AUDIT PROGRAMS
SOFTWARE DESCRIPTION
1A PROCESSOR

CONTENTS PAGE

CONTENTS PAGE

CONTENTS PAGE

FIGURES

TABLE

This description is based on the CPR2 generic program in issue as of the date of this section, but is also applicable for those offices equipped with the CPR1 generic program.

1. GENERAL

INTRODUCTION

1.01 This section provides the following information for telephone company personnel in switching offices equipped with a 1A Processor:

(a) A functional description of the 1A Processor Audit programs:

(1) SAWS—Writable Store Audit

(2) SAST—System Audit of Stores Using Tape

(3) SADK—Auxiliary Unit System Audit.

(b) A discussion of the software functions which are accomplished by SAWS, SAST, and SADK. No discussion is provided about the detailed encoded instructions contained in the programs.

(c) A discussion of the software interactions of SAWS, SAST, SADK, and the programs with which they function. This discussion includes the reasons for the interactions, descriptions of the data exchanged between programs at the interfaces, and a brief summary of the functions performed by the programs with which SAWS, SAST, and SADK interface.

SAWS and SAST perform the following functions:

(1) Check the integrity of the stored program and nontransient data

(2) Provide error information

(3) Provide corrective action.

SADK performs the following functions:

(1) Initialize the file store system

(2) Audit memory

(3) Time file store requests

(4) Audit Automatic Message Accounting (AMA) buffer pointers.

The program listing may use the term "disk" rather than the term "file store" which is used in this section.

1.02 When this section is reissued, the reason for reissue will be listed in this paragraph. The material in this section was formerly contained in Section 254-280-260, Issue B, September 1976.

1.03 Table A provides a list of abbreviations and acronyms with applicable terms used in this section.

PIDENTS DESCRIBED IN SECTION

1.04 This section describes the following program identifications (PIDENTs):

(a) 1A Processor Writable Store Audit Programs (SAWS) which consist of:

(1) SAWSBASE (PR-5A238)

(2) SAWSCMMN (PR-5A239)

(3) SAWSSUBR (PR-5A240).

(b) System Audit of Stores Using Tape Programs (SAST) which consist of:

(1) SAST (PR-5A237)

(2) SASR (PR-5A236).

(c) Auxiliary Unit System Audit (SADK) (PR-5A235).

PURPOSE OF 1A PROCESSOR AUDITS

1.05 The reliability of an electronic switching system is critically dependent on the integrity of the stored program and its associated nontransient data in the 1A Processor. This primary data base resides in a "protected write" area of writable store memory which contains both program stores and call stores. The protected write feature significantly reduces the probability of software error propagation, but there is still an ever present possibility of memory mutilation from software errors and hardware failures. Additional provisions are required for protection and maintenance of the software. The 1A Processor audits provide preventive and corrective maintenance. This is accomplished by checking the integrity of the stored program and nontransient data, then providing error information as well as corrective action.

1.06 Backup data consists of two file store copies of the generic program and nontransient data. SAST uses backup system tapes and buffers in the paging area in its task of auditing stores and making necessary corrections. SAST is manually requested to correct problems SAWS could not handle. Figures 1 and 2 are functional diagrams showing the relationship of the 1A Processor audits and their relationship with other software programs.

1.07 SADK performs audits of the DRR, DRB, CAB, and queue data to ensure that the DRRs do not contain incorrect information which would prevent the FS hardware from answering the request. A particular DRR containing such data would remain in the busy state indefinitely, eventually timing out if no audit check were made. SADK also verifies that no data which is in the DRRs or their associated memory will destroy valid data in either main memory or the file stores.

Note: Call store and program store are referred to in this section as ***main memory.*** This term is applicable to either core store or semiconductor store.

2. 1A PROCESSOR WRITABLE STORE AUDIT PROGRAM (SAWS)

GENERAL

A. Purpose of SAWS

2.01 The 1A Processor Writable Store Audit Program (SAWS) monitors and maintains the integrity of the resident ESS nontransient data base in both primary and secondary memory. Memory mutilation is detected by means of matching and hashing techniques. Failing ranges are corrected if a good copy can be identified, based on hash sum results.

2.02 In addition to the common audit and application interfaces (SAWSCMMN), a common interface (SAWSSUBR) is used to maintain hash sums and coordinate all update, audit, and tape copy activity affecting the stored program and various types of hashed office data.

B. SAWS Audit Strategies

2.03 Audit strategies include:

(a) Hashing of nontransient data

(b) Matching of store and backup copies.

Hashing provides a fast and powerful means of error detection over a given range without specific error resolution. Matching provides the ability to identify specific error addresses if each copy is not overwritten with the same bad data. However, matching is necessarily slow due to file store dependence.

