it may have to evaluate a subexpression, which it calls itself to do, before it can continue with the original operation. Expressions can thus be nested to any depth. Moreover, although a FORTRAN program can't call itself (or result in a call to itself before it has returned), this recursive language can, for example, contain an AND subexpression in an AND expression. This recursive interpreter was quite natural to write in PDP-10 assemply language.

AIRS Filtering Programs

We will not describe in detail the programs written by AIRS to filter flights, but just give a general idea of what's involved. Every request will need tests for a flight's date and time, but tests of such conditions as aircraft type are included only if applicalbe. But even the date/time tests vary from request to request. For example, if the time period covers more than one day (e.g. 1800-0200), the day to be tested depends on the time of the individual flight. The date and time tests also depend on whether arrivals or departures are being processed In one case, arrival times must be checked, in the other, departure times. Also, since the effective dates of a flight schedule refer to its departure, the date test for arrivals is more complex than that for departures. Depending on the individual flight, the departure and arrival days may or may not be the same. Thus if a request is for arrivals on June 18, a flight must be tested for flight either on June 18 (if it departs and arrives on the same day) or on June 17 (if it departs and arrives on different days).

Another complexity results from the treatment of delays (airport delays, 11.1; or AFCP controls, 10.3). Although a flight schedule in the data base may stand for many repetitions (on different dates) of the same flight, it can normally only contribute one instance of the flight to a request answer, because requests cover at most a 24-hour period and the flight, being for the same time on each of

its days, can only exist once in 24 hours. If delays are entered into the flight schedule for some day, however, that flight record can conceivably contribute two instances of the flight to a request answer: one at the regular time on some day and one delayed from the previous day so as to fall within the same requested period. There is thus no filtering program we can write that could answer yes or no on a flight (presumably also telling whether the delayed or regular flight passed) when delays are concerned. The solution requires that two filtering programs be written, one to catch flights that satisfy the request due to delayed flight, and one for scheduled flight. Each flight is processed through both programs, so it may correctly pass once, twice, or not at all. Of course, the content and even existence of the two programs will depend on the request, since the user may override (ignore) delays if desired.

Other Uses

Certain commonly used programs, such as those to extract data from a flight record, are written just once, when AIRS is initialized. Any part of AIRS that wants information on a flight calls the interpreter with a pointer to the appropriate pre-written program. No part of AIRS other than these programs needs to know the format of a flight record, and most parts don't even need to know (in detail) what's contained in it. For example, there are programs to return the planned time of departure (which is extracted from the flight record) and the estimated time enroute (which is computed from the flight record). The interpreter also takes care of all storing of data in flight records. In table 8-1, where certain words were given dictionary values which were pointers to programs, these programs were the ones we meant.

9.3 RETRIEVAL AND STORAGE

Now that we have pointers to the possible flights and programs to test the flights, we are ready to perform the retrieval. This entire procedure (including the formation of the pointers and the program) is done separately for arrivals and for departures, since the two directions result in both different pointers and different programs. The retrieval proceeds as follows.

The pointers are either on a scratch file or cross-reference file in increasing order. They are used to read in flight schedules, one at a time. The sorted pointers make the access more efficient than it would be otherwise by making it locally more serial, as discussed in 5.1. Each flight has the testing program run on it. Actually, as we saw in 9.2, a flight schedule having delays in it can fly twice during the span of a request - once with its delayed time and once with its scheduled time for the next day. Thus there are really two testing programs, one to see if the flight passes when it flies its scheduled time and one to see if it passes for its delayed time. The flights that pass the test are counted up - the flight is counted twice if tt passes both tests, since it is really two flights. A separate count is kept for each hour and for arrivals and departures. During arrival processing flights are counted by their hour of arrival, and during departure processing they are counted by their hour of departure.

The response to a user's demand request is simply a table of traffic counts (see 10.1). However, for many requests, the actual flights must be known, not just the number. A listing may be desired after a demand request (10.4), the request may have been made by AIRS in order to make arrival delay predictions (10.2) or do flow control (10.3), etc. Thus the flights that satisfy the request are remembered as well as counted. Since all the processing activities need the flights grouped by hour, they are remembered grouped by hour. Remembering consists of copying the flights, by hour, onto a scratch file (there is no room to hold them in core). Along with each flight is stored its pointer, since several processing activities (entering delays, 11.1; implementing AFCP controls, 10.3; cancelling a flight, 12) modify flights - i.e. write back different information into the flight on the master schedule file - hence need to know where they came from. Since a flight schedule, if it has delays in it, really is different flights on different days, an indication of which set of flight times (scheduled, departure-delayed, AFCP-controlled) were applicable in this retrieval is kept too. Arrival delay predictions and flow control require the flights in time order. If the retrieval was performed for

one of these activities the flights on the scratch file are sorted to be in time order, not just hour order.

There is a limitation in the current AIRS that only a hundred flights can be remembered for any hour (that is, a hundred arrivals and a hundred departures). This is because the saving of flights is done with an intermediate stage, to make grouping them by hour more efficient. When a flight passes the retrieval test, it is not actually saved. Just the added information about it described above (pointer to it and indicator of delays vs. scheduled) is saved, in an array in core, arranged by hour. When the retrieval is finished, the pointers saved for each hour are used to re-access the flights and save them on the scratch file. The 100-per-hour limitation thus derives from the room available to save the pointers in core.

If a user makes a demand request and is sure he will only want a table, not a listing, he may so indicate by including a key word ("XLIST") in the request. AIRS will then skip the remembering of the flights, saving considerable time.

10. PROCESSING OF FLIGHT DATA

10.1 DEMAND COUNTS

DEMAND TABLE

We saw in 9.3 that the flights retrieved in response to a demand request are counted up by hour (arrivals by arrival hour and departures by departure hour). AIRS responds to a user-generated demand request by typing out a table showing these hourly counts, and also the estimated unscheduled traffic (computed from the GA factor) if it can be estimated.

Table 10-1 shows two sample demand tables, one in which the GA factor was applied and one in which it wasn't. The table contains a heading telling the date and time period covered by the data, then gives the number of arrivals and departures by hour. Of course, if the request was for just the arrivals or just the departures, only the appropriate columns appears. If the GA factor is applicable, the arrival and departure columns are further broken down into scheduled traffic (A/C column) and estimated unscheduled traffic (G/A column). All of the time intervals reported start and end on the hour except, perhaps, for the starting and ending times of the report, which match the request times.

GA ESTIMATES

AIRS must decide whether it makes sense to apply the GA factor and, if so, whether the normal or current-day value applies. Since the GA factors are derived historically as a percentage of total scheduled traffic, it doesn't make sense to apply them to partial traffic counts, such as only the jets or only the traffic from certain centers. The factor is thus not used if the request placed any restrictions on the traffic. It also can't be used if the request was for demand at more than one airport, since each airport has its own factor and the traffic count is a single value, not broken down by airport.

If it does make sense to make an estimate, the factors are looked up (on the airport data file). If the airport has no factors, then of course no GA estimate will be made. If it has a current-day factor differing from the normal factor, AIRS must decide which one to use for each hour of data. (See 11.1 for an explanation of what date/time period the "current-day" factor applies to.) The appropriate factor is then applied as a percentage to each hour's traffic counts. The resulting counts are included in the demand table and also stored along with the counts of retrieved flights for use in further listings (10.4).

TABLE 10-1 - SAMPLE DEMAND TABLES

REQUEST=BOS JUN.18 FROM 2200 TO 0100

IT IS NOW 1755 THU. JUN.28

2200 MON. JUN. 18 TO 0100 TUE. JUN. 19

TIME	ARRIVALS		DEPARTURES	
	A/C	G/A	A/C	G/A
2200	26	9	29	10
2300	22	7	21	7
0000	27	9	16	5
0100				

REQUEST=A BOS JUN.18 FROM 2200 TO 0100 ZNY

IT IS NOW 1757 THU. JUN.28

2200 MON. JUN. 18 TO 0100 TUE. JUN. 19

TIME	ARRIVALS
2200	6
2300	6
0000	9
0100	

10.2 AIRPORT DELAY PREDICTIONS

The previous section described the processing required in reporting arrival and departure demands upon airports. Knowing these arrival demands and, if given estimated airport landing capacities, it should be possible to predict arrival delays, if any, and also the number of aircraft holding in the arrival terminal's airspace. This section describes how AIRS predicts these data in response to a request to "TEST" for airport delays. The approach takes the flights on a first-come-first-served basis (arrival times) and simply puts each arrival into the next available landing slot. When there are more arrivals than landing slots, the program simulates them in a holding queue until their landing times. Statistics are computed by time intervals ranging from one to six per hour. A report is then produced for the requested time period which tabulates by time interval, the arrival counts, the numbers landed and averages and peaks for both the arrival delays and the stack sizes. The report can be produced for the current day only.

Chapters 7 and 9 covered the formulation and processing of the AIRS request for retrieval of arrival flight data associated with airport delay prediction requests. The arrival delay processing follows the retrieval of flights, sorted by arrival time and temporarily stored on a scratch file on the disc. Two other forms of data are required and are prepared in conjunction with this flight data: the estimated number per hour of arriving general aviation flights (i.e. the fictitious flights computed from the GA factor; see 11.1) and the hourly estimates of landing capacity (see 11.1). Given this data, the program then checks the airport's control file (see chapter 5.3) to determine the time interval for the statistics, the requirement for applying the GA factored flights and the period of application of the GA factor. This control file is shared with the AFCP and the Quota Flow Procedures (see chapters 10.3 and 11.2). If no special requirements (control file) exist for the airport, the default controls are employed (currently one interval per hour and GA factor used all day). The computation of arrival delays can now proceed.

In computing the landing slots, the program uniformly divides the 60 minutes per hour by the estmated landing capacity for the hour (i.e. each landing slot is given a specific time uniformly distributed through the hour). Only one aircraft may use a landing slot; thus if two arrive during an available slot, one will use it, the other will take the next slot, thus being air delayed. The program sequences through the arrival flights, serially reading them

from the scratch file, and as required, mixes in the fictitious general aviation flights (using specific times of arrival as computed from the numbers of GA flights per hour in the same way as the landing slots). During each statistical interval the counts, averages and peaks are recorded and retained in core in an array. At the end of the flight processing the data array is output to a disc file combining it with other similar data (to be discussed in section 10.3). The data on this file is then output to the teletype by a formatting program and optionally can be plotted along with other data on a display terminal (currently the Conograph-10 unit). Table 10-2 is a sample of the delay prediction report.

TABLE 10-2 - DELAY PREDICTION REPORT

REQUEST=TEST BOS FROM 1400 TO 0400

BOS 6/28 1811

BOS LANDING CAPACITIES:

0700 TO 2000	35
2000 TO 2100	15
2100 TO 2200	20
2200 TO 0200	30
0200 TO 0700	35

GENERAL AVIATION FACTOR= 34 %

ORIGINAL TRAFFIC

TIME	ARR	LAND	AVHLD	AVDEL	PKHLD	PKDEL
1400	29	32	1	4	5	8
1500	28	25	2	1	3	4
1600	28	28	2	2	4	6
1700	28	31	1	2	4	6
1800	27	27	0	1	2	3
1900	29	28	1	1	3	4
2000	39	15	19	44	25	70
2100	39	20	39	78	44	87
2200	35	30	48	93	50	98
2300	31	30	50	99	53	105
0000	34	30	53	101	56	108
0100	23	30	50	95	57	102
0200	15	35	33	66	47	79
0300	20	35	18	40	29	48
0400						

There is one additional feature incorporated into delay predictions. The controllers often have knowledge of the stack size at some time. This information, if cranked into the prcgram, could improve the subsequent prediction accuracy. Specifically, the user optionally enters a stack size and its time of occurrence in the original "TEST" request. The program processess the flights in the normal manner until the time equal to that of the entered stack size. It then compares the holding queue with the given stack size. If they differ, the queue is adjusted to match the given size by landing or returning to airborne status the proper number of earlier flights. An equivalent number of landing slots for the affected aircraft is consumed or made available as appropriate. The program then continues computing the rest of the statistics in the normal manner.

10.3 AIRPORT FLOW CONTROL PROCEDURES

The ability to predict airport arrival delay conditions leads to possible actions to prevent or at least minimize excessive airborne delays and saturated airspace problems. Some problems are severe enough to affect many adjacent ARTCC's and require application of flow control procedures for improvement. In assisting the controllers in these procedures, AIRS computes ARTCC departure clearance allocations for AFCP, the original ground delay control method (Refs. 4, 5 and 8). It can also compute adjacent ARTCC release quotas for the newer Quota Flow Control Procedures (Ref. 6) This section discusses AIRS processing required to support both procedures applying to the current day only. The processing for the latter procedure employed an approach which capitalized upon the existing AIRS/AFCP program and was designed to expedite support for this newer flow control procedure. The section will discuss in detail the AFCP processing and then will discuss the minor program additions to the AFCP processing supporting the Quota Flow approach.

The AFCP processing begins like the arrival delay prediction processing, the only differences being that the request is for AFCP and that in addition to the airport's control file being input from the disc, a zone file, if any, is also read from the central data base. These files were summarized in Chapter 5.3. The zone file is restricted in its use to either the AFCP or the Quota Flow Control Procedures at any one time. This is because the zone structures for AFCP differ in their purpose and use from the Quota flow structures and because in expediting support for Quota Flow, a dual purpose zone file was of least importance.

As in the arrival delay prediction section, statistics are computed for the original set of arrivals. However, the output of the statistics to the teletype is suppressed following its storage on a disc file. Instead, the AFCP program examines the peak values of predicted air delay and stack size throughout the current day (a full 24-hour period starting at 0300 Eastern local time) and compares the peaks to the threshold conditions for imposition of AFCP. If a threshold, either maximum air delay or maximum stack size, has been exceeded the program undertakes detailed processing of ground delay assignments and the associated flow control data. If no AFCP thresholds are exceeded, the program outputs a corresponding message to the teletype and returns to the normal AIRS request entry mode for further instruction.

Following the computer's determination that AFCP is required, the program reads through the flights on the scratch file (same file used in computing the arrival delay statistics) and analyzes the status of each flight. The status includes whether it is currently airborne, what ARTCC and zone it originates from if any, whether it has already completed its flight, whether it is already AFCP controlled and if so, whether it is too late to alter the assigned ground delay if needed. This status information will be used later in deciding the controllability of each flight in assigning flow ccntrol ground delays. This status is also written into each flight record on the scratch file.

Next, the program computes the theoretical maximum rate of arrivals which will not exceed the control thresholds and compiles the associated discrete arrival times into a list (in core) with the corresponding landing times. Maximum arrival rates usually involve a requirement to maintain a backlog of waiting flights. The method of computing this maximum arrival rate considers both the stack size and air delay limitations and will be discussed in the next paragraph. But first we need an explanation of the use of this list. Given that the original traffic demand for an impacted airport exceeds the landing capacity, it follows that a number of aircraft will have arrival times different (earlier) than their landing times, hence air delays. One can make up a list of corresponding arrival and landing times for these flights similar to the theoretical maximum rate of arrival list discussed above. In one list we have the predicted (uncontrolled) traffic delays by flight; in the other we have the maximum flight delays tolerable within flow control restrictions. Looking at correspponding landing times in each list, one can ask if the associated arrival time for the actual flight is worse (earlier), than the theoretical (controlled) arrival time. If so, then the procedures dictate that the flight be assigned a ground delay such that it arrive no earlier than the theoretical arrival time. This comparison and ground delay assessment is the heart of the program's administration of these procedures.

The method of computing the maximum arrival rate is based upcn four control parameters: the maximum delay, the desired delay, the maximum stack and the minimum stack desired (the method may be viewed more clearly if one considers the first two parameters as equal and the last two equal; they are often set equal by the users). The arrival slot computed for a given landing slot is determined by the following procedure. The first step is to make the arrival slot equal to the landing time minus the desired delay. By examining the arrival times and landing times for the entire

day, the stack size at any given time can be computed. If this computed stack size is greater than the maximum stack value, the associated arrival slots are moved to a later time until the stack size equals the maximum stack threshold. If the computed stack is less than the minimum stack desired, the arrival slots are shifted in the opposite direction, to an earlier time, until the stack is equal to the minimum size. This step is dependent upon not exceeding the maximum delay, however. If this maximum delay is encountered, the arrival slot is set equal to it and is not shifted any earlier. The produced set of maximum delay arrival times corresponding to all landing times may be in fact, a mix of rates which alternately follows each of the four control parameters. Any one or combination of these four control parameters may be the predominant factor in determining the arrival rates by proper choice of values.

In assigning the landing slots for each flight, the same consideration of fictitious general aviation consumed slots is employed as in the arrival delay prediction processing. To summarize, the process involves a serial reading of the flights (in time of arrival order) from the scratch file. The next available landing slot is assigned each flight (the GA fictitious flights being included in the applicable distribution). If a flight's original arrival time is later than or equal to the maximum delay arrival time, no controls are needed and it is written out on a separate section of the scratch file showing that it arrives at the originally scheduled time. If the flight arrives earlier than the maximum delay arrival time and if it has a controllable status, the flight is flagged as controlled and an appropriate ground delayed departure time is computed and recorded on this separate section of the scratch file. When all the flights have been processed the program examines the newly written flights on the scratch file and computes the controlled arrival delay statistics in exactly the same manner as the original traffic statistics. The controlled traffic statistics are then output to the same disc file as the other statistics.

The program now outputs the recommendation for AFCP and the time it should be initiated (the original arrival time for the first controlled flight) and the time it should be terminated (the original arrival time for the last controlled flight). The peak delays and the maximum stack sizes are also output for three specific times; the peak delay and maximum stack for the entire day, the values at the time of control initiation and the values at the termination of the control period. The following example illustrates what the AFCP recommendation looks like:

```
JFK   7/5   1553

UNCONTROLLED PEAK DELAY 97 MIN AT 1812, PEAK HOLD 49 AT 2059
RECOMMEND CONTROLS BE INITIATED:
        INITIATE  1748 PKHOLD 28 DELAY 69 MIN(PK)
        TERMINATE 2205 PKHOLD 31 DELAY 60 MIN(PK)
```

The program then asks what the user wants to do next; does he want the statistical reports typed, does he want them plotted, does he want to print out the flow control allocations or does he want to issue the flow control allocaticns? The statistics are typed or plotted as in the arrival delay prediction case with the added choice of both original and controlled traffic conditions. The printing of the allocations involves further processing. The newly written flights on the scratch file are again read serially. Each flight is slotted by controlled time of arrival (or original time of arrival if not controlled) and by zone (and any grouping within the zone) into an allocation array. Upon completion of reading the flights, the allocation data is typed out. The issuing of allocations is similar to printing them but it performs one more operation. It writes the flight status and the delayed departure times of every flight processed onto the master schedule file of the central data base. This data is thus available for recomputations of AFCP at a later time if revised landing capacity estimates or changes in the traffic demand take place.

This recomputation of AFCP is triggered by including the key word "CONT", meaning continue, in the AIRS AFCP request. The processing is the same as before except that the controllability of some flights is altered by the earlier AFCP's. Indeed, a flight which would have been completed if it had not been delayed may be still on the ground awaiting the delayed departure time. If the need for AFCP's has diminished, the flight may be allowed to depart immediately, cancelling the remaining ground delay. The determination of such control adjustments is accomplished by flight status assessment discussed earlier and by a routine which closely monitors the flight's operational envelope (i.e. scheduled departure, controlled departure, scheduled arrival, controlled arrival and landing), to assure that flights before, after and/or during the required AFCP period be assessed the minimum amount of control delay consistent with the change in conditions from the earlier issuance of allocations.

The AFCP processing will automatically compute the need and duration of the control period as discussed above. It

can also compute the control requirements when given an overriding initiate and/or terminate time at the option of the users. It will use the given override time(s) and compute the other end of the implementation period as applicable, and will produce the associated control allocation.

As in the case of arrival delay predictions, a stack size and time of occurrence may be entered with the AFCP request. In the same manner as before, the computed airborne queue is adjusted to match the given stack size at the stated time. The stack adjustment affects the availability of landing slots, which in turn affects the subsequent control delays.

The first part of this section has described the AFCP processing. The remaining part will present the processing changes implemented to provide Quota Flow Control Procedures assistance. The principal difference between Quota Flow and AFCP is that Quota Flow does not impose arrival delays by ground departure controls but leaves the choice of air or ground delay to the adjacent tier centers (the ARTCC's surrounding the impacted airport's center) and the pilots. The control mode regulates the numbers of flights cleared during each interval of time for entry into the impacted center through quotas for the tier centers (and the impacted center). The procedures require that the traffic be viewed as seen by the tier centers (and the impacted center) in order to administer the flow controls. For this reason the program was modified to establish zones adjusted by boundary times which compensate for the flight time from the gate arrival (as recorded in the centralized data base) back to the tier center's boundary. The organization of the zones is extremely important to the quota flow operation since AIRS does not have the built-in knowledge of which aircraft pass through the tier centers. By carefully organizing into a zone all the origins whose traffic usually passes through a specific tier center, the program can compute the appropriate quotas for the center. Using this zoning information and control criteria establishing a maximum stack size for the impacted center, the program can compute the required delays exactly as it does for AFCP, but instead of producing the allocation report, the program is modified to produce the quota report. This quota report is simply a count by zone groupings, of controlled aircraft attributed to each tier center. The count represents the release rates to produce the controlled flow. In order to indicate how many aircraft might be delayed at any time in each tier center, the report also includes the number of aircraft held during each control interval. This is also simply computed by obtaining the difference between the cumulative flow of

original traffic and the cumulative flow of controlled traffic. Table 10-3 illustrates this Quota Flow Control Report.

One further note on processing: the quota flow report, because of the zoned traffic distributions, is of value to the controllers even when flow controls are not needed. For this reason the quota flow processing automatically continues through the quota report section, instead of returning to the AIRS request entry mode as AFCP does if controls are not recommended.

TABLE 10-3 - QUOTA FLOW CONTROL REPORT

QRD LANDING CAPACITIES:

0700 TO 2000	60
2000 TO 2200	45
2200 TO 2300	50
2300 TO 0700	60

ZONE 1 QUOTA FLOW

TIME	DELAY	PRZAU	PRZOB	THZOB	PRZID	THZID	PRZKC	THZKC	PRZMP	THZMP	TOTAL
BNDY TIMES		20	40	60	40	60	40	60	40	60	
2000	0	11	1	4	1	4	4	5	3	5	38
QF CARRY		0	0	0	0	0	0	0	0	0	0
2030	0	8	3	4	1	2	0	5	2	4	29
QF CARRY		0	0	1	0	0	0	1	0	0	2
2100	0	10	3	4	1	2	3	2	0	2	27
QF CARRY		3	1	9	1	5	0	0	1	3	23
2130	12	10	2	9	1	3	0	0	1	1	27
QF CARRY		6	4	5	5	6	2	3	1	4	36
2200	36	6	4	5	4	6	2	3	0	4	34
QF CARRY		3	2	6	3	5	3	1	4	8	35
2230	36	3	2	4	1	5	3	1	3	5	27
QF CARRY		7	2	6	3	5	1	3	4	13	44

10.4 LISTINGS

As described in section 9.3, the flights retrieved in response to a demand request (formulated either by a user or by AIRS) are stored on a scratch file, separated into arrivals and departures and grouped by hour (unless the user specifically said not to store them). Any number of detailed listings of these flights can be requested. This listing request can (optionally) choose the time period, the information to list, and the order of the flights.

Time Period

If no time period is given, all the stored flights are listed; if times are given they must fall within the time range of the stored flights. The flights are listed the way they are stored - by hour. No further subdivision is done, so the times must be either on the hour or at the start or end of the data period (if they are not, they are automatically rounded off as appropriate).

Sort

If no sort is given, estimated time of arrival (for arrivals) and estimated time of departure (for departures) are put into the sorting specification lists.

Information

The information to list can be specified (in an INFO request) either in the listing request or separately. Column headings, codes identifying the desired data, and format descriptions for typing the data are set up (separately for arrivals and departures) and remain in effect for all listings until explicitly changed. The standard information listed if none is specified is: flight identification, origin, estimated time of departure, destination, and estimated time of arrival.

The arrivals are reported first, then the departures. For each, the appropriate column headings (stored when the user made an INFO request, or standard headings if he didn't) are typed out. Then each hour is reported on as follows:

- The number of flights and the GA estimate, if any, are typed out (the same values that appeared in the demand table - see 10.1).
- The hour's flights are read in from the scratch file and sorted as specified. Up to ten subsorts can be handled (if two flights match on one item being sorted by, they are compared on the next item). The sort algorithm used is a Shell sort, a very fast method.
- The flights are then listed in this sorted order. The code numbers identifying the data to list are used to branch to appropriate data preparation activities (if the user didn't specify the information to list, standard code

numbers are filled in) which extract or compute the desired information from the flight record. The data is then typed in the appropriate format, using the specification stored when the user made an INFO request, or the standard one if he didn't.

If there were more flights retrieved for an hour than could be remembered (at most 100 arrivals and 100 departures per hour can be stored), that hour's traffic cannot be listed.

A sample listing is shown in table 10-4.

TABLE 10-4 - SAMPLE LISTING

```
REQUEST=LIST 2100-2200 , INFO IDENT ORIG DEST TYPE ;
       =SORT IDENT

LISTING FROM REQUEST:
JFK   JUN.  18    FROM  1800  TO    2300  J747  JD10

            ARRIVALS

    IDENT        ORIG  DEST  TYPE

2100    6

    AA00006      LAX   JFK   J747
    AA00014      SFO   JFK   J747
    AZ00608      FCO   JFK   J747
    BA00509      LHR   JFK   J747
    LY00015      LHR   JFK   J747
    TW00800      SFO   JFK   J747

2200

          DEPARTURES

    IDENT        ORIG  DEST  TYPE

2100    7

    AA00186      JFK   BOS   JD10
    AA00665      JFK   SJU   J747
    DL01023      JFK   ATL   JD10
    DL01069      JFK   MIA   JD10
    PA00110      JFK   FCO   J747
    PA00295      JFK   SJU   J747
    TW00049      JFK   SFO   J747

2200
```

10.5 PLOTS

AIRS offers the user a choice of plots to aid him in accessing the data produced by demand, delay prediction, AFCP or Quota Flow requests. The plots are designed for output on a Conograph-10 display device. The processing requirements for plots are divided into two categories, the processing involved in compiling the data and the processing required to format the data suitable for plotting. The data compilation processing depends upon the specific plots requested. If a plot is requested following a demand request for arrivals at an airport for a given time period, the processing first checks to see if the airport's landing capacities are available and can be plotted together with the demand data. It makes little sense to plot landing capacities with the demand data unless the plot concerns the total arrival traffic for a single airport even if there are known capacities for the airport. However, if the demand data is partial, the user is informed that the landing capacities contrast in that they are for all traffic. AIRS allows the plotting of capacities and the demand data to continue for this partial traffic case at the option of the user (assuming that capacities are known to AIRS). If landing capacities are unavailable or do not make sense to use, the plot will only contain the demand curve.

After checking for the landing capacities the data processing continues by one of two collection methods depending upon the interval of time selected by the user for data resolution. If the user selects an hourly bar graph, the hourly demand counts are already computed in an array in core. The processing simply transfers the data into the plotting array. Also available are the GA factor traffic estimates. These too are transferred if the plot requires them (i.e. if it makes sense to include them, as in the landing capacity situation). The data collection is now complete and the format processing can begin. If, however, the user has selected a ten minute resolution for data presentation (6 bars per hour; may be as detailed as eight per hour), the data collection must involve reading the flights serially from the scratch (disc) file as is done for computing the arrival delay predictions (see Chapter 10.2). The process basically counts the number of flights per interval and forms an array of the results (e.g. a histogram). Before proceeding to the formatting processing the GA factored traffic must be uniformly distributed through the applicable intervals as required. There is one potential limitation in plotting data intervals less than one hour. Because of a core restriction in retrieving flight data (see 9.3), AIRS currently does not retain individual flight data on more than 100 arrivals per hour

(and also 100 departures per hour). The counts are accurate, but the scratch file is incomplete. The plot processing recognizes this and will only plot in hourly data intervals when this limit has been exceeded in retrieving the flight data.

The format processing involves scaling of the data into cartesian coordinates, axis sizing, graph labeling, titling and the transformation of the data into bar and/or line graphs. The transformed data is then output to the display device and produces the desired plot of the data versus time. Figure 10-1 shows a sample plot of arrival traffic at a major airport.

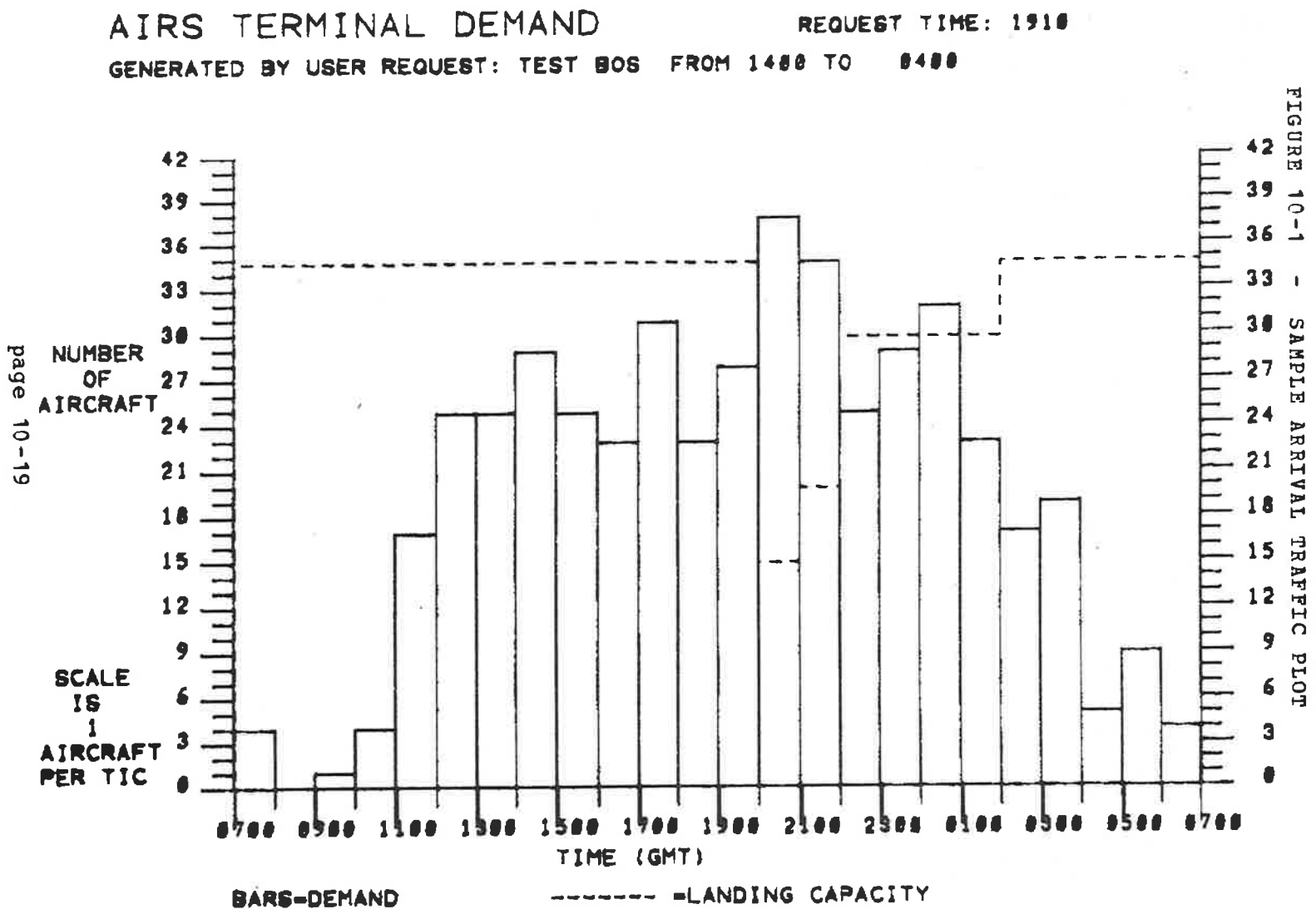
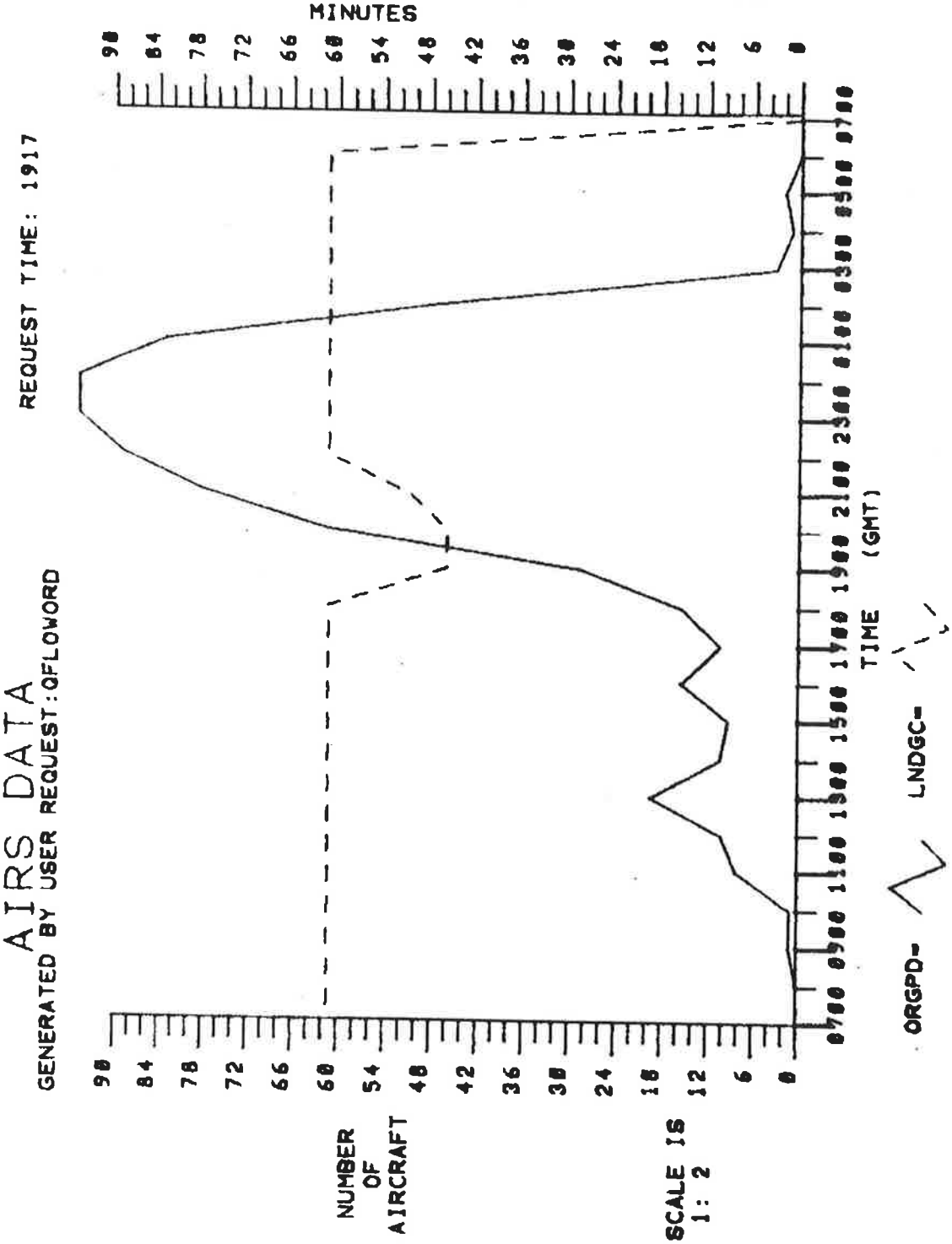