2.04 Combined hashing and matching utilizes the advantages of both audit strategies. Frequent hashing of store data followed by a three-copy hash and match of "bad" ranges provides protection of store resident generic program and office data. Combined hashing and matching of all copies on a less frequent basis provides protection and error information for the backup data. In both cases, error information and automatic correction capabilities are maximized.

2.05 Combined hashing and matching is also used to audit nontransient data which is file store resident only.

SAWS—FUNCTIONAL DESCRIPTION

A. General

2.06 SAWS is made up of the three PIDENTs:

(a) SAWSCMMN/SAWSBASE

(b) SAWSSUBR.

The combination of SAWSCMMN and SAWSBASE perform the common audit functions of hashing, matching, and correcting nontransient data. SAWSBASE is associated with Program Store 0 (PS0). SAWSSUBR consists of the client service subroutines used in maintaining interfaces between SAWS and other programs which update, copy, or audit the hashed data. All three PIDENTs are used by the Processor Configuration Recovery Program (PCRV). In order to perform audits on writable stores which are associated with application programs, SAWS, must access application interface programs to acquire the preliminary information required to perform the audits.

B. SAWSCMMN—Functional Description

General

2.07 The program SAWS (Writable Stores Audit) consists of two distinct audit functions which are generally applicable to the nontransient ESS data base. These two functions are:

(1) Hashing

(2) Matching.

2.08 The hashing function makes use of previously initialized resident hash sums to detect memory mutilation within BINK (Binary one-thousand: See 2.11) ranges. The matching function detects mutilation of individual memory words by comparing duplicated copies of the nontransient data base.

2.09 The hashing function and the matching function are combined to audit various data types by IDtag. The data may be store-resident or file-store-only. The audit can employ the hashing mode or the hashing-matching mode. Failing BINKS (Binary one-thousand: See 2.11) are corrected automatically by overwriting the bad copy from a good copy provided the following conditions exist:

(a) Automatic correction is not inhibited.

(b) A good copy of a failing BINK can be identified.

Hashing and Matching Functions

Obtaining the Hash Sum

2.10 Hashing consists of end-around-carry addition of consecutive words in a specified memory range, coupled with rotation. Rotation is accomplished by rotating the accumulator left by five bit positions prior to each addition. Rotation starts over on 32-word boundaries. This facilities the updating of 32-word blocks and associated file store writes which must be done in multiples of 32 words. For purposes of System Reinitialization (SYSR) and System Update Programs (SYUP), hash sums having a value of logic zero are changed by the loader programs to minus zeros.

2.11 The standard hashing range for the 1A Processor is 1-BINK. A BINK is a binary (BIN) one thousand (K), or 1024 words which is the closest power of 2 to 1000. Each hash range begins and ends on BINK boundaries. This scheme provides speed and simplicity in detecting, resolving, and correcting errors. It also makes possible the auditing of any portion of memory in terms of its 1-BINK subblocks. This is especially important because of SAWS' interface with update programs which may involve temporary lockouts of audit or update activity.

2.12 Hashing of a specific structure is accomplished by hashing the 1-BINK range in which the structure is contained. The only structures which do not have 1-BINK hashing ranges are the merge structures. The merge structures are structures which are created during reinitialization and system update; they consist of:

(a) Main memory-to-file store map

(b) IDtag-to-file store range table

(c) Hash sum headcell and headtables

(d) Hash table.

Three special hash sums are computed over these critical merge structures to permit verification of their integrity before auditing other data.

2.13 A small number of 1-BINK ranges contain a combination of generic program data, various types of office data, and unhashed transient data. Whenever a combination of generic data and one or more types of office data occur in the same BINK, their partial hash sums, known as collisions, are maintained separately. No hash sum covers more than one data type. When running the System Audit of Stores Using Tape (SAST) (see 3.32), the use of separate hash sums eliminates confusion in determining which tape is to be mounted. The use of separate hash sums is also important in the interface between the audit and other programs that update, copy, or audit the same data. Transient data and data types other than the type being hashed (those having different IDtags) are skipped over during hash computation. The distinction between data types (such as generic and office data assembler (ODA)) and the skipping of transient data are accomplished via the range descriptors of the main memory-to-file store map and the IDtag-to-file store range table.

2.14 Since the stored hash sums are generated off-line by the generic loader and various office data assembler programs, it is possible to check the validity of tape data during a system reinitialization or system update by using hash sums. During a partial system update, the system data to be updated is checked before the update is performed. Hashes over the structures created during a system reinitialization are generated by the System Reinitialization Program (SYSR). Hashes over the structures created during a system update are generated by the System Update Program (SYUP).

Hash Sum Storage

2.15 Hash sum storage is in the highest numbered call store for the following reasons. Hash sum storage must take into account memory spectra for more than one electronic switching system. It must accommodate stores containing mixed data types and the scattering of data types over the memory address spectrum without requiring an interface between the generic loader and ODA programs. In addition, immediate accessibility of hash sums during system reinitialization is required.