FIGURE 10-1 - SAMPLE ARRIVAL TRAFFIC PLOT

The above example covered the demand capacity type of plots. The arrival delay prediction statistics (data produced as described in Chapters 10.2 and 10.3) can also be plotted. The data compilation for the plot is simply a transfer of the appropriate data from the statistical data file produced by delay prediction and flow control requests. The controller is allowed to select up to four curves per plot, chosen from the statistics recorded for the original and the controlled (if any) traffic and including the landing capacity. The user can also control the line modulatien for each curve (i.e. solid, dashed, dotted, etc.). The format processing is as described above and the curves are plotted against time. Figure 10-2 shows a plot of the predicted peak arrival delays during a period of reduced landing capacity at a high density airport.

FIGURE 10-2 - ARRIVAL DELAY PREDICTION PLOT



11. AIRPORT DATA

11.1 ENTERING AND RETRIEVING AIRPORT DATA

Three kinds of airport data discussed in chapters 3 and 5.2 are the subject of this section. The data is that which is entered and updated by the staff of the SCC and consists of: 1. airport general aviation (GA) factors, 2. airport landing capacity estimates, and 3. airport departure delay estimates. These data are retained in the AIRS central data base distributed among two files, the airport data file and, as needed, the flight schedule file. The form and scope of data varies for the three kinds because of their nature and use within AIRS. In general, this data applies to the current day since the primary users of AIRS are the staff of the SCC and the problems they deal with originate and terminate within their current operational day. However, to reduce the amount of data required to be manually entered into AIRS each morning, a set of normal (or typical) data can be entered once (or as needed) for each airport. This normal data is automatically inserted (during the AIRS daily reset operation discussed later) as the current day airport data each morning (or first daily use). The users need only enter the abnormal data as required when an airport varies from the norm. This considerably reduces the volume of manual data entry. In order to handle both the current (today) data and the normal data, which frequently differ, AIRS treats each as a separate set of airport information. The airport departure delay data is the exception; the current day information is the only data entered and used. One further note is that the reference to operational day means from 0300 to 0300 EST or EDT. This 24-hour period is used in AIRS to begin and end computational processes at clearly cff-peak hours.

GENERAL AVIATION FACTORS

Entering and retrieving general aviation factors is the simplest of the three. The GA factor is derived from historic data on the ratio of scheduled air carrier traffic to general aviation traffic at an airport. It is expressed in percent GA of air carrier. The factors were derived for whole days, and since little variation occurs day to day at the major airports of interest, only one factor was required for each airport. Thus, single GA factors for normal and today only data are retained in AIRS. The entry of a GA factor involves obtaining the value from the user's request (Chapter 8) and storing it in the airport data file in the appropriate airport data block (one is created if none exists for that airport). Retrieval is simply the reverse; its use (as in Chapter 10.1) involves reading the value in

from the airport data file. After GA factors have been entered they may be reviewed by the user at any time through a request for this action. Airport data requests permit the user to look at the GA factors (or the landing capacities or the departure delays) for the airport or airports desired. It is also possible for the user to request this information for all the airports in a single request by not specifying any airport at all.

Before going on to the next kind of data, it is appropriate to discuss the entry interface between normal and today values. AIRS is designed to minimize the burden to the users for airport data entry as illustrated by the normal and today only classification and storage of data. AIRS goes one step further by automatically setting the today data equal to the normal data, if separate today data was not entered earlier in the day. In this way, the normal data entry also updates the today values, saving the user the extra entry work.

LANDING CAPACITIES

Entry of airport landing capacities is slightly more complicated than GA factors because in place of a single value, landing capacity estimates have a set of hourly values ccvering a twenty-four hour period. The interval of the estimates should be as small as is practical because it tends to improve the arrival delay prediction accuracy of AIRS. the practical interval, from the ATC operational view, was set at an hour and was so designed into AIRS. Entry of capacity data is therefore by hour and AIRS provides an assortment of input forms to facilitate entry (see Ref. 7). The data is stored in the airport data file in two sets of 24 elements - one for normal and one for today data. Retrieval and review of the values for an airport involves simple access of the block of data associated with the airport. Typical use of landing capacities can be seen in chapters 10.2 and 10.3. In general, landing capacities vary because of severe weather and wind conditions; otherwise they normally will remain at a predictable level all day long, any day. Except for noise abatement or operating hour restrictions, the normal capacity is usually constant. It is during problem periods of the current day that the estimates vary from hour to hour. AIRS retention of a 24-hour set of airport arrival capacity data (for the normal and the today categories) thus provides sufficient resolution and scope for predicting airport arrival delays.

DEPARTURE DELAYS

The third kind of data, airport departure delays, is the most complex of the three. Departure delay data entry was intended as a crude substitution, avoiding the programming of a complex departure delay prediction capability similar to the arrival delay prediction processing (Chapter 10.2). The selected approach is crude because it delays all the flights by the same amount during the hour. There is also no regard for changing departure order when delayed flights overlap the following hour's departures. The principal objective was to propagate major airport departure delay effects to the associated destination airports. This approach explicitly applied to current day delays which occur in the ATC system and their effect on future arrivals. With regard to design value, this departure delay data, and the need to factor it into the AIRS traffic predictions has never been used operationally. Its use and this approach should be questioned by any follow on system. There are no normal values for this kind of data. The data is processed similarly to the today values of landing capacities. That is, it is specified in hourly intervals covering a 24-hour period and the data is retained in the associated airport data block in the airport data file. The complexity is due to the additional processing which accompanies the entry of airport departure delay values. A design decision was made that AIRS would be more efficient in the use of departure delay information if the delay effects were recorded directly on the affected flights as a special set of time and date data. Thus the flight schedule file needs updating as well as the airport data file. This decision traded the one time update of departure delayed flights, for the repeated checking and updating of flights each time they are retrieved. The process of updating the schedule file involves the computation of the delayed departure and associated arrival time for each affected flight. Therefore, AIRS makes an internal request for the departure flights from the delayed airport (see Chapter 7). After assembling the flights for the period having departure delays, the special set of delayed times is added to each flight, along with a special date flag indicating the presence of this special set of data (meaning it is to be used in place of the original schedule times) on that date. The flights are then written on the schedule file with the original and the special data.

DAILY RESET

This brings us to the daily reset operation. If, during a day, today values for GA, landing capacity estimates and/or departure delays were entered into the AIRS data base, the values will be in the data base the next day. These values (the next operational day) must be reset to the normal values consistent with the new day. The process for doing this is straightforward. When AIRS is run, it checks a date and time on the beginning of the airport data file to see if it is within the current operational day. If it is not, it checks two flags in the file which indicate that on a previous day some airport data was entered. One flag indicates whether the airport data file must be reset, the other indicates whether the schedule file must be reset. These flags are automatically set any time airport data is entered. When required, AIRS resets all the today values in the airport data file to normal values for GA and landing capacities and removes all departure delays. Similarly in the schedule file, AIRS resets the date flag to show that no departure delay data exists for the previously affected flights. After reseting both files as required, the current date and time are written on the airport data file. This inhibits any further attempts to reset until the next day when the recorded date and time is not within the current operational day.

The daily reset also does the resetting of the schedule file the day following AFCP procedure implementation. When AFCP ground delays are issued (AIRS updates the schedule file accordingly), the delay data is stored on each flight record as a special set of data similar to departure delays. This data must be reset the next day and is done at the same time the above resets are accomplished.

11.2 ENTERING AND RETRIEVING FLOW CONTROL DATA

The processing requirements for entering flow control data into the AIRS centralized data base are quite simple. The process involves a dialog between the user and the computer. To initiate the dialog the AFCP, Quota Flow or arrival delay prediction request must include a key word indicating that editing of the control data is desired. After the flight data retrieval but prior to the data procesing, the user is asked what he desires to edit for that airport. He may enter or alter the control parameters and/or the zone structure files (discussed in Chapter 5.3). The program allows the user to select just the areas to be edited; it does not sequence through the full set of editable items. For example, he is given a choice of any one of several areas he may edit or he may go on with the request processing. Choosing an area, he is asked to enter the desired value(s) of the parameter(s). Then he is returned to the choice question. Some choices produce a subset of dialog until the editing is completed. Zone structuring is of this kind; the program will inquire if you are editing the old zone file or replacing it with a new zone structure. It then conducts a dialog for the entry of the zone: the control status for the zone, the individual places and groups of places in the zone with associated boundary time estimates. One can recycle within the zone editing dialog, entering and replacing other zones for the airport. Listings and printouts of the control values and zone structures can be obtained during the editing phase to aid the user in determining the flow control conditions for the airport.

Entering and retrieving this data is handled by reading the control and zone files (if any) into core, allowing the desired editing and then writing the altered files back on the disc as desired. The copy in core is retained after completion of the editing to be used in the subsequent flight processing for the original AIRS request. If the files do not exist for the airport at the time of edit, the program begins with a default set of cntrol parameters (and no zone structure). It will create the control file when the editing is finished and the user chooses to establish it as a permanent file for the airport. Contrasting with this, if the user edits zones for an airport, the zone file is automatically created and retained for that airport (i.e. the user has no choice of suppressing permanent retention of zones).

12. ARO OPERATIONS

There were two major design considerations in the implementation of the ARO operations. One was to retain the same entry format (Chapter 4.1) as the predecessor system; the other was to expand the updating operations to apply, as desired, to any airport in the AIRS data base. ARO operations can be divided into two categories, entry of new reservations and cancellation of existing reservations. The ARO of the SCC currently provides airport reservation management for 4 high density airports during peak traffic hours. The scheduled and general aviation traffic information they handle is the input data to AIRS. In addition, the airlines and the airline scheduling groups provide limited quantities of flight update information for a broad range of airports. This data is entered into AIRS when available.

The entry of data is performed in real time, soon after it is received in the ARO. As mentioned in Chapter 4.2, entry of these flight updates can use both keyboard and paper tape modes at the convenience of the users. AIRS treats the two entry modes differently, however. The keyboard mode operates interactively and will ask for clarification as needed to complete an entry; the paper tape model will reject without further processing any entry which is unclear. The users have favored the paper tape entry mode because it frees them from waiting for AIRS to process an entry before they can enter the next. The keyboard mode is used most frequently for correcting entries. The paper tape is easily prepared off-line (not connected to the computer) at the user's convenicence and when entered into the computer, its rate of entry is automatically controlled by the ccmputer, requiring no user supervision.

The kinds of requests entered into the computer during ARO operations were discussed in Chapter 8.4. Basically, a reservation or cancellation request must contain adequate information to complete the operation or an error will be noted. The form of these entries, as previously mentioned, was inherited from the predecessor. A key result of this inheritance is a constraint on the date for which a flight can be entered. Because the format did not provide for a month identifier, but only the day of a month, AIRS treats the day as falling within the range starting with the past day up tc three weeks in the future. This range provides for data entry which might be left over from yesterday, while older entries are rejected because they seem too far in the future. But it does not allow ARO to enter longer range (over 3 weeks) data for future schedules. This is not an important restriction since the bulk of the ARO data

falls within the allowable time period.

The format of ARO entries also provided for optional entry of aircraft type (equipment) with each reservation. The omission of aircraft types in the AIRS flight schedule file was not consistent with the existing (at the time of ARO implementation) AIRS method for retrieval of flights by type. In order to remain within the AIRS type structure, an ARO reservation entry without specificied type is automatically classified as type "NONE". A beneficial spin-off from this design decision, since ARO rarely enters a type specification, was that retrieval of just the ARO-entered flight data is easily accomplished by specifying type "NONE" in the retrieval request.

Another AIRS incompatibility occurred with entry of unknown airports (i.e. unknown to AIRS are airports not serviced by the airlines). Since AIRS was primarily structured around airport categorized data and retrieval methods, it was impossible, without major AIRS changes, to accommodate unknown airports. A decision was made to provide AIRS with the knowledge of 21 artificial airport codes (mnemonically similar to the ARTCC codes), one for each ARTCC. The artificial airports can be used in place of the unknown airports in the ARO reservation entries and should be associated with the same ARTCC for flow control reasons. The small number of unknown airports (since AIRS knows over 1200) makes it easy for the user to replace the unknown airports by the appropriate artificial airport codes.

In general, the design approach for implementing ARO operations into AIRS has received the most exhaustive scrutiny of any added feature. First and foremost, updating the central data base as a frequent real-time operation substantially increases the problems of assuring file integrity despite computer hardware and software failures or communication noise and disconnects. A considerable amount of extra processing and careful ordering was applied to provide fail-safe operation (discussed in detail in Chapter 13.6). The second area of scrutiny concerns the speed and efficiency of updates. In the first AIRS version with ARO implemented, a space saving approach had been followed in the reuse of flight record disc storage space. New entries could be written upon previously cancelled entries. But this approach required resorting of the associated cross-reference files, a very time-consuming operation. After accumulation of several weeks of operational experience, it became obvious that improved response time was needed to handle the daily updates more rapidly. At the cost of more rapidly growing files, the elimination of

reusing the flight record space made it possible to remove the re-scrting processing from the cross-reference files. Specifically, the new flight's cross-reference pointer would always be appended (since the flight is appended) and thus maintain the sorted order. A considerable improvement in response times resulted. Many less significant improvements were pursued to reduce the cost of entry operations, the major contributor to operating cost. In the following discussion of reservation entry another design change will be discussed which also resulted in substantial response time improvement and the associated cost savings.

RESERVATION ENTRY

The information supplied in a reservation request is listed in Chapter 8.4. At this point, in accordance with a previous user instruction, AIRS either skips to the date entry preparation (discussed in the next paragraph) or conducts a redundancy check to see if the reservation might already be entered from a previous request. If checking is required, the correct date (AIRS appends the appropriate month) and the time of operation is then used to formulate and process an internal AIRS request (Chapter 7) which checks for redundant entries. This internal request produces a list of flights previously entered for the airports and time period (for improved checking the entire hour of departure is requested). A quick check of the list is performed to avoid an erroneous duplication attempt. In interactive mode, the user has the option of entering the flight anyway. Multiple users handle ARO entries and receipt of multiple data could result in duplicate entry attempts. This necessitated redundancy checking to maintain file integrity. This checking is costly in both dollars and response time. Recently, the procedures were changed to improve manual redundancy checking. Associated with this change, AIRS was modified to make its internal redundancy checking optional when inputting reservations using the paper tape mode. Use of this option (not to check the data base before entry) has significantly improved the ARO operation in AIRS with the associated savings in time and money.

Before the flight is entered on the schedule file, AIRS computes the day flags and effective date range, encodes the airports and aircraft type and packs the data in the form for writing on the schedule file. AIRS also performs a check of the departure airport to see if there has been a departure delay entered for the hour of this flight's departure. If there is a delay, it is entered into the flight as described in Chapter 11.1. The flight is then appended to the schedule file and the pointer (location of the flight on the schedule file) is added to the

13.2 TRUNCATION

The monitor maintains a table on the disk for each file on the disk telling where to find its parts (a file needn't be contiguous). If you create a file, its table is also created, and if you append to a file, its table is expanded to encompass the new data. The monitor does not write out the new table when you write on the file (if you appended one word at a time, this would in effect double the writing), but waits until you close the file. If the system crashes after you have created a file, but before you have closed it, the table will not have been written. Although you have written data, there is no knowledge of its existence, so the file is lost. Similarly, if the system crashes after you have appended to an existing file, but before you have closed it, the appended data is inaccessible, so the file is in effect truncated. There is no problem if the program is interrupted by other than a crash (for example, the user may interrupt it), since in this case the monitor closes the file.

If truncation (or file loss) occurs, there is no way to recover the missing data. It is therefore necessary to take steps to prevent such damage. The precautions used for the files appended to by AIRS are described below.

USAGE RECORD

Section 15.4 described the usage record, on which AIRS records the requests it receives, problems it encounters, and other information. This file is appended to throughout the AIRS session. In order to minimize loss of these records after crashes, the file is periodically closed and reopened. At worst, the last few items written will be lost, but this is not critical.

SCHEDULE, ETC.

When an ARO entry (reservation or cancellation) is made, some or all of the five files discussed in 5.1 are appended to.

Schedule file: A new flight is added to the end of the schedule file.

Cross-reference files (terminal and center): If there is no room to add a pointer for the new flight, the cross-reference may have to be moved. If there is no free space to move it to, it is appended to the end of the file.

Free-space file: New free areas are added to the end of the file unless a free area has been used up and can be overwritten.

Truncation of any of this appended data would be very serious, and in fact intolerable, requiring (in some cases) backing up the files to the previous night. For example, if a newly moved cross-reference block were lost, there would be no way to access the flights for that airport. The old cross-reference would no longer exist, and the pointer for that airport would point to the nonexistent block past the end of the file. Any attempt to use this pointer would yield garbage results. The schedule and free-space files have similar problems.

But how can we prevent truncation? There would be no truncation if there were no appending. So we eliminate appending. This is done by keeping the files extended with dummy zeros and keeping track of the logical end of the file (the end of the real data). Now when we append to the data, we are nct appending to the file, just modifying the zeros. Figure 5-1 showed this expansion room on the end of these files, and also showed where we keep track of the "end" of each file. The schedule has on its start the number of flights, and each free-space file has on it the number of items it contains, and the the address of the "end" of the associated cross-reference file. The files are created with some expansion space, then checked each night (see 13.7) by comparing the stored length with the file length, and extended if the remaining space falls below some threshold.

AIRPORT DATA FILE

When a data block is created for an airport, it is appended to the end of the airport data file, and the airport's pointer (on the beginning of the file) is set to point to it. If the new block is lost, the pointer will point past the end of the file, causing garbage to be retrieved. Worse yet, the next data block appended will go in the same place, so two airports may point to the same data, messing each other up.

Truncation of this file is very unlikely. Entry of data for new airports is very rare, and the operation so short, that a crash during the entry will probably never happen. A crash during an ARO entry, on the other hand, is very likely (and has happened), since hundreds of entries are made each day. We have not, in fact, taken any precautions with the airport data file. It could be handled by the method described above - keeping the file extended and keeping track of its "end" - and perhaps AIRS should be modified to do so.

13.3 CHECKSUM ERRORS

In order to guard against errors in accessing the disk, the system (time-sharing monitor) computes a value called a checksum from data in a file and stores it separately from the file, in the index it uses to access the file. Whenever it accesses a file, it computes the checksum for the accessed data and compares it against the checksum stored for that data. A mismatch is a signal that something is wrong - perhaps the disk hardware accessed the wrong place, or the file has somehow been clobbered. When you update a file, the checksum may be affected. However, the system does not write the updated checksum each time it needs modification (this might double the amount of writing), but rather waits until you close the file. Thus if a file is written on but not closed, it may have updated data but the old checksum, which is inconsistent with the new data. If a program which is writing on a file is interrupted before it closes the file, whether or not there is a bad checksum depends on the type of interruption. If the interruption is due to a system crash, there will be a problem. However, after any other interruption (such as the user interrupting the run, the phone connection shutting off, etc.) the system closes the file, so there is no problem.

The ARO spends enough time updating the data base, and crashes occur often enough, that this type of error is a real problem for AIRS. If a checksum error in the data base is not dealt with, it causes trouble in two ways. First of all, AIRS will fail when trying to access the data in question. Secondly, the backup copy of the file (see 13.1) will be bad. The system program which saves files on magnetic tape stops if it encounters an error, so the backup copy of the file is truncated! AIRS and the nightly fail-safe procedure contain provisions to correct the first problem and prevent the second.

The random access software used to access the data base automatically transfers control to an AIRS error location when it encounters a file error. If a checksum error occurs on a main data-base file (the schedule, the cross-references, the free-space file, the airport data file), it is fixed as follows. Since we know the data is correct, but has an incompatible checksum, we want to force the system to compute and store the new checksum. To do this, we simply rewrite the troublesome data (although an error occurred, the data was read in). Although we haven't changed the contents of the file, we have written on it, so the system recomputes the checksum. After fixing the error, AIRS checks the whole file, in case other parts need fixing. It then quits, telling the user to rerun AIRS. If a checksum error occurs on a scratch file, it needn't be fixed. AIRS deletes the file, quits and tells the user to

rerun AIRS. When it is restarted, fresh copies of the scratch files are created (this is the procedure of 13.4).

To avoid truncation of files on the backup tape, the six main data files are read through just before the backup tape is made each night. If checksum errors are encountered, they are fixed as in AIRS. Thus the backup copies will be complete.

13.4 FILE ERRORS

As described in 13.3, control is transferred to a special error location in AIRS if a file error is encountered. If the error is on a scratch file, AIRS simply deletes all scratch files and quits. When it is restarted, it creates fresh scratch files, so the problem is gone. To prevent bad scratch files left over from crashes from causing trouble, AIRS routinely deletes and recreates the scratch files when it starts up. If any error other than checksum (13.3) is encountered on a non-scratch file, AIRS hopes that it's spurious. It can't handle the error, so it simply quits. If the error was spurious, it should not reappear when AIRS is rerun. If there is a real error, it will simply remain until TSC can do something about it.

13.5 DATA CHECKING

AIRS originally worked under the assumption that its data files were good, blindly using whatever pointers it found there to tell it where to locate or write data. If for some reason the data base was damaged, AIRS might not only produce bad reports, but might propagate the damage by using a bad pointer to write data in the wrong place, destroying other data. This actually happened soon after the ARO started using AIRS. A bug in the random access software made it write data in the wrong place, and the damage propagated as described. The extent of the damage made it impossible to figure out what damage happened first, hence to figure out what the cause of the problem was. It was not until we managed to catch the damage at an early stage that we were able to diagnose it. To prevent such a situation from happening again, checks were added to AIRS to detect bad data, record any damage found on the usage record for analysis, and refuse to continue operating with the bad data.

There are three benefits gained from having these checks in AIRS.

1. Bad data is not used to perform requests, so the user is not given garbage results.
2. Trouble is detected earlier, so the data base can be backed up sooner and less wasted work is done.
3. The cause of the problem is easier to determine, since the evidence is not allowed to wipe itself out.

The checks in AIRS, however, are not sufficient. Since AIRS only tests the data it is trying to use, a damaged area may not be discovered for several days. There may be no way to know when the damage occurred, hence no way to determine how far to back up the files. If they are backed up several days, all modifications made during those days are lost. To avoid this problem, checking programs are run each night when the files are saved (see 13.7), performing reasonableness checks similar to those in AIRS.

NIGHTLY CHECKS

The following checks are made each night. Any discrepancies found are reported.

The schedule file is spot checked (one item in each flight is examined); this would turn up any large-scale damage to the schedules.

The lists of pointer ranges on the cross-reference files are checked; each pointer must be positive, and the pointers in each list must be in increasing order.

The free-space files and cross-reference files are checked for consistency. No area on the cross-reference file should be listed as both free (on the free-space file) and used (on the index on the start of the cross-reference file).

Of course, these tests are not comprehensive. More exhaustive tests (such as checking that each flight is correctly cross-referenced) would be too time-consuming. The tests we have implemented are sufficient to catch damage like any we have experienced.

AIRS CHECKS

The following checks are made in AIRS. If a test fails, AIRS quits. If the error was spurious, hopefully everything will be OK when AIRS is rerun. In any case, since the action is aborted, no further damage will be done.

Flight Retrieval

The pointer ranges being used to retrieve flights (chapter 9.1) must make sense. Each pointer must be positive and not greater than the number of flight schedules, and the first pointer of a pair must not be greater than the second.

Flight Entry

When a new flight is added to the schedules, its pointer is added to the cross-references. The pointer must be greater than the last one already on the cross-references (since the flight is appended).

If there is no room to add a pointer to a cross-reference list, the list is moved to a bigger space, as found from the list on the free-space file (chapter 5.1). On these occasions, the following precautions are taken.

1. The address to write on must be legitimate (positive, and at most equal to the end of the file) and the number of words to copy must look reasonable.
2. To provide a double-check that areas listed on the free-space file really are free, AIRS fills free areas with -1 before listing them. Before copying onto a free area, it reads what's there to be sure it is -1.
3. While copying the cross-reference, AIRS checks it to be sure it looks reasonable. The pointers must be positive, not ridiculously large, and in increasing order.
4. After the cross-reference is copied, but before the airport's pointer is changed to indicate the new copy, the newly-written information is reread and checked as in

3 above. Thus if something goes wrong during the writing, the original cross-reference is still the one in use.

This appears to add a lot of extra work to ARO entry processing - extra writing (to fill free areas with -1), and extra reading (to check free areas for -1, then check waht was written). However, cross-references have to be moved in only a fraction of all flight entries. Considering the amount of change being made to the files at these times, it is well worth the extra work on these occasions to ensure file integrity.

Errors relating to free space (as detected by tests 1 and 2 above) are self-healing. AIRS removes free areas from the free-space file before attempting to make use of them, rather than after using them. thus it will not try again to use the bad area (it no longer thinks it's free), and may work the next time it is run.

The checks in AIRS, like those run at night, are far from comprehensive. Much data is not checked for reasonableness, and other items written are not rechecked as in 4 above. We did not want to add too much overhead of rereading and checking to AIRS, but just put in tests where necessary to catch the kind of damage that had been encountered.

13.6 PROGRAM INTERRUPTION

Sections 13.2 and 13.3 covered two types of file damage that could result if the computer crashed while AIRS was modifying its data base. It was pointed out that neither kind of damage would occur if AIRS was interrupted by any means other than a crash, since in all other cases the files are closed, and those types of damage result from the files not being closed. Some possible causes of program interruption are: AIRS quits after detecting file damage (see 13.3, 13.4, 13.5); the telephone connection is broken; ths user stops the program, perhaps because he has accidentally requested data he doesn't want, and doesn't want to wait for the answer. If AIRS is interrupted while in the midst of an operation which involves several related data base modifications, a data inconsistency could result, due to some of the changes being done but not the rest. These inconsistencies vary in seriousness. Some have been taken care of in AIRS, others have not. In some cases, inconsistency can be prevented by writing items in a particular order, such that the files are never left vulnerable. In other cases no order can ensure data integrity, so special procedures must be adopted. For example, recovery data can be recorded before making the dangerous series of modifications such that if the operation is not finished, this can be recognized, and it can be completed later. In deciding what series of writes are vulnerable, we have been extremely cautious, always assuming the worst. No matter how unlikely an interruption is between two writes (for example, two consecutive words being writen one after another), we have considered it possible. The rest of this section will describe some of these vulnerable situations and what is done about them.

In some cases, simple care in the order in which things are done can prevent any inconsistencies from arising.

1. Daily resetting

Section 11.1 described the daily resetting of the previous day's temporary data, if any. Stored on the airport data file are the date and time the reset was last performed and flags telling what data needs resetting (the airport data file, the schedules, or none). When a modification is made to the data base that will require resetting the next day (entering landing capacities, general aviation factors, or departure delays or implementing AFCP controls), the flag indicating the need for a reset is set before the modification is performed. Thus even if the modification is not finished, the reset will be done. When the daily reset is performed, the flags are not reset or the date/time recorded until the reset is completed. Thus if it

is not completed, it will be redone when AIRS is rerun.

2. Pointers and Counts

A new data item should be written before any associated pointers or counters are updated to include it. Otherwise a nonexistent item might be pointed at. For example: A new flight entry is appended to the schedule before the number of flights is updated; A new pointer range is appended to a cross-reference list before the count of the number of words on the list is updated; A new data block is written on the airport data file before the airport is given a pointer to it. With the correct order of operations, the worst that can happen here if an activity is interrupted is that it simply may not have been performed. That is, the new item is not available.

3. Free-space files

The free-space files keep track of reusable areas on the cross-reference files by listing the address of each free area and the number of words in it. There are two possible types of inconsistency that can exist between a free-space file and its associated cross-reference file - holes and overlaps. By a hole we mean an area on the cross-reference that is really free, but isn't listed as such. This is not serious; all that happens is that some space is wasted. By an overlap, we mean an area that is not free, but is listed as free - i.e. a "free" area overlaps a used area. This is serious. Earlier versions of AIRS would write on the "free" area, clobbering good pointers. Section 13.5 described the double-checking that prevents this from happening. Free areas are filled with -1, and will not be used unless they are. Moreover a "free" area that doesn't look free is removed from the list, so it won't cause trouble again. Overlaps should still be avoided, however. AIRS only removes from the list the part of an area it is trying to use, so if a large overlap exists, it may take many time-consuming failures before the mess is cleaned up. We thus trade off overlaps in favor of holes. The order of operations in free-space manipulations is always chosen such than an interruption will at worst result in a hole, never an overlap.

In other cases, careful ordering of operations is not sufficient to prevent inconsistencies if the program is interrupted.

1. AFCP and Departure delays

The implementation of AFCP (10.3) and the entering of departure delays (11.1) both result in writing delayed times into large numbers of flights in the schedule file. If this

is interrupted, some flights will have the delays and others won't. The data will be inconsistent, and the user may not be aware of it. If he asks what departure delays exist (11.1), for example, he will be told either the old or new delay (depending on whether we update the schedules or the airport data file first), although some flights have one and some have the other. Even if nothing is done to correct this inccnsistency, it will automatically be fixed the next day when the daily reset removes all delays. Repeating the aborted request, allowing it to run to completion, would also remove the damage. We could have made AIRS fix up the inconsistency automatically by having it record (on some special area of the schedule file, perhaps) what it was about to do before doing it. Then when it next tried to use the schedules, it could recognize that the operation had not been completed and either undo or finish it before making use of the schedules. We have not bothered to implement such a recovery scheme because neither the AFCP nor delay feature of AIRS has proved useful. In general, though, recovery should be provided for such situations.

2. ARO entries - flight reservations and cancellations

We saw in chapter 12 that making a reservation entails creating a new entry on the schedule file and adding it to the associated cross-references, while cancelling a flight can entail either simply modifying an existing flight entry or both modifying an entry and creating a new one as in the case of a reservation. We will see in this section what inconsistencies could arise if the program were interrupted (no matter what order related changes are made in) and what has been done to prevent them. Such interruptions are not unlikely, since ARO uses the system so heavily, so they must be prevented from doing harm.