2.16 The hash sums for both main memory resident data and file-store-only data are in a single hash table. This table consists of 64-word blocks. There is one block for each store containing hashed data and one block for each successive 64-BINK range of the file store spectrum containing hashed data that is file store resident only. The first 64-word block is reserved for extra hash sums that result when multiple data types occur within the same 1-BINK range. This first 64-word block is referred to as the ***collision block.***

2.17 The hash table's origin and length are specified in a two-word head cell. Two head tables provide 6-bit indices into the hash table. One head table is for main memory resident data and is indexed by a main memory address. The other head table is for file-store-only data and is indexed by file store address. The head cell and the head tables reside in Program Store 0 at Datapool-defined locations. For purposes of system reinitialization, hashes over the merge structures also reside in Datapool-defined locations in Program Store 0.

Control Of Audits In SAWSCMMN/SAWSBASE

2.18 Any application of the 1A Processor must provide certain application functions to be used by the Writable Store Hashing and Matching Audit (SAWS). These application functions include:

(a) Initialization of control data

(b) Handling of entries

(c) Segmenting and exits

(d) Administration of a buffer used for file store requests and automatic correction

(e) Outputting of error data.

Audit Control Block (ACB) Initialization

2.19 Before initial entry into SAWS, a number of control and data words must be initialized for use by the audit. These words are part of the ACB which can also accommodate words to be used by the audit scheduler. SAWS uses the ACB to save control and error information over a real-time break. Initial values of the audit control words determine the mode (hash-only or hashing-matching), range, and file store IDtags for the audit task associated with the particular ACB.

File Store Requests and Buffer Administration

2.20 When auditing in the matching mode, a SAWS task requires use of a store buffer for its file store reads. The buffer may have shared usage but must be provided and administered by the application system. Before any SAWS task begins matching a BINK or the merge structure, it must request the use of the buffer. An entry into SAWSCMMN/SAWSBASE for an audit task that is already in the matching mode implies that the buffer has already been obtained and its start address is stored in the ACB. When a SAWS task in the hash-only mode detects a hash failure, it must obtain use of the buffer before reauditing the failing range in the matching mode. This is handled by temporarily placing the task in the matching mode and exiting from SAWSCMMN/SAWSBASE to global PATTSZBF. Then, as with any SAWS task in the matching mode, a subsequent entry to SAWSCMMN/SAWSBASE implies that the buffer has already been seized.

2.21 Application administration of the buffer includes handling of file store request completion returns. Before submitting a file store request, SAWSCMMN/SAWSBASE transfers to PATTLFSC where it leaves a completion return address. This address is used by the File Store Administration Program (DKAD) when the file store operation is complete. SAWSCMMN/SAWSBASE reports each immediate success return from DKAD by transferring to PATTFSIR.

2.22 After submitting file store requests and segmenting, SAWSCMMN/SAWSBASE expects no further entries until all file store requests have been successfully completed. When auditing of a BINK or auditing of the merge structures in the matching mode is complete, the buffer is released by a transfer to PATTRLBF. At this point a SAWS task that has been temporarily placed in the hashing/matching mode is restored to the hashing mode. PATTRLBF returns to SAWSCMMN/SAWSBASE after the buffer is released.

2.23 More than one SAWS audit (or task) may be in progress at any given time on a time-shared basis. However, of those tasks in the matching mode, only one can have control of the buffer and other memory used by a matching task. Control of the buffer and other needed memory is seized alternately by various SAWS tasks for the auditing of a single BINK, or of the merge structures, and then released.

Segmenting and Exits

2.24 SAWSCMMN/SAWSBASE returns control to the audit controller for segment breaks and upon completion of an audit task. Two modes of segmentation and a single completion return are used.

2.25 One mode of segmentation is used to return control when normal interleaving with other audits is acceptable. A subsequent entry is not necessarily expected during the next base level cycle.

2.26 A second mode of segmentation is used as a non-interleaving return. This mode is used by SAWS tasks when a BINK or merge data is being matched. A subsequent segment entry is expected, preferably during the next base level cycle if file store requests are complete. When auditing of the

BINK or merge structures is complete, the buffer is released and the normal segmenting return is again used.

2.27 Upon final completion of an audit task, SAWSCMMN/SAWSBASE returns control to the audits controller by using the task completion return.