Reservation: If the flight is added to the schedule before it is cross-referenced, then an interruption could result in the flight being on the schedule but only being accessible by one of its airports (or perhaps by airport but not be center). If it's not in the cross-reference for its airport, it could erroneously show up in a request for traffic which excludes its airport (when the cross-reference is complemented, see 9.1). For example, if a flight departing Chicago did not get added to the cross-reference for Chicago departures, then it would show up in a request for arrivals at New York from all places except Chicago. If, on the other hand, the flight is cross-referenced before being entered into the schedules, similar problems could result. The cross-reference for its airport(s) and center(s) could point to an unfilled spot on the schedule file. The next reservation entered would take that place, so the cross-reference would point to a flight it shouldn't.

For example, if the interrupted entry were for a Chicago departure and the next completed entry were for a New York departure, the New York flight would show up in the answer to Chicage departure requests. Thus we see that both orders are unacceptable. The solution makes use of the "purge flag" in each flight entry (described in 12, for cancellations), which if turned on, prevents the flight from being used in the answer to any request. Reservations are made as follows: the flight is entered into the schedules, but with its purge flag turned on; it is then cross-referenced; and finally the purge flag is turned off. It thus becomes retrievable only when it is properly cross-referenced.

Cancellation: We saw in 12 that there are three different cases in cancelling flights. Cancelling a single-day flight entry simply involves turning on its purge flag. There is nothing here to be interrupted, since the whole operation is a single write of a single word. Cancelling a flight on the first or last date of its date range requires modifying its effective or discontinued date and removing any delay data if necessary. This involves rewriting several words of the flight entry. An interruption (very unlikely, of course) could leave the entry partly rewritten and possibly inconsistent with itself. To prevent this, we write the modified entry along with the address in the file where it belongs in a special "recovery data" area on the start of the schedule file. Only then is the modified entry written on top of the original one, after which the recovery data is removed. Every time AIRS opens the schedule file, it checks the recovery area. If it finds that recovery is needed, it writes the specified flight entry at the specified address, then removes the recovery data. The cancellation is thus completed before AIRS will use the schedules. The last case is the cancellation of a flight for a date in the middle of its effective date range. This involves splitting the entry in two, modifying the old one to be effective only until the cancellation date, and creating a new entry effective for the rest of the period. This is a combination of the processing required for reservations and the above case of cancellations, and the safety techniques used for both are required. That is, we must modify the old entry, using recovery data to ensure its consistency, and create and cross-reference the new entry, using the purge flag to make sure it is not retrievable until it is cross-referenced. However, this is not sufficient. If the old entry is modified but the new one not completed, then the flight will have been cancelled for all dates from the desired date on, since the modified entry only goes up to the date being cancelled. If the new entry is made but the old one not modified, the flight will still exist (in the old entry) for

the date being cancelled, and will appear twice for all dates from the cancellation date onward. It is thus necessary to ensure that either both operations (modifying the old entry and creating the new) happen or neither happens. The method used is as follows. The new entry is created as for reservations, but its purge flag is left on. Up to this point, if AIRS is interrupted, nothing has been changed. Next the recovery data is written as before - containing the modified entry and its address - with one addition, the address of the new entry which needs its purge flag turned off. The modified entry is then written and the new one made available, after which the recovery information is removed. When AIRS finds recovery necessary (as above) it performs both of the required operations.

13.7 NIGHTLY FAIL-SAFE PROCEDURE

It was explained in 13.1 that a batch job is run each night to make a backup copy on tape of all our files. Other parts of the chapter introduced programs that are run at night to fix or check the AIRS data base. This section simply summarizes the nightly procedure.

1. Asks the operator to mount the tape.

2. Protects the schedule file to prevent modification of the data base during checking and saving (see chapter 14).

3. Checks the six main data base files, fixing any checksum errors encountered (see 13.3). This prevents truncation of the files on the backup copy.

4. Checks the lengths of the schedules, cross-references, and free-space files, and extends them if the allocated space is getting used up (see 13.2).

5. Checks the contents of the schedules, cross-references, and free-space files as described in 13.5.

6. Copies all files onto tape.

7. Removes protection from the schedules, so the data base can be modified by AIRS.

8. Asks the operator to dismount the tape.

9. Resubmits this job for the next night.

14. MULTIPLE USER MANAGEMENT

This chapter discusses the conflicts that can arise when more than one user is running AIRS or using the AIRS data base and how these conflicts are handled. Priority of one user over another is also discussed.

There are three possible combinations of simultaneous users:

1. More than one user could be running AIRS. The ARO and the CFCF each have a computer terminal, and are frequently logged in at the same time. Moreover, each could be logged in on more than one terminal (if available). At the same time, TSC could be using AIRS.
2. Someone could be using AIRS while the data base checking and backup programs (chapter 13) are running.
3. AIRS could be in use while a new data base (incorporating the next month's data) is being prepared (chapter 18).

AIRS/AIRS

With two AIRS jobs running at once, we must be concerned with conflicts in their use of both the central data base and files created by AIRS. AIRS routinely creates several files while it runs: the usage record (15.4), containing a record of the AIRS session, is written on throughout the session and remains until it is typed out and deleted by TSC; several scratch files, created when AIRS starts, are used to hold intermediate data during processing, and deleted when AIRS quits. Obviously, if two AIRS's are trying to record their progress or store their temporary data (from different requests!) in the same place, there will be trouble. Thus whenever AIRS runs it must create and use files which are separate from those currently in use by any simultaneously running AIRS. This can be done if the file names, rather than being fixed (programmed into AIRS), can be assigned by AIRS to be unique at the time it computes them. The monitor assigns each user a "job number" when he logs in. Although job numbers are reused when the user logs off, they are unique at any given time. AIRS thus assigns names to its scratch and usage files which contain the job number (of the job running AIRS), and thus do not conflict with other users.

Considering the variety of AIRS activities and the different uses they make of the central data base (some write, some only read), we should expect conflicts in simultaneous access to the data base. Some activities will not conflict because they use different files. For example, if one user is retrieving an airport's traffic demand (reading the cross-reference and schedule files) while another is updating landing capacities (writing on the

airport data file), there can be no problem. For activities which use the same file, three combinations can occur: both are just reading, both are writing, or one is reading and one is writing.

1. read/read

Any number of users can read a file simultaneously. This is not a conflict situation.

2. write/write

The monitor does not allow two users to have the same file open for writing at the same time. An attempt to open a file for writing which is already open for writing by another job will cause an error. As in 13.3, AIRS can recognize the error condition and retain control. It can't do its job until it gains access to the file, so it must wait its turn. It goes to "sleep" (having the monitor reactivate it after a specified interval, such as a second) then tries again. This cycle (trying and sleeping) continues until AIRS succeeds in opening the file. Sleeping is valuable because just trying over and over wastes computer time, which not only costs money, but means less time available for the competing job to finish its activity and release the file. There is one other problem that must be dealt with when more than one user wants to write on more than one file - the problem of "deadly embrace". We will explain this by an example. Assume that two users want to write on the same two files, F and G, and that user#1 opens F first while user#2 opens G first. User#1 is now unable to open G, since user#2 has it, so he waits. Similarly, user#2 waits for F to become free. They will wait forever (or until the computer shuts down) since each is waiting for the other's file and cannot proceed until he gets it. Deadly embrace can be avoided by establishing a standard order for opening files. Thus if all programs open F before G, the deadly embrace above cannot occur. This is the solution used in AIRS. To summarize: The central data base files needed for an AIRS activity are opened in a standard order. If a file cannot be opened for writing because it is busy, all files opened so far are released and the program tries again after a short sleep. Thus the activity waits its turn. This conflict management is invisible to the user. This write/write lockout means that having two jobs processing ARO tapes at once is no faster than running one, since they cannot both make entries at once.

3. read/write

Any number of users can read a file while one is writing on it. This can cause a data conflict. That is, if one is reading the same data the other is updating, the reader might get inconsistent data - some from before and some from

after the modifications. There is no simple, practical way to prevent such conflict in the current time-sharing environment. Simultaneous access, hence conflict, could be completely prevented by always opening the files for writing (even though the activity only reads). This would be an extreme measure, however, since it would prevent any AIRS activities (even two retrievals) from proceeding in parallel. A more reasonable scheme might involve communication between separately running AIRS's. When an activity started, it would record which (potentially conflicting) activities it wanted locked out. An activity would not begin if another one was currently requesting it not to.

It turns out that in AIRS this sort of conflict is very unlikely. Activities that make large changes, such as entering departure delays into the flight schedules, are rarely used. Frequent writing activities, such as entering flight schedules, make very local, very fast changes (except when moving cross-reference blocks around, which is relatively rare), so they are not likely to interfere with anything. Moreover, since the CFCF and the ARO are the only AIRS users, and use different AIRS features, they are not likely to be doing two conflicting things (such as entering landing rates for the same airport) at once. If AIRS were a system with many isolated users, this data conflict problem might be very serious and would have to be dealt with.

AIRS/FAIL-SAFE

Section 13.7 summarized the batch job which runs each night to check and maintain the integrity of the central data base. The necessity of preventing file modification during this procedure was pointed out. It is not sufficient to just schedule the job for a time when AIRS is not usually used, since in extenuating circumstances - such as the computer being down - the job can be automatically rescheduled for an arbitrary time. If the files could be held open for writing throughout the checking, AIRS would be prevented from modifying them (by the write/write conflict handling described above). However, the checking procedure consists of a sequence of separate programs, and there is no way to hold files open between programs. The solution makes use of file protection (see 17.2). The batch job starts by protecting the files against writing. If an AIRS user is currently writing on the files, the protection change fails and the job tries again later. Otherwise, it proceeds with its tasks, removing the protection when it is done. (Some of its tasks, as in 13.2 and 13.3, may require writing on the files. This is possible because the job runs under a different account number than AIRS does, and the files can be selectively protected against different accounts). If

AIRS tries to write on a file while it is protected, an error occurs, which is recognized as in the write/write conflict above. AIRS refuses to perform the requested activity (it could be a long wait!) and so informs the user.

AIRS/DATA PREPARATION

Each month a new data base is prepared, combining the next month's data with still current schedules from the current data base (see chapter 18). From the point when the desired schedules are extracted from the current data base until the new data base supersedes the old one (during which time cross-references, etc. are formed), modifications to the old schedules must be prevented, since they would not appear in the newly-formed schedules. This is accomplished by protecting the schedule file, exactly as described above for AIRS/Fail-safe conflict.

PRIORITY

AIRS was designed as a tool for the Central Flow Control Faciility, with the ARO data modifications enhancing its accuracy. If CFCF needs information, it should not have to compete with ARO for the computer's resources. There is no direct way, however, to give CFCF priority over ARO. Instead, when they really need fast response (for example, when doing quota flow), CFCF can include a key word in certain requests which causes AIRS to open the schedule file for writing instead of just for reading, hence locking out ARO until the CFCF activity finishes. Different sleep times (between attempts to open a busy file) are used for different activities to give CFCF a better chance at locking the file (short sleep times) and keep ARO out of the way longer (long sleep times).

15. RECORDS AND MESSAGES

AIRS has several features to facilitate the transmission of information between TSC and SCC. Communication from TSC to SCC includes a brief "message-of-the-day" typed out to users when they run AIRS, longer messages available to AIRS users upon request, and format descriptions of ARO requests available upon request. Communication from SCC to TSC includes the sending of arbitrary messages and the automatic recording of the use of AIRS.

15.1 MESSAGE-OF-THE-DAY

When you log in to most time-sharing systems (including First Data), a short message is typed out containing pertinent information about the system. For example, it may tell you that the system will be down for maintenance, give you a phone number to call for help, refer you to documentation of a new feature, etc. AIRS has a similar capability. When a user runs AIRS, the first thing it does is type out the contents of the AIRS message file (AIRS.MSG). Since the message is read from a file, not programmed into AIRS, it can be changed (or deleted) at will by simply editing (or deleting) the file. This message is used to tell the user the effective dates of available data, to announce a new feature, etc. It is only used for short messages, since it is typed each time AIRS is run. Users should not be burdened by unnecessarily lengthy typeouts, especially of information they have already received or that is irrelevant to them. If longer messages must be sent, they are sent via the MAIL feature (see next section).

15.2 MAIL

TSC may need to send SCC longer messages than can be handled by the above message feature. These may include descriptions of new features (which may run to several pages in length), further explanations of current features, further discussion of a point briefly stated in the short message, etc.

Messages are typed by TSC into a file called 'MAIL'. When AIRS starts up, it notifies the user if there is MAIL. To obtain a typeout of a long message from TSC, the user makes the request 'MAIL' to AIRS [REQUEST=MAIL]. When AIRS receives a request for 'MAIL', it reads this file and copies it onto the user's terminal. If there is no MAIL, it does nothing.

15.3 HELP

Need: Most users of a complex system such as AIRS will not know (or need to know) all its ins and outs. Those who use it infrequently or who make a kind of request they have never made before, are likely to make mistakes. AIRS error diagnostics cannot always be good enough to really explain what the problem is, and some requests may seem to work all right but not be doing what the user thinks they are doing. Although the user guide contains a complete description, it is not always convenient to look up what you want. It is desirable for AIRS itself to be able to help a user by answering his questions, explaining requests, etc. This need is partially met by AIRS as follows.

AIRS feature: When in the ARO section of AIRS, a user can make the request 'HELP'. AIRS then asks what kind of request he wants help with. If the desired help is not available, AIRS says so; otherwise AIRS types out the desired information. For example, the answer 'RA' types out a description of the RA request.

Implementation: Request descriptions are typed into files having first name the name of the request it describes, and second name the standard name 'HLP'. When the ARO secton is asked for help on a particular type of request, it looks for a file with that first name and last name 'HLP'. If it finds one, it types out its contents.

Limitations and Extensions: As of now, the HELP request can only be made to the ARO portion of AIRS. This is because ARO requests, having fixed formats, are easy to describe concisely. It would be desirable to make the HELP feature universally available in AIRS. For example, if you asked AIRS for help with 'DATES', it could describe the legal ways to specify a date, the types of requests that can contain dates, the fact that only one date can appear in a request, and so on.

15.4 USAGE RECORDS

In order to be responsive to SCC needs, it is imperative that TSC know how AIRS is used: how often requests are made, what features are used, what mistakes are made, how good the response time is, etc. The easiest, most comprehensive way to get this information is to have AIRS record it. Then users needn't keep such records themselves.

AIRS records information about its use on a file. It writes out every request it receives, the time at which it receives it, and the computer time used to execute it. It

flags errors and records any file damage it may encounter. The name of this usage record file contains the job number of the job running AIRS to ensure that records for simultaneous users don't interfere with each other.

15.5 COMMENTS

We encourage comments from SCC for two reasons:

1. To improve the AIRS system: If AIRS isn't working properly, we need to know the problem so we can fix it; If SCC finds something confusing, we would like to clarify or modify it; If they want a new feature, we can consider providing it.
2. To analyze the usefulness of AIRS and help in planning future systems, it helps to know why a request is made. Is it part of a training session? Is it needed to help deal with an immediate ATC problem? Is it for use in anticipating traffic loads for a holiday weekend?

Since it is not always possible to get in immediate touch with TSC when a problem occurs, it is useful to be able to transmit a message via AIRS, while it is fresh in the user's mind.

One type of AIRS "request" is the comment. When AIRS asks for a request, the user can type any text he desires, just by preceding it by a 'C'. This text is communicated to TSC through the usage record. The comment is simply recorded on the usage record, just like any other request. (See section 15.4). No other processing is done to it.

The above features are embedded in AIRS in keeping with our goal of making AIRS as self-contained as possible, insulating the user from the time-sharing monitor. Users should not have to know anything about the system except how to run AIRS; AIRS should meet all their data needs.

For example: If AIRS didn't have the MAIL feature, the only way to transmit long writeups (without resorting to the post office) would be to have the user type out the file containing the information. This would require him to be familiar with the monitor 'TYPE' command and to be aware that he must be outside AIRS (at monitor level) to do it.

16. DAYS, DATES, AND TIMES

This chapter discusses the handling of days, dates, and times in AIRS. In order to get a feel for the features provided, we will compare AIRS with the pilot system which preceded it (Ref.2).

In the pilot system, only the days of the week, not the dates, of flights were stored. Reports were thus for "typical" day traffic. Experience with the system showed, however, that schedules vary enough that information for specific dates was desirable. This data was available; the source of schedule data (Reuben Donnelley OAG tape) gives the effective and discontinued dates of each flight as well as the days of the week on which it flies. AIRS thus stores this information, and accesses schedules by particular dates. In the pilot system, since dates could not be specified, only one month's "typical" traffic could be available at once, so a sharp break occurred between months - e.g. on the last day of a month you could not look ahead even a day. In AIRS, the new month's data can be made accessible as soon as it becomes available, while the current month's data is still in use.

In order to select flights, the pilot system had to know the day and time period desired. Both of these had to be supplied by the user in each request for data. AIRS must know even more - the desired date as well as day and time - but it is not reasonable to require the user to supply it all. Certainly he should not have to give both a day and a date, since they are redundant, and the system should be able to make reasonable assumptions for unspecified data. If neither a day nor date is given, the current day is assumed; if one is given, the other is deduced (to go from days to dates, AIRS assumes that the next occurrence of that day was meant); if no times are given, a time period based on the current time is assumed (if the request is for the current day). In practice, it turns out that usually the times are given, but the date is omitted (most requests are for the current day).

The pilot system only allowed requests covering a single day, so wraparound time periods (passing midnight) could not be given, but had to be broken into two requests. This could be inconvenient, since a single local day (of interest to the ccntrollers) covers two GMT days (all times and dates in the computer systems are in Greenwich Mean Time), so wraparounds would frequently be desired. AIRS allows wraparound time periods, though it does not allow more than a 24-hour period to be given. It breaks up the time into two time ranges, deducing the day/date for the second part from the given (or defaulted) day/date for the first.

In order to provide the features just described - day/date/time defaults and wraparound time periods - AIRS needs the following.

1. Wraparound time period - Assuming the day and date of the start of the time period are known, AIRS must be able to get the day and date of the end of the time period. Thus it must be able to increment a date.
2. Date given, day not given - AIRS must be able to figure out what day of the week a given date falls on.
3. Day given, date not given - AIRS assumes the next occurrence of that day is meant. Thus it must get the current date and day, then increment the current date by the amount needed to get from the current day to the desired day.
4. Neither day nor date given - AIRS must know the current date and day, which it uses as default values. Actually, it need only know the current date, since the day can be deduced from it as in 2.
5. No times (or only one time) given - AIRS must know the current time.

This boils down to three basic abilities: knowing the current date and time, incrementing dates, and converting from dates to days.

Current date and time:

These can be obtained from the monitor (time-sharing system). Unfortunately, the monitor works in local time - Eastern time, since it's located in Massachusetts - while the flow controllers (hence AIRS) work in GMT. Thus AIRS must convert from local to GMT. This requires knowing the time difference, which is either four hours (during daylight time) or five hours (during standard time). Each time AIRS is run, it decides which time difference is applicable by getting the current date, deducing the current day, and deciding how they relate to the last Sunday in April and October. This time difference is then applied each time AIRS needs to get the current time. Note that the date might also have to be incremented.

Incrementing dates:

All that's needed here is a table telling the number of days in each month. It's easy to decide whether it's a leap year, hence the number of days in February. A side effect of having such a table is the ability to recognize and reject illegal dates. This table is actually combined with the one described below.

Converting Dates to Days -

This could be done by storing an initial date and its corresponding day, then computing all others in terms of it.

By calculating the number of years, leap years, months, and days between the base date and the date of interest, the corresponding day could be determined. AIRS doesn't need much range of knowledge, since its data spans at most two months including the present. It thus keeps a table for the past six and next six months, giving the first day in each month. Then the day on which a date falls (within that 12-month period) can easily by determined from the first day in the month of interest. This table is kept on a file (described in 5.4) containing the month and year about which the data is centered, then for each month, the number of days in it, its first day, and the year in which the values hold. The file was originally typed in, but is automatically kept up-to-date by AIRS. When AIRS starts up, it reads the table, and if it isn't current (centered around the current month/year), updates it to the present and writes out the new copy.

A side effect of date-day conversion is that if both a date and day are typed, they can be checked for consistency.

The need for the abilities just discussed was derived from their use in traffic load, or demand, requests. Of course they are used throughout AIRS, in many aspects of its processing. A few more examples would be:

The current time and date are used each time AIRS starts up to decide whether it needs to reset temporary data - i.e. whether it is being run for the first time that day. See 11.1.

A reservation request gives a date but not the day. The day of the week must be determined in order to create the flight entry.

The dates in the schedules refer to departures. Thus when arrivals are being retrieved or canceled, they must be tested or canceled for flight on either the specified date or the previous date, depending on the individual flight - whether it arrives on the same day it departs.

We have said that all dates, days, and times in the data base are in GMT, and that AIRS and its users communicate in GMT. The OAG tape, however, from which the schedule file is produced (see chapter 18), gives the days and dates in local (for each flight) time, though it states the departure and arrival times in both local time and GMT. Thus we must convert the dates and days to GMT in preparing the AIRS data base, deciding for each flight whether the GMT day is the same as, before, or after the local day. The tape provides no explicit information on the time difference (whether ahead or behind GMT) or hemisphere, so the answer must be deduced from the relationship of the GMT and local times.

For example, consider a flight whose departure time is 2200 in local time and 0300 in GMT. We know that all places in the world are within twelve hours of GMT, either ahead or behind. Thus this flight's origin must be 5 hours behind GMT, since it can't be 19 hours ahead, hence its GMT departure days are one day later than the local ones listed.

17. AIRS PROGRAM ENVIRONMENT

Back in chapter 1, an introductory statement was made that AIRS was scoped to perform within a time-shared computer's resources. More precisely, the AIRS software has been significantly influenced by the hardware and software environment of the First Data Corporation's PDP-10 computer system. The AIRS development has capitalized upon the beneficial resources of the First Data Corporation service and to a lesser extent, suffered from resource limitations of their computer environment. This chapter addresses the major environmental factors in the AIRS development.

As discussed in Chapter 2, the scope of the automation required sufficient core memory to accommodate a complex program and its associated data arrays. In terms of the PDP-10's 36-bit word structure, up to 35,000 words of core memory per user was available. Although more core could be effectively utilized, this amount was deemed adequate to support AIRS. The following section, 17.1, covers the design effects of the 35,000 word core memory sizing and the supplementary use of disc storage.

The time-shared PDP-10 system is endowed with an extraordinary set of machine instructions and monitor features which give application programs considerable control over the efficiency and scope of program execution and applicable system resources. The software benefits are described in the 17.2 section of this chapter, as they apply to AIRS utility operations and as they vitally manage the system resources. Also discussed are the special system software packages which control the random accessing of disc files and the operations of the Conograph-10 graphical display terminal.

The remaining factor in addition to the hardware and the operating system is the programming language (s) used in the application. As stated in Chapter 2, FORTRAN was selected as the principal high level language for AIRS. Its selection was greatly influenced by the established wide support of this language by most of the time-sharing services. This permitted the broadest consideration of computer sources for the AIRS development. The FORTRAN language supported on many time-shared computers included: interactive input and output during program execution, which is essential for real-time operations, flexible input/output formatting ability for effective user interface control, and a comprehensive repertoire of logic, arithmetic and control operations for ease in programming. It was recognized at the time of language selection that no single language could completely and efficiently fulfill the AIRS programming

requirements. FORTRAN best fit the anticipated needs and could easily interface with assembly level programs in areas where it failed to provide adequate efficiency.

17.1 CORE MEMORY VERSUS DISC STORAGE

The First Data Corporation PDP-10 system has considerably more core memory than the 35,000 word portion available to each user. The restriction is impposed for practical reasons to provide reasonable service when several users are on the system. If more core were made available to each user, more time would be required swapping user programs in and out during their share of each second of computer operation. This time is overhead, non-productive to the user and degrading to the general performance of the system. The 35,000 word core limit was the maximum offered for this type of service. Restricting AIRS to this size necessitated using supplementary disc storage, the inclusion of special space saving provisions and the overlaying of active parts of the program upon inactive parts during various automation operations.

The use of supplementary disc storage is a common method for expanding the size of programs and/or data which can be processed within a fixed amount of computer memory. Overlaying, or the use of disc storage to expand program size, is discussed in 17.2. This section addresses data expansions involving core and disc trade-offs. Data retained in core during processing is normally structured into blocks referred to as arrays. Since available core has a limit, data arrays must be limited in their share of core. By storing these arrays in part or as a whole on a disc storage device and by bringing into core just a portion of data as needed, it is possible to accommodate much larger data arrays than the core alone could handle. The price of this expanded data handling ability is increased processing time, because it substitutes the slower disc access of data for the thousands of times faster core memory access. This expanded data approach may not be applicable for all data processing requirements. For example, if the processing requires frequent hopping around the data array, the associated volume of disc accessing might be impractically slow. In effect, each piece of data used would require a new disc access if outside the portion of the array in core.

There are two types of data expansion situations involving core versus disc storage. One type is data which has a known size, the other has a variable or unknown size. When the size of the data block is known, it is readily evaluated with regard to fitting in core. If it does, one

must further consider the relative value of whether it should be in core in part or whole. An example of this value judgment is seen in the AIRS dictionary (see Chapter 8.1). It was originally retained in core during the early version of the prototype system. With the addition of expanded AIRS features, the need developed to use this core for other more frequently used data arrays. The dictionary, which is used only during a small fraction of an AIRS operation, was relegated to disc storage. A minor addition of a routine to access the dictionary on the disc was then substituted for the original dictionary array in memory. If however, the known data block is too big for retention in core, the trade-off is that of determining how much of it to buffer in core. This trade-off can be very complex. It is possible to retain in core specific elements of data from every line (set of elements) of data in an array. For example, today's or the nominal landing capacities (Chapter 11.1) for an airport may be brought into core while the remaining airport data is not. If the processing requires only these specified elements of data and their number is a practical size, the pertinent data may be readily accommodated in memory. But if the data involves all the elements in each line of data (e.g. a complete flight schedule), the data will require buffering (i.e. transferring a segment of this data in or out of core to disc storage). The solution necessitates the inclusion of array swapping processes and disc accessing to control and maintain the program's own buffering operations. In this situation the storage size of the peripheral devices is important. No data block in the AIRS prototype approaches the capacity of a disc storage device. In essence, this program controlled buffering can handle all of AIRS large data blocks whether the size is known or unknown.

In this same manner, the second type of data expansion, the unknown data size, can be treated. However, one further consideration for unknown data blocks might be knowledge of typical data block size. Although the upper size of a block of data may be unknown, it may be known that the typical data block falls within a size which can fit in core. If this is the case, allocating this typical size amount of core for the data swapping will improve processing response times in general, since no disc accessing will be required when processing the typical data blocks. AIRS employs this last technique in data arrays used in the analysis of requests (Chapter 8.3).

An alternative in dealing with unknown size data blocks is to limit the data to a maximum allowable size. For example, if it is only remotely possible that a block of data reach a very large size, the program could be

restricted in processing the data up to a practical size. If the limit is exceeded, it does not perform the process at all (i.e. it might simply return an error message stating that the operation can not be performed). For example, the current method used in controlling listings of detailed flight data employs this techniques. A limit was set of 100 arrival flights and 100 departure flights which could be listed per data hour in response to a list operation (Chapter 10.4). This limit was imposed to avoid the programming complexity of buffering this data during assembly and sorting processes. When the program employs this limit, it does not process the listable data for the offending hour but it does continue to process all other hours which do not violate the 100 flight maximum. The 100 flight limit was determined to be sufficient to accommodate all but the rarest of listing requests, which output data collectively for a group of airports. This restriction was a minor sacrifice for the considerable savings in increased program complexity associated with buffering provisions.

As discussed above, there are a multitude of trade-offs in the design of AIRS which depended directly on the 35,000 word sizing. The mix of using memory and disc storage is the primary approach for handling data of essentially unlimited scope. The price is slower processing because of the proliferation of disc accessing. To reduce this proliferation of disc accessing, additional means were employed to fit more data into core memory by capitalizing on packing and coding techniques. Packing and coding were adequately discussed in earlier chapters. Currently AIRS has up tc 13,000 words of data in core at any one time. On the average this data is packed far greater than two pieces of data per word. If it were only 2:1 the unpacked version would require 26,000 words of core memory and would be an untenable situation because it leaves insufficient core for the program which manipulates this data.

In general, the restriction of AIRS to 35,000 words of memory has necessitated special treatment of data and extreme concern for efficient use of core. Designing the program and its data to operate in this restrictive environment was the only practical approach for the timely development of a flexible and broad scoped prototype automaticn system using a commercial time-sharing computer.

17.2 MCNITOR AND SPECIAL SYSTEM SOFTWARE

The scope of operation and efficiency of AIRS draws heavily upon the software support provided by the monitor and system software of the First Data Corporation's PDP-10. This software is enhanced by the addition of RAX, a random access disc servicing package (developed by the Bedford Group) and CONOPACK II, the Conograph-10 display terminal servicing package (developed by Conograph Products Division, Hughes Aircraft Corporation). Before discussing these special packages, three system support areas will be outlined: 1) overlaying, 2) system data and 3) file support.

As mentioned in the previous section, the use of supplementary disc storage is a common way of expanding the size of programs which can be processed within a fixed amount of memory. The PDP-10 provides such expansion capabilities through its chain loader. The loader structures the oversized program in accordance with the programmer's directions into a resident portion (i.e. always in core memory during program running) and overlays (i.e. an overlay is a portion of the program which can be dynamically brought into core from disc storage as needed to carry out its processing function). The resident program controls the overlaying, transferring control and data to each overlay. It also cntains data and routines needed in more than one overlay. This common data provides communication between overlays. The first version of AIRS fit within the 35,000 word core; the current version of AIRS requires six overlays. Figure 17-1 presents the overlay structure of AIRS in terms of the major activity of each part and the user's interface. The partitioning into the six overlays was designed to minimize the frequency of overlaying during any automation operation.

The monitor provides AIRS with a variety of system data during the course of each run. The data falls into two categories, program job information and status information. The first category involves conveying to the AIRS program the user's job number and the program name. The job number is used in creating (i.e. naming) non-conflicting scratch and data files in the multi-user environment (see Chapter 14). The program name is required for control of the proper overlay file. AIRS, designed to evolve during its operational lifetime, requires a backup in the event of new version failure and exists in two forms available to the users. The newest version is named AIRS, the predecessor is renamed BACK. Requestng its program name from the monitor eliminates the need for re-programming the BACK version where it would be running the AIRS (instead of BACK) overlays. Both BACK and AIRS use the same data base.

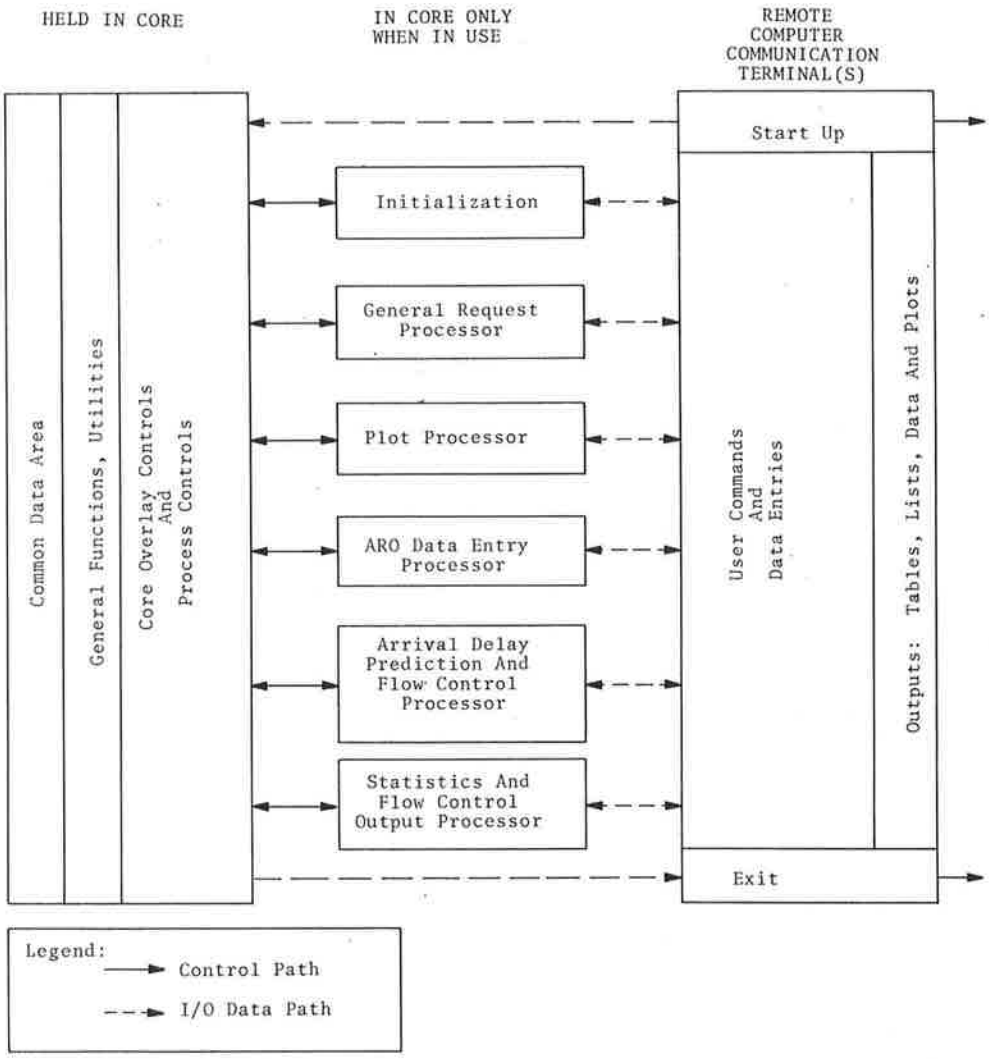


FIGURE 17-1 AIRS OVERLAY STRUCTURE

The second category, status information, pertains to time and date data. When a user makes a request to AIRS, the program uses the current time and date information in recording the usage and, as necessary, in processing the request. The system makes this data available on call from AIRS. In addition, it can provide the computer processing time on call. AIRS uses this information is recording the computer time (i.e. the user's share of the computer) and the elapsed time (i.e. the user's wait time from request to response) with each request or major operation (see Chapter 15.4). Although not specifically status information but more status changing, AIRS also uses a sleep request to the monitor. The monitor activates AIRS after inhibiting all opeations for the requested seeping period. This enables AIRS to take turns using common data files by waiting briefly between attempts to use them (see Chapter 14), thus assuring the opportunity for other users to access them.