SAWS Multitask Interface

2.28 Since a number of SAWS tasks can be concurrently active (on a time-shared basis), measures have been taken to prevent possible interwrite problems where more than one SAWS task would concurrently audit and attempt to correct the same data. This is accomplished by subdividing an audit into subtasks based on file store IDtag and resident or file-store-only storage of data. A record is kept of active subtasks. Before a subtask is scheduled, the record is checked. If that subtask is already in progress, the SAWS task performing the check returns control to the audit scheduler and waits until the subtask is marked inactive. Once the subtask has been marked inactive, the SAWS task waiting to initiate that subtask obtains the subtask information it requires, including the file store IDtag and the file-store-only indicator, by transferring to PATTGSBT.

SAWS Interface With Update Programs

2.29 In addition to coordinating SAWS audit tasks, SAWS must interface with update programs to prevent auditing of data that is being updated, copied, or audited by other programs such as recent change System Update Programs, the Generic Utility Procedure, System Audit of Stores Using Tape (SAST), and Tape Writing Program (TWRP). SAWS handles this interface through SAWSSUBR which maintains an inhibit word and an activity word in call store. The inhibit word includes the client identifier and file store IDtags of data being changed, copied, or audited. The activity word includes a record of overall SAWS subtask activity.

2.30 Before initiating a SAWS subtask, SAWSCMMN/SAWSBASE checks the inhibit and activity words maintained by SAWSSUBR to determine if there is conflicting audit or update activity. If the subtask is in conflict with update activity which is in progress, control is either returned to the audit scheduler or a nonconflicting audit subtask is initiated.

Correction of Errors

2.31 SAWS' correction strategy is to correct failing ranges whenever hashing and matching results point unambiguously to a good copy. Correction is accomplished by overwriting the bad copy from the good copy. Whenever SAWS is entered, the hash sums and other merge structures critical to the audit are normally checked first. If an unresolved error is detected during this phase of the audit, SAWS outputs an error message and continues auditing in a noncorrective, hash-only mode.

2.32 SAWS routinely hashes (or hashes and matches in some instances) each 1-BINK range of nontransient data, comparing the resulting hash sum of each BINK with that BINK's hash sum stored in the hash sum table. If the hash sums match, SAWS proceeds to the next BINK. If the hash sums do not match, SAWS performs a combined hash and match of the BINK and the BINK's duplicated data located in file store 0 and file store 1 (if the BINK does not routinely receive a hash and match). The three hash sums and each data word in the three locations are compared in an attempt to find a match. (Hash sums of matching copies must be equal. However, hash sums can match if the data words are different. That is why each data word is compared.) If SAWS can determine that one of the data copies is correct, then the bad data is overwritten from the good copy. If a good copy cannot be identified, an error message is generated containing the relevant data.

2.33 Failure to find a correct copy of the data and the resulting error message which is generated should result in manual initiation of System Audit of Stores Using Tape (SAST) actions. These actions include:

(a) Mounting and matching of the appropriate system reinitialization tape(s) containing system data.

(b) Saving mismatching data

(c) Rollback of mismatched data at the completion of the match.

The System Audit of Stores Using Tape (SAST) is discussed further in another part of this section.

2.34 Hash sums of matching copies must be equal. However, copies do not necessarily match if their hash sums are equal. If such an event does occur, neither of the mismatching copies is regarded as a correct copy even though their hash sums appear correct. Such an inconsistency results in appropriate error messages.

SAWSCMMN Program Units

2.35 PIDENT SAWSCMMN consists of the following program units. The title, which depicts the function, and the global entry point(s) of each are provided. A detailed description of the program units may be obtained from the current program listing of PIDENT SAWSCMMN.

PU	TITLE/FUNCTION	GLOBAL ENTRY POINT(S)
1	Determines and initiates the next logical audit function in the program sequence. That function will be one of the following:	SAWSEXEC
1	(a) Start next subtask if none is in progress	
1	(b) Proceed with next step of automatic correction	
1	(c) Hash and check results over the current BINK for the specified IDtag	
1	(d) Match data in the buffer, checking the results if the BINK is complete	
1	(e) Submit a file store request(s) to move file store data into the buffer for matching during the next entry.	
2	Subtask Selector	(local)
3	Range Update	SAWSNEXT
4	Verify Merge Structures	(local)
5	Miscellaneous Internal Subroutine (Initialization and Updating)	(local)

PU	TITLE/FUNCTION	GLOBAL ENTRY POINT(S)
6	IDtag-To-File-Store-Only Range Selector	SAWSIDFR SAWSIDFS
7	File Store Read Into Buffer	(local)
8	Match and Hash	(local)
9	Mismatch Handler	(local)
10	Analyze BINK Failure	(local)
11	Automatic Correction	SAWSAUT0
12	Error Data Filter and Formatter	(local)

SAWSBASE Program Units

2.36 PIDENT SAWSBASE consists of the following program units. The title, which depicts the function, and the global entry point(s) of each are provided. A detailed description of the program units may be obtained from the current program listing of PIDENT SAWSBASE.

PU	TITLE/FUNCTION	GLOBAL ENTRY POINT(S)
1	Store-to-File-Store Range Selector	SAWSC2FS SAWSC2F0 SAWSC2FR
2	Hash Buffer Range	SAWSHSH1 SAWSHSH2
3	Hash BINK (pass/fail used by PCRV)	SAWSHTBK
4	Hash Sum Fetch For Specified BINK, File Store IDtag	SAWSGHPB SAWSGHSH
5	Mod 24 Rotate Amounts	SAWSMD24

C. SAWSSUBR—Functional Description

General

2.37 PIDENT SAWSSUBR consists of client service subroutines used in maintaining interfaces between SAWS and other programs that update, copy, or audit hashed data such as recent change, System Update Program, Generic Utility Program, and System Audit of Stores Using Tape (SAST). The two basic functions involved are:

(1) Data production. This feature prevents SAWS from auditing any data while that data is being updated or copied, and conversely, inhibits updates of any data while SAWS is auditing that data.

(2) Update. The following information is updated:

(a) Hash sums over updated data

(b) Superhash (hash sum over the hash table).

2.38 SAWSSUBR accomplishes its data protection function by maintaining in call store the client identifier and file store IDtags of data being changed, copied, or audited. This is an inhibit word. It also maintains in call store a record of overall SAWS subtask activity. This is an activity word. Before initiating any update activity affecting the hashes, an interfacing program must transfer to SAWSSUBR (PU 1) where the inhibit and activity words are checked for conflicting audit and update activities. If a conflict is apparent, the client is given a client-in-progress return implying that the requested update activity is temporarily inhibited. Otherwise, the appropriate client activity and file store IDtag indicators are set and a success return is given. Upon completion of any update activity, the update program transfers to SAWSSUBR (PU 1) which resets the subtask's activity indicator in the activity word located in call store.

2.39 Any of the interfacing programs can modify the hash table and the hash over the hash-table (superhash). For this reason only one update program may be active at any given time.

2.40 Associated with the update functions of SAWSSUBR are several subroutines used in the timing of file store activity. These are intended to guarantee the client program a completion return without unreasonable delay from a subroutine which depends upon file store operations. Update subroutines which are dependent on the File Store Administration Program (DKAD) obtain a timing list entry (PATTTIME) after successfully submitting their first file store request(s). Following a successful completion of the subroutine function, the timing list entry must be cancelled. This is accomplished via an entry into PATTRMOV. If DKAD fails to give a completion return to the update subroutine within a reasonable length of time, a timeout occurs. The subroutine then cancels any outstanding file store requests via SAWSCNCL (PU 15), and gives the client a completion failure return.

SAWSSUBR Program Units

2.41 PIDENT SAWSSUBR consist of the following program units. The title, which depicts the function, and the global entry point(s) for the program units are provided. A detailed description of the program units may be obtained from the current program listing of PIDENT SAWSSUBR.

PU	TITLE/FUNCTION	GLOBAL ENTRY POINT(S)
1	Set Interface Activity and Inhibit Bits	SAWSSAIB
2	Reset Interface Activity and Inhibit Bits	SAWSRAIB
3	Print and Pass Interface Error Data	SAWSNTFC SAWSNTFJ SAWSSERD SAWSSUER SAWSHPRJ SAWSHPRB
4	Check Client Activity and Inhibits	SAWSCCIA SAWSCCAI SAWSRCLK
5	Compute Updated Superhash	SAWSCUSH SAWSCSH1
6	Update File Store Range and Hashes From Data Buffer	SAWSUPFS SAWSUPFC

PU	TITLE/FUNCTION	GLOBAL ENTRY POINT(S)
7	Update Hash Over Hashtable	SAWSUHOH UHOHSUBR
8	Hash BINK, Single IDtag	SAWSHSBK
9	Build Partial Hash Block For Given K-Code, File Store ID	SAWSBPHB
10	Overwrite Procedure Hash Update	SAWSOPHU
11	Write Hash Table To File Store	SAWSWHFS
14	Submit Audit Request To Hash Specified Range	SAWSHKCD
15	Cancel Outstanding Subroutine File Store Requests	SAWSCNCL

3. SYSTEM AUDIT OF STORES USING TAPE PROGRAM (SAST)

GENERAL

3.01 The System Audit of Stores Using Tape Program (SAST) is a paged client of the Maintenance Control Program (MACP). SAST audits the 1A Processor store memory using the appropriate system backup tape as a reference.

3.02 The System Audit of Stores Using Tape—Resident Portion (SASR) is a main memory resident portion of SAST. SASR provides interface functions for SAST communications with the teletypewriter (TTY) and MACP. With the exception of SASR, SAST is entirely paged.

3.03 SAST is performed in two successive parts. These are:

(1) The non-correcting mode

(2) The correcting mode.

During the noncorrecting mode, system maps are audited and corrected images are built. The actual data is then audited and mismatches found between main memory or file store and the tape are buffered and printed to aid in identification of types or patterns of mutilation. The correcting mode writes the corrections into main memory and file store.

3.04 SAST may also be used to load nonmemory allocating file store-only data tapes. In this mode, maps are checked for consistency with system maps. If all tests pass, the data is written directly into file store. This feature is useful after a system reinitialization or a system update.

3.05 SAST requires a large buffer for file store reads, tape reads, and correction blocks. The scratch pad and unused portions of the paging area are used as the necessary buffers. Figure 2 is a functional diagram showing SAST's relationship with SAWS and other programs.

SYSTEM AUDIT OF STORES USING TAPE—MAIN MEMORY RESIDENT PORTION (SASR)

3.06 SASR is the only portion of SAST which is main memory resident. The remainder of SAST is entirely paged. SASR contains:

(a) Input message entry point

(b) Message processing routines

(c) MACP request code

(d) Termination routines.

Entry to SAST and exit from SAST are accomplished via SASR which interfaces with the TTY. Manual entries into the TTY are the only method of entry into SASR and, consequently, into SAST.

3.07 SASR accepts input messages from the TTY, analyzes the messages, sends an acknowledgment to the TTY, requests SAST paging, and sets up the proper data to initiate SAST operation. Output messages from SAST are processed by SASR and passed to the TTY. Upon completion of the SAST operation, SASR handles the termination routines.

SYSTEM AUDIT OF STORES USING TAPE PROGRAM (SAST)—PIDENT

A. Noncorrecting Mode

3.08 When running in the noncorrecting mode, errors which are detected in the data are buffered in a form known as correction blocks. These correction blocks contain:

(a) The address of the mismatch

(b) The corresponding data from tape, main memory, and both file stores.

The noncorrecting mode continues until either the entire tape has been processed or the correction block buffer area is filled. (If the correction block buffer area is filled, a check should be made to ensure that the proper tape is mounted since it would be ***very*** unusual for that many errors to exist.) In either case, a number of correction blocks will be printed so they may be visually examined for any patterns in data mutilation.

3.09 Upon completion of the noncorrecting mode, SAST will enter a "wait loop" for up to ten minutes, awaiting manually entered instructions. No corrections are made unless a message is entered to begin the correcting mode. If the message is not entered, the audit will time out and all data gathered by the audit is lost.

B. Correcting Mode

3.10 SAST enters the correcting mode only after the noncorrecting mode has completed and then only when the proper, manually entered message is received instructing it to begin the correcting mode. When the correcting mode is begun, mismatches found in main memory data are corrected using the tape data in each correction block as a reference. The main memory corrections are made in one time segment. Then all file store backup copies and file-store-only data are repaired with segment breaks occurring between file store read and write operations.

3.11 As described in 3.08, when SAST was in the noncorrecting mode, processing stopped for one of two reasons. Either the correction block buffer was filled (probably caused by the wrong tape

being mounted) or the tape completed. At that point, the audit either timed out or the correcting mode was requested. If the correcting mode was requested and the tape had finished running, then the corrections will be made and the audit will be terminated by SASR. However, if the correcting mode was requested and the tape had not finished running (the correction block buffer was full), then the noncorrecting mode will be reactivated and SAST will continue reading the tape, building a new set of correction blocks. Corrections will be made automatically this time without manual intervention. This cycle continues until the tape is completely processed and all corrections have been made.

3.12 After the entire tape has been processed and the corresponding corrections have been made, SAST moves the tape header to the correct location in file store, places the maps in file store, generates the "superhash" words used by SAWS, and stores the "superhash" words.

C. Data Loading Mode

3.13 After system reinitialization, system update, or some phases, areas of file store are zeroed which may be assigned to file-store-only hashed data structures. Since tapes containing this data are not included as part of a system reinitialization or system update, another method of loading the tapes is required.

3.14 SAST has the capability of loading tapes which have memory allocation provided by either Datapool (on generic tape) or office-dependent data assemblers. When it receives a request to load such a tape, SAST verifies that the data contained in the maps does not conflict with data contained in system maps. If any conflicts are detected, the tape is not loaded. If the tape is compatible with system data, SAST moves data from the tape through a main memory buffer to file store. It then places the hash sums in the correct locations in main memory and file store, creating additional head table indices as necessary.

SAST PROGRAM OPERATION

A. Noncorrecting Mode

General

3.15 The noncorrecting mode of SAST is divided into the following functions which are performed by the indicated program units:

PU	FUNCTIONS	PROGRAM UNITS
1	Initialization	Initialization and Startup
2	Checking Main Memory To File Store Backup Map	Audit Main Memory To File Store Map
3	Checking The ID Tag To File Store Address Map	Audit IDtag To File Store Map
4	Checking Hash Sum Head Table	Audit Hash Sum Headtable
5	Checking Hash Sum Collision Block	Audit Hash Sum Collision Block
6	Check Hash Sums	Audit Hash Sum Tables
6	Check Data	Audit Data With Input IDtag

Initialization

3.16 Since SAST uses the unoccupied portion of the paging area and the MACP scratch pad as buffers, SAST must determine and save the addresses of available areas. The tape header is read and matched to information contained in the input message. If an error occurs, a message reporting a tape header mismatch will be printed.

Checking Main Memory-to-File Store Backup Map

3.17 Each tape which SAST uses to audit has a partial main memory-to-file store map. The data in the tape main memory-to-file store map is assumed to be correct. Any entry in the system main memory-to-file store map which conflicts with the tape version is assumed to be mutilated. In order to preserve all data on the tape and eliminate any mutilated or overlapping descriptors, a revised main memory-to-file store map is constructed.

3.18 The main memory-to-file store map consists of a head table and a number of three-word file store descriptor blocks (DDB). The following validity checks are made on the head table:

(1) Head table entries which indicate the presence of DDBs are checked to verify that the pointer to the DDBs falls within the range of the main memory-to-file store map.

(2) Head table entries which indicate 32 BINK of homogeneous backed up data are checked to verify that the file store address is valid.

When these internal checks are made, the system head table is compared to the tape head table and discrepancies between the two are counted.

3.19 Next the DDBs are checked. The following tests are made:

(1) The main memory start addresses and main memory end addresses in the DDBs are checked to assure that they fall into the same 32-BINK range.

(2) File store addresses in the DDBs are checked to assure that they are valid.

All DDBs which pass these tests are transferred to the revised main memory-to-file store buffer. Any tests which fail cause the error counter to be incremented.

3.20 After ordering the DDBs in the buffer, the DDBs from tape are merged with those in the buffer. Any system DDB which only partially matches a tape DDB or which overlaps a tape DDB is deleted and the error counter is incremented.

3.21 After the merge procedure, the DDBs which remain are internally consistent (no overlaps) and are in order by ascending main memory address. The final step is to verify the linkage from the head table to the corresponding DDBs. If there are DDBs present for a range which the tape claims is either not backed up or is a 32-BINK homogeneous data area, then all the DDBs are deleted. After all pointers are adjusted and consistency checks are made between the DDBs and the head table, the unused portion of the DDB area is zeroed.

3.22 The main memory-to-file store map which is built by SAST is consistent but not necessarily complete. All information pertaining to the given tape is correct and complete and all information pertaining to ID tags not present on the tape is consistent with the tape data. A DDB of a different ID tag may have been mutilated and deleted or mutilated in such a way as to have an incorrect (but not overlapping) range. Such errors will remain until the tape of that data type is audited, at which time the errors will be detected.

Checking The IDtag-To-File Store Address Map

3.23 Each system tape has a partial IDtag-to-file store map which contains ranges of file store addresses corresponding to a given IDtag. The IDtag-to-file store map consists of:

(a) A head table which contains one entry for each IDtag

(b) A number of four-word range descriptor blocks.

A revised IDtag-to-file store map is built by first placing all tape information into a buffer. Pointers are then adjusted to be consistent with the compool location of the IDtag-to-file-store map.

3.24 Next, the system head table and range blocks are checked for consistency. The following checks are made on the head table:

(a) The pointer for IDtag zero must be zero since no range blocks are associated with this ID.

(b) Pointers to range blocks are range checked to verify that they fall within the limits of the range block area of the IDtag-to-file store map.

3.25 The range blocks are verified using the following checks:

(a) The file store start address of the range must be less than the maximum file store address minus one sector.

(b) The file store end address of the range must be less than the maximum file store address.

(c) The start address must be less than the end address.

(d) The range covered by any single range block must not be greater than the amount of storage on a single file store.

(e) The back linkage pointer must either point to the head table or to the previous range block of the same ID.

(f) The range specified is checked against each of the range blocks in the buffered IDtag-to-file-store map. If no overlap is found, the test range block is moved to the rebuilt map.

After all range blocks have been checked, the unused portion of the range block area is zeroed.

3.26 As mentioned concerning the main memory-to-file store map, the IDtag-to-file store map built by SAST is consistent but not necessarily complete. All information pertaining to the given tape is correct and complete and all information pertaining to IDtags not present on the tape is consistent with tape data. All range descriptors of a nontape IDtag which follow a mutilated linkage pointer will be lost. Ranges pertaining to IDs not on the tape may be mutilated in such a way as to have an incorrect (but not conflicting) range. To detect and correct such errors, the tape corresponding to the remaining ID tags should be audited.

Checking The Hash Sum Head Table

3.27 Several data structures are used to define the location of system hash sums. The hash sum head cell is a block of two words containing the start address and length of the table containing the

actual hash sums. The hash sum head table (main memory data) and the hash sum head table (file-store-only data) are two adjacent tables containing indices defining the location in the hash sum table of those hash sums pertaining to a particular 64-BINK range.

3.28 Head cell data appears only on the generic tape. If this tape is being audited, the tape values are compared to system values and errors are noted. If any other type of tape is being audited, system values are used.

3.29 The length is checked to assure that it is not larger than the maximum value. Then the rebuilt main memory-to-file store map is checked to determine the file store address of the hash sums. The main memory address of the hash sums is used to determine the entry into the main memory-to-file store map. If no DDB can be found which describes the address range claimed by the head cell, SAST will be unable to determine the file store address of the hash sums and will abort.

3.30 The following validity checks are made on the head table:

(a) All duplicate indices are removed.

(b) The main memory-to-file store map is checked to verify that backed up data exists for each index in the head table for main memory data.

(c) All indices are deleted which correspond to an area outside the length of the hash sum table defined in the head cell.

(d) Indices are deleted which correspond to invalid file store addresses.

(e) All indices of hash sums which appear on the tape are verified.

3.31 SAST accomplishes the head table audit by building the head table in its buffer space. After the audit, all indices are consistent and those pertaining to the hash sums on the tape are correct. It is possible that an index pass all checks and still not indicate the correct hash sum location.

Checking The Hash Sum Collision Block

3.32 A collision block is a method of storing hash sums for areas of main memory where two or more types of hashed data fall within the same BINK boundary.

3.33 Since storage of hash sums in the hash table is limited by the table structure to one hash sum per BINK, special provisions must be made for storage of hash sums over BINKS in which two or more types of hashed data are present. When this case arises, the BINK address of the "collision," the IDtag, and the hash sum over data corresponding to that IDtag are placed in a collision block in the hash index zero area of the hash sum table.

3.34 To audit the collision blocks, the main memory-to-file store map is interrogated using the index corresponding to the collision address. This verifies that DDBs exist for the range in question. All IDtags claimed in the collision blocks are verified by searching the corresponding DDBs. Only those which are verified are moved to a buffer area.

3.35 Next, the collision blocks are read from tape. If collisions exist on the tape (a rare case), they are compared to those in the rebuilt buffer. Any necessary corrections are made in the buffer and the error count is incremented.

3.36 During the hash sum audit, if the system entry indicates collision blocks, SAST will interrogate its rebuilt version to assure that the hash sum is correct.

3.37 After the collision area has been reconstructured, the table is placed in descending order according to BINK address/IDtag.

Checking Hash Sums

3.38 The hash sum head table provides the means for locating the address of hash sums corresponding to a given 64-BINK range. Since SAST cannot rely on the system copy of the head table, another means must be used to locate the hash sums.

3.39 When a block of hash sums is read in from tape, SAST compares each nonzero entry to the corresponding entry of the system hash sums for index 1, counting the number of matches found. Then hash sums for index 2 are compared to the tape and the number of matches is counted. The process is continued until the tape block has been compared to each block in the system. Whichever index resulted in the highest number of (nonzero) matches is declared to be the correct index and the corresponding entry in the rebuilt head table is made. Correction blocks are built for all mismatches found.

3.40 The entire process is repeated for each block of hash sums in the tape. When all tape hash sums have been processed, the rebuilt table reflects the position of each hash sum block for all data on the correct tape.

Checking Data

3.41 After the merge data file has been processed, the remainder of the audit proceeds in a straightforward manner. From this point on, there is a 1-to-1 correspondence between what is on tape and what should be in the system. After a block of data is read from tape, the file store data corresponding to that data is read from each file store. The tape data is then compared in sequence to the main memory data (if data resides in main memory), file store 0 data, and file store 1 data. If any of the three mismatch, a correction block is built.

3.42 The data audit proceeds until all data on the tape has been checked or until the correction block buffer is full. In either case, up to 600 correction blocks are printed to aid the craftsman in visually detecting patterns to the mutilated addresses or data. SAST will then wait a maximum of 10 minutes for further instructions.

B. Correcting Mode

3.43 The correcting mode of SAST is the part which actually makes changes in the system. During the noncorrective mode, mismatches are only detected. During the correcting mode, the mismatches which were detected are corrected. Normally, any mismatches which are the result of recent changes are not considered as errors. However, a mode of operation is provided in which mismatches caused by recent changes are treated as errors.

3.44 When the correcting mode is requested, SAST writes all main memory corrections in one segment, using the correction blocks as a source. After a break, file store backup addresses