The last area of system support cncerns the PDP-10 file structure and support software which has been very important to the management of the AIRS data base. In general, files are named and can be created, deleted, updated and accessed through user programs. Files may also be renamed by a program or protected against undesired reading or writng access. This latitude in file management has significantly benefitted AIRS and its supporting programs, the nightly (Chapter 13) and monthly (Chapter 18) processing activities. For example, usage of AIRS would not be practical without the creation, use and deletion of scratch files to accommodate the large data processes. If the scratch files could not be deleted by the program, their proliferation would soon exhaust disc storage space and prevent further processing. Another example involves the monthly updating programs, which run with a minimum of human intervention because of file management control within the programs. Files are created under temporary names, predecessor files deleted, and the newly created files renamed, replacing the deleted files during various upgrading stages.

Accessing the files, as well as writing on them, requires supporting software to provide the bookkeeping for the device (disc addressing), physical location of the data, checksum error checking and the read, write mode switches. AIRS uses a proprietary software package, RAX, to access and update its disc files. RAX permits files to be accessed in two modes: 1) read only, 2) read and/or update (write). The monitor permits simultaneous access of the files by any number of users with the condition that only one may be updating the file. In addition, under the update mode, RAX provides either deferred or immediate write options. Some explanation is required to clarify the significance of this

option. RAX is a random access file management system which works within the physical and monitor constraints of the First Data computer system (monitor). From the user's point of view, random access is in terms of randomly addressable words in a file (i.e. a file is viewed as a one dimensional array of 36-bit words). To the monitor, a disc file is one or more blocks, each 128 words in length. RAX maintains a 128-word file buffer invisible to the user (programmer), no matter how many words are being read or written. RAX is efficient enough to know when the current buffer contains the word or words of interest and avoids any unnecessary disc accessing or updating. However, in cases where an update of one or more words must be immediately written on the file for fail-safe reasons (see Chapter 13.6), RAX provides the immediate write option. In this mode the 128-word buffer is written on the file each time an update is processed instead of the deferred write which waits until the writing fills the 128-word buffer (or writing is required in another 128-word block) before physically outputting the data to the disc file. AIRS thus controls the efficiency and fail-safe management of its file updating to best fulfill the needs for file integrity and system response. RAX also provides error recovery capabilities through file error trapping. When an input error is detected by the monitor, RAX transfers control to the user's program for dispositon of the error condition (for example, see Chapter 13.3).

The final special software support concerns the operation and control of the Conograph-10 display terminal used for AIRS graphical outputs. The terminal, being a graphical display device, requires special software when drawing plots and pictorial displays for AIRS. The proprietary Conopack-II software package is the primary interface with the terminal for producing such displays. This package provides byte manipulation, special control stream formulation and input/output buffering for display operations.

18. MONTHLY DATA BASE FORMATION

Chapter 1 mentioned that AIRS is more than the program which the SCC staff operates. You've seen the nightly programs required to maintain file integrity and provide fail-safe operations (Chapter 13). The remaining group of programs to be discussed performs the monthly update operations. These programs are currently run by the TSC staff on a convenient day between the receipt of the R.H.Donnelley data tape (about 15 days prior to the effective month) and the first day of the month covered by the tape. The entire monthly update can be accomplished within a four hour period during a normal working day. TSC often does the monthly updating during off-peak hours (after the normal working hours) where processing times are closer to two hours and the interference with SCC operation is reduced.

The monthly update operations can be divided into six distinct phases. The phases are themselves composed of several steps (normally separate programs) which process the required portion of the data. Figure 18-1 outlines these six phases. This chapter will describe the processing steps in each phase in sufficient detail to understand the basic processing for monthly updating. No attempt is made in the document to examine all of the design details of these support programs. To aid in visualizing the scope of data processing and file development, Figure 18-2 gives a schematic overview of the file formation sequence, the parts of which will be discussed in the subsequent paragraph.

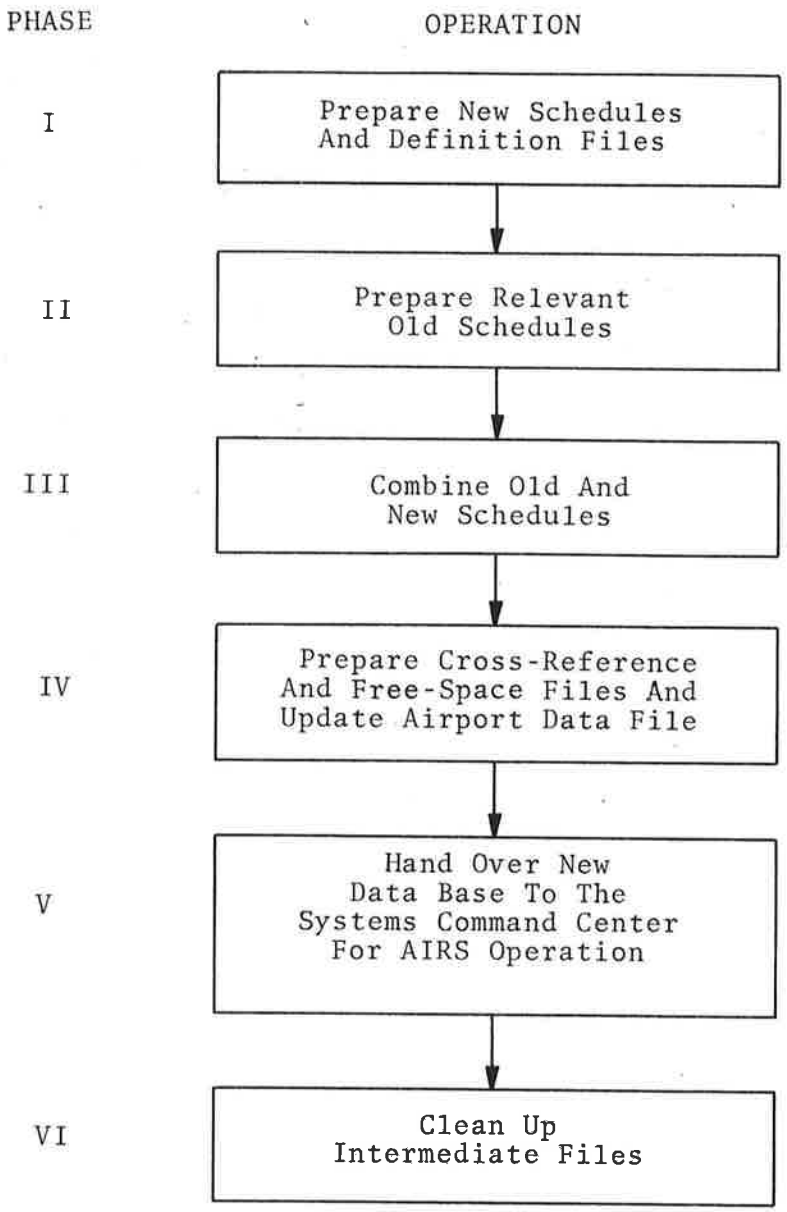


FIGURE 18-1 MAJOR MONTHLY UPDATE OPERATIONS

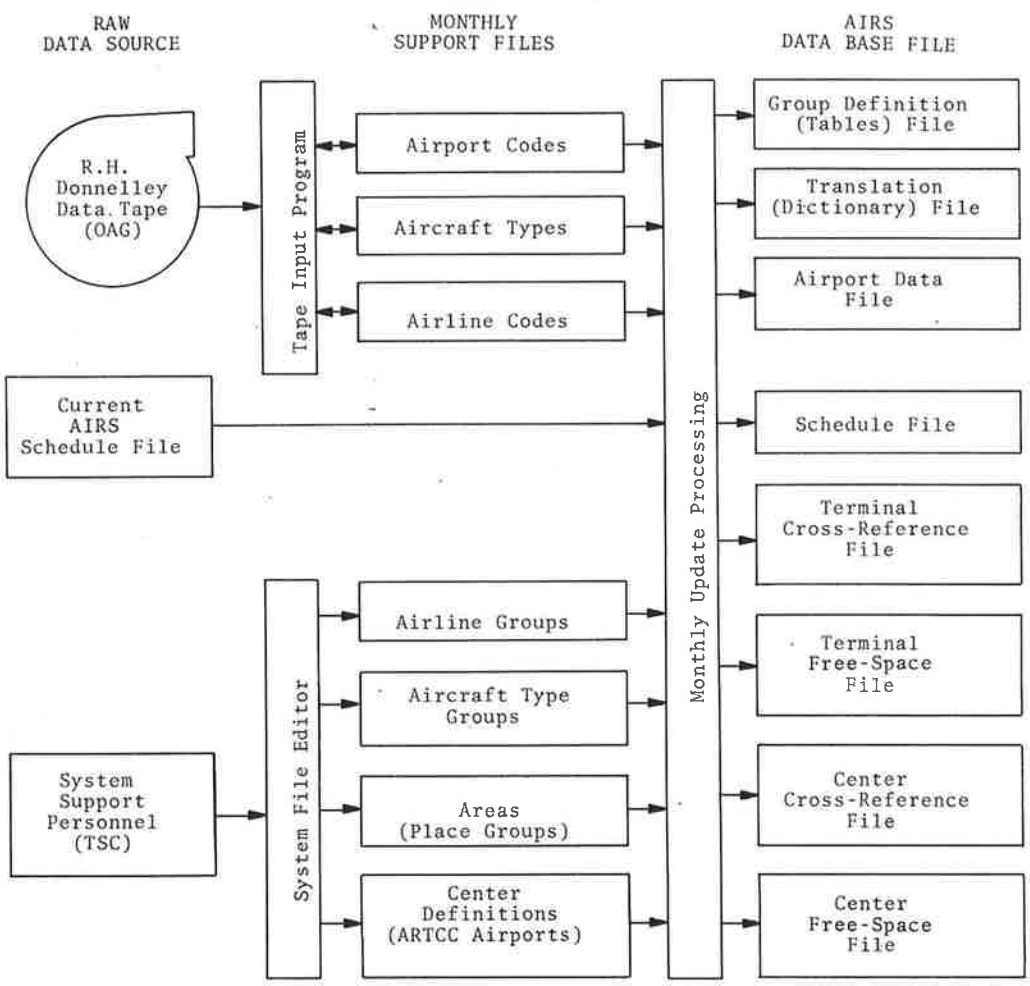


FIGURE 18-2 MONTHLY FILE FORMATION OVERVIEW

PHASE I

This phase requires the bulk of human interface in preparing the new schedule file and the definition files. After the data tape is received at the computer installation, the first step of phase I is to read through the data tape checking block sizes. A PDP-10 system program is used for this purpose because it efficiently verifies the quality and integrity of the data tape and tape reading device. This step is required because there have been frequent bad data tapes and/or malfunctioning tape readers. The earlier such problems are detected the sooner they can be rectified, either by request for a new tape or maintenance of tape reading hardware.

The processing then proceeds to step 2, reading in the flight records (approximately 60,000) for screening and processing. Step 2 performs many operations in reading the flight records, it:1) converts the record from IBM format (EBCDIC) to PDP-10 format (ASCII), 2) rejects helicopter flights, 3) rejects foreign airport to foreign airport flights, 4) fills in start and end dates (effective period) for flights not having supplied start/end dates - default period supplied by operator, 5) converts departure day flags from local to GMT days (see chapter 16) and produces the arrival day flags (in GMT), 6) writes the accepted flight record on a temporary schedule file, 7) appends any new airports, airlines, or aircraft types to the existing cumulative files of these codes, 8) reports these new code additions, 9) saves status information after every 1000 flights have been written on the temporary new schedule file to facilitate restart if computer malfunctions, 10) generates file of statistics with regard to approximate numbers of airborne flights, 11) adjusts arrival times by ten minutes earlier, and 12) reports the number of rejected flights (by category) and the number of flights (about 20,000) in the new month's schedule.

Step 3 involves the use of a PDP-10 system file editor. The support operator must edit the four group and definition files shown on Figure 18-2 to incorporate any additions of airports, airlines or aircraft types into the appropriate groups or definitions. For example, a new domestic airport code must be added into the set of airports defined as being within the associated ARTCC. This is mandatory to assure accurate center (ARTCC) cross-reference.

Step 4 produces the translation (dictionary) and the group definitions files for AIRS usage (chapter 5.4). The files are assembled from the updated support files (those produced in steps 2 and 3). In entering each code into the

dictionary, appropriate categories and values are included as shown in table 8-1. Group definitions are set up as described in 5.4. Values are assigned to airports, centers, and aircraft types consecutively as they are read in. Since step 2 only appends to the support files, the assigned values are compatible with the current data base. These two finished (updated) files are immediately available, superseding the old in AIRS usage since they are compatible.

The fifth step begins the automatic sequence of programs which carry the monthly updating through phase IV without human (operator) intervention. If an error or failure occurs during this automatic sequence, the process may be continued from the last completed phase after corrective action (if required) is effected.

Step 5 performs the final encoding and packing operations on the flight schedules. It reads in the first temporary version of the flight records, uses the dictionary codes where applicable (replacing airport and aircraft type names by their code values) and forms the packed flight record which is written on a temporary new schedule file.

Step 6 is simply the process of sorting the temporary new schedule file into arrival airport order, subsorted by departure airport. This step concludes the phase I processing.

PHASE II

The second phase of monthly updating involves recovering the current month's schedules from the old (current) AIRS schedule file. This phase is divided into two steps. The first step starts by protecting the current schedule file (see Chapter 14) against change during the remaining file update activity. The file is then read, flight by flight. Each flight is checked for effective date range and for being cancelled (purge flag on, Chapter 12). If the flight is more than 15 days old (15 day data retention is required for ARO records), it is rejected along with the purged flights. Those flights not rejected are written cn a temporary old schedule file.

Step two of this phase is like step 6 of phase I. The temporary old schedule file is sorted by arrival airport and subsorted by departure airport.

PHASE III

This phase is a one step operation, merging the temporary old and new schedule files into a single, sorted (maintaining arrival and departure sorts) file which is given a temporary name. This file will become the new schedule file after the subsequent phases of processing are completed. The file is extended in length to accommodate growth (Chapter 13.2).

PHASE IV

Phase 4 has four steps in it. It handles the development of the cross-reference and tree-space files and the updating of the airport data file. In the first step, the terminal cross-reference file is formed by reading the new schedule file and grouping pointers for all arrivals by airport and for departures by airport. Step 2 does the same pointer grouping except that it is by arrival center (ARTCC) and by departure center. This requires use of the center definition file, since centers are not included in the flight schedules. The terminal/center correspondence derived from the center definition file is also used to update the centers on the airport data file (5.4). Step 3 sets up new, empty free space files for the terminal cross-reference and the center cross-reference files These free files simply contain the pointers to the ends of their respective cross-reference files. The last step of this phase is to extend these files to allow for safe growth (see Chapter 13.2). This is the last phase and step of the automatic update sequence.

PHASE V

This phase involves a simple set of file renaming operations, performed at the operator's command. The renaming is automatic (under program control) and establishes the conditional schedule file and its associated cross-reference and free files as the new AIRS data base, while saving the superseded files under temporary names as backup files.

PHASE VI

The last phase in monthly data updating is the clean-up operation. This housekeeping activity deletes all intermediates temporary files which were deliberately left until now in order that restarts or recovery could be effected if any portion of the update sequence went astray.

TRUNCATION

Although not associated with the monthly update, a procedure has been employed by TSC to aid in speeding AIRS response. The procedure employs the appropriate monthly update programs in truncating the data older than 15 days, at a time when truncation will remove the entire previous month's R.H.Donnelley data. The process takes on the order of one hour and simply involves starting the monthly update process at phase II step 1. When the phase III, merging activity is initiated it simply finds no new schedule data and passes the old, now truncated and re-sorted, data on to the cross-reference and subsequent processing. The results is better AIRS response times for one week to ten days while AIRS has this smaller schedule file (half the normal size).

19. RECOMMENDED IMPROVEMENTS AND EXPANSIONS

Eighteen months of operational experience has been accrued on the AIRS prototype automation system. During this time, AIRS has evolved from an information retrieval system to a system which predicts airport arrival delays and provides traffic quotas for conducting nationwide flow control operations. Its data base evolved from the accuracy of monthly updates to the accuracy of real-time daily updates (when both the need and the data availability permit). This evolution reflects almost continuous improvement and expansion toward making AIRS more productive and useful to the System Command Center. Expansion ideas, generated during these eighteen months, far exceeded the available of time to evaluate, design and implement all of the improvements. In addition, studying the detailed operation of AIRS has resulted in the formulation of alternate methods of treating and processing some of the AIRS activities. The alternatives are directed toward improving the efficiency and response time by tuning AIRS to operate more harmoniously within the characteristics of the host computer. Many of these improvements have been made; many have sufficient potential to warrent future consideration. This chapter strives to identify and characterize the nature and the payoff of the major expansion ideas and the suggested areas of improvement.

The order of presentation of these recommendations conveys no meaning of relative importance. Each recommendation must stand on its own merit in the light of current user's needs, implementation difficulty and available resources. In general, any change or addition should strive to minimize response time; this characteristic of AIRS operations can not be overstressed. Poor response time will distract from AIRS operational value in the SCC on the time-shared computer. In a time-sharing computer environment, the elasped time between request and answer can often degrade to more than fifty times that of a dedicated computer. Especially worthwhile is the benefit of design complexity for the sake of reducing disc accessing. This added complexity is almost always justified in AIRS, as it is in most information based systems, because AIRS is input/output bound (i.e. limited not by the computer processing speed but by the effective peripheral storage and retrieval speed).

IMPROVED ACCESSING SPEED

AIRS is required to maintain two months data for continuity of operations between months and for preservation of records for ARO entries the previous fifteen days. This results in accessing schedule file data covering the two month period whenever an AIRS retrieval is required. The most practical approach conceived to date for reducing ths access is to add a date flag to the cross-reference pointer ranges (packed in the pointer word to avoid expanding storage and core requirements). The date flag (a single bit) will indicate if all the flights pointed to by the range have discontinue dates earlier than yesterday. Thus when cross-references are used by AIRS, about half of the accessing of the schedule file can be avoided during the majority of each month (i.e. up to the last week, when next month's data is present), by testing this flag for date applicability. Maintaining these cross-reference flags can be easily accomplished by an added nightly program which scans the cross-reference files and checks them against the records which became a part of the older group since the previous night. The affected pointer flags are then set. This recommended improvement should reduce the response time for a normal traffic request approximately 30 to 45 percent.

WHEELS DCWN TIMES

Currently AIRS incorporates a ten minute earlier adjustment in arrival times during the monthly entry of flights from the R.H.Donnelley OAG tape. This adjustment produces an approximation of wheels down time and produces better flow contrcl arrival loading results. However, for secondary AIRS applications, such as the checking of airport reservation quotas, the hourly counts produced by AIRS are inaccurate with regard to gate time counts. AIRS can be modified to compute the wheels down times from the gate arrival times when retrieving the flights, thus eliminating the recording of the wheels down times on the schedule file in place of gate arrival times. Airport reservation counts could then be accurately produced by specifying (by key word) that gate time of arrival be used.

ARO ENTRY RECOGNITION

ARO entries during the day are regularly retrieved by requesting for the airport of interest all flights with aircraft type "NONE". This works only if ARO does not enter the aircraft type. Future needs may require both the entry of aircraft type and the continued retrieval of just the ARO entries. It is possible to do both by adding a special flag to each ARO entry. AIRS can then be expanded to recognize a

keyword in a request which screens the flights for the ARO flag.

ARRIVAL DELAY AND FLOW CONTROL REQUEST EXPANSION

AIRS only computes arrival delay predictions and flow control data for the current day and for all arrival traffic. The need has arisen on occasion for the application of these AIRS computations for other than the current day or for traffic loads excluding some flights (such as occurs during an airline strike). The appropriate AIRS routines can be modified to recognize the non-current day status of the request, use qualifying traffic conditions entered with the request and produce the desired outputs. To simplify AIRS modifications, the today values of landing capacities (for the airport of interest) would be used no matter which date was requested. These capacities are readily changed by the user to suit the situation and can be reset after the non-current day outputs are produced.

PRIORITY AMONG AIRS USERS

A priority scheme for controlling the use of central data base files between simultaneous AIRS users can be implemented through the use of PDP-10 feature making a portion of AIRS common to all copies of AIRS running at any one time. Flags and priority data can be stored in this common memory area and periodic testing of this data by each copy of AIRS could control and interrupt lower priority operations. For fail-safe purposes, the approach involves a concept of challenge and response, where the highest level priority operation must periodically answer the challenges of lower priority operations (other users) in order to continue its priority use of the central data base. This prevents the situation where lower priority activities might be permanently locked out if the high priority activity were interrupted and could not release its claim.

CLASS CODE USE IN REQUESTS

AIRS records with each flight record its ARO class code (e.g. AQ, AT, GA). Although the class code can be specified for listing (by INFO requests), AIRS currently does not allow requests for traffic counts using these class codes to specify such retrieval. The only difficulty in adding class code recognition is the conflicting meanings to AIRS of these two letter codes with airline codes. A simple solution might be to require a longer code specification word (e.g. add a period to the code, AQ. or AT.). The user can then retrieve specified classes of flights such as air carrier, air taxi and general aviation.

EXPANDED HELP

ARO users can request AIRS to help in identifying the acceptable forms of flight entries. This kind of tutoring assistance can be expanded throughout AIRS. The refinement is minor, but it is voluminous since it requires many parts of the User's Guide (Ref. 7) to be incorporated into a file where it can then be accessed and output as needed. The AIRS modification is minor, involving the establishment of an expanded "HELP" request. The key word "HELP" is followed by one or more identifiers such as "PUNC" for punctuation usage help and "ENTER" for help in understanding modes of entering airport data.

COMMAND FILE

The SCC staff makes a standard set of requests to AIRS each morning and sometimes mid-afternoon too. The requests number about twenty and require the user to enter each in turn. The proposed approach permits the user to enter these requests into AIRS once, under a unique file name identifier. The user can then request by a keyword and the file name identifier, that this command file of requests be executed. AIRS will then automatically take each request in turn from the file, execute it and output the results to the user's terminal device.

LISTING SUBSETS

The user currently can list a specified hour or hours of data following AIRS demand type requests. Further specification in "LIST" requests is not permitted (e.g. if a demand request asked for all arrivals at JFK, the "LIST" request could not ask for listing just the MIA portion of the arrivals or just the J747 equipment). In general, the AIRS request analysis routines (Chapter 8) could be expanded along with the retrieval routines (Chapter 9), to operate on the scratch file flights and to further filter those flights to be listed by using added specifications in the "LIST" request.

LISTING ALL FLIGHTS

AIRS is currently limited to listing up to 100 arrival (and 100 departure) flights for each hour because of an internal memory array restriction. Removing this limit will allow combined airport traffic to be listed for all hours no matter how many flights are involved. The approach involves the use of disc file swapping to supplement the core memory array. The change is complex because the sorting of these lists, partly in core and partly on disc, requires careful

treatment to maintain efficiency and good response time.

LONGER DICTIONARY WORDS

Although ARO operations permit words longer than 5 characters to be recognized, the other operations of AIRS do not. The main reason is that the AIRS dictionary is unable to handle longer words. This constrains the AIRS requests to 5 character words with associated difficulty of working with less meaningful mnemonics when shortened words must be used. Increasing the dictionary word size to 10 characters doubles the permissible word size and the dictionary size. Since the dictionary is on a disc file, this size increase has little effect on storage, but it does require that the routines for parsing and word recognition be modified to input the longer words and more extensively, that all outputs including these words must be reformatted to handle them as well.

MULTIPLE WORD DEFINITIONS

The dictionary presently rejects any multiple definitions of words (Chapter 8.1). It is often possible to unambiguously determine the meaning of a multiply defined word when the context of its use is examined. AIRS can be upgraded to recognize such context meaning and thus permit the multiple word definitions. The modifications would expand the word recognition and syntax analysis sections of AIRS (Chapters 8.2 and 8.3).

ARO UPGRADING

The current ARO operations can be upgraded in many areas. The first would involve adding entry and cancellation modes which update in terms of flight schedules as opposed to single day flights. Thus a whole month's reservations for a daily flight could be entered in a single request and be processed in the same time as a one day flight entry. This upgrading involves a new entry format which recognizes effective data ranges and days of the week flags. The second area concerns the omission of information in reservation entries. For example, if one of the airports is unknown, AIRS will currently reject the reservation. If the data is valuable to the data base even when missing such information, AIRS should permit its entry. For AIRS to tolerate incomplete flight schedules, areas that use flight data must accommodate missing data without erroneous results. The third ARO upgrade involves giving AIRS the capability of recognizing airports not included in the OAG data tape. There are two approaches for doing this: one extends the encoding range to accommodate the greater number

References

1. "Technical Program Plan For Headquarters Air Traffic Service Automation", Office of Systems Engineering Management, FAA, Aug. 1971.

2. "Toward Developing An Improved Central Flow Management System", Richard Hakkarinen, Office of Management Systems, FAA, Oct. 1971.

3. "A Survey to Determine Flight Plan Data and Flight Scheduling Accuracy", John R. Coonan, Transportation Systems Center, DOT, Jan. 1972, DOT-TSC-FAA-72-10.

4. "Evaluation of the FAA Advanced Flow Control Procedures", J. F. Bellantoni, J. R. Coonan, M. F. Medeiros, Transportation Systems Center, DOT, Jan. 1972, DOT-TSC-FAA-72-8.

5. "Advanced Flow Control Procedures (AFCPS)", ORDER, FAA/DCT, Dec. 19, 1968, 7230.9A.

6. "Flow Control Procedures", ORDER, FAA/DOT, Nov. 21, 1972, 7210.7A.

7. "Airport Information Retrieval System (AIRS), User's Guide, Preliminary Version", Manuel Medeiros, Julie Sussman, Transportation Systems Center, DOT, Oct. 1972, (Unpublished).

8. "Advanced Flow Control Software Documentation", Technical Report, Vols. 1-6, The Mitre Corp., dates Jan. 1970-Aug. 1970, MTR-4109.

9. "LISP 1.5 Programmer's Manual", M.I.T. Press, 1966.

APPENDIX A

DESIRED FEATURES AND SELECTION CRITERIA FOR TIME-SHARING COMPUTER

Requirements for time-sharing computer service to meet needs of Air Traffic Flow Control Experiments under FA-206 (TSC/FAA Project).

1. Operating hours 7:00 a.m.-Midnight, 7 days/week.

2. Interactive FORTRAN (i.e. interactive during object program execution).

3. Random access capability for data file management under FORTRAN program control.

4. Private disc pack storage which can be mounted upon command from terminal (user).

5. Large interactive usable core area (after subtracting system utility needs) of at least 34,000 words of 36 bits each (or equivalent bytes, etc.).

6. Demonstrated capability to interface and fully operate over teletype 110 Baud lines in graphic mode on one or more of the following storage tube terminals: Tektronix T4002, Adage, ARDS 100B, Computek Series 400, Conographic Conograph/10, or equivalent storage tube display terminal.

7. Both 9 and 7 track tape drives for data input and output and associated data storage.

8. Accessability of the same account (including all programs and data files) from local call in Washington, D.C. and Boston, Mass.

9. Highly desired is the capability to back up main computer with another computer in event of malfunction, with the associated swapping of accounts and files and continuation of service with minimum interruption.

- Combined monthly update programs to generate new data base with less operator supervision.
- Decision made to procure a display device for AIRS graphical output experimentation.

Apr. 1972 - Added General Aviation factors to provide approximations of non-scheduled (non-R. H. Donnelley data) air traffic loads in AIRS airport demand reports.
- AFCP design specification changed to include the recommended improvements of TSC report on AFCP, (Ref. 4).
- Began AIRS file structure change to accommodate the AFCP data requirements in the central data bank.

May. 1972 - Completion of AIRS file modifications for compatibility with AFCP needs.
- Completed AFCP design decisions.
- Completed draft AFCP Input/Output design report.
- Began AIRS/AFCP program development.

Jun. 1972 - Added airport departure delay adjustment capability to improve accuracy of data during periods of severe departure delays.
- Contract awarded for purchase of Conographic-10 display device.
- Added airport landing capacities to AIRS capabilities and data bank.
- AIRS/AFCP automation operational June 29, 1972; conducted initial training at the SCC.

Jul. 1972 - Conducted several training sessions on AIRS/AFCP.
- Implemented 'TEST' option which provides airport arrival delay and stack size (holding in air) predictions.
- Added feature for stack status input with AIRS requests (for AFCP or TEST delay predictions) to adjust computed predicted stack size to know size at given time.
- Significantly improved the speed and efficiency of a major monthly update program.
- Concentrated development on ARO (real-time update) expansion to AIRS.

Aug. 1972 - Implemented the AIRS/ARO operation in the SCC's Airport Reservation Office.
- Conducted training on AIRS/ARO at the SCC.

Sep. 1972 - Began documentation efforts to prepare a preliminary version of the "AIRS User's Guide."

Oct. 1972 - Completed and distributed the preliminary "AIRS User's Guide."
- File damage detection routines were added to AIRS to record and suppress propagation of damage to other files and focus early attention on correcting file.
- Designed and implemented nightly file checkout programs to screen the AIRS data bank for any detectable damage.
- Began AIRS display development on leased ARDS display devices.

Nov. 1972 - ARO paper tape entry of real-time updates was added.
- Documentation of the AIRS program initiated.
- ARO updates were speeded by a factor of three by analyzing and trading off a less important operation, reducing the processing and disc I/O operations.
- Automatic file repair and recovery capability was added to AIRS to restore integrity of data files which suffered the most common form of damage, checksum errors.
- AIRS display of demand plots implemented on ARDS at the SCC.

Dec. 1972 - Added feature to recognize the class of service (general aviation, air carrier and extra sections) in detailed report requests.
- Added automatic file repair and recovery capability to the nightly checkout programs.
- Added new surveillance program to nightly programs which examines both data and program files for system detected damage.
- AIRS display of AFCP and delay prediction (TEST) plots implemented on ARDS at SCC.

Jan. 1973 - Began graphic display development on loaned Conograph-10 preparatory to purchased unit delivery.
- Preliminary design for converting AIRS/AFCP to Quota Flow procedure assistance was completed.

Feb. 1973 - Final design for converting AIRS/AFCP to Quota Flow operation was developed which would be additive to AIRS (no interference with AFCP use).

- Quota Flow automation implemented at the SCC.
- Training on Quota Flow automation coducted at SCC.
- Conograph-10 display device delivered to TSC for AIRS display experimentation.
- Initiated procurement of second display unit.
- Increased efficiency of certain AIRS operations.

Mar. 1973 - First Conograph-10 AIRS displays completed which provided various plots of traffic demand, delay predictions, AFCP and Quota Flow data.
- Modified AIRS/Quota Flow to provide, at the user's choice, controlled and/or original traffic reports.
- Modified AIRS/Quota Flow to improve man-machine efficiency through added dialog and program control options (based upon SCC AIRS operational experience).
- Developed fail-safe nightly recording programs for master files to provide a dependable back-up capability. (First Data fail-safe system did not assure collective file integrity because files could have been changed during the failsafing).
- AIRS enables the CFCF to shut off ARO file conflicts during priority operations such as Quota Flow procedure requests.
- Added nightly file extension program, a failure prevention measure which inhibits file truncation damage.

Apr. 1973 - Implemented a feature to reduce the processing for requests which do not require detailed listings of flight data.
- Introduced an option to inhibit the normal ARO flight data duplication checking, producing considerable cost and time savings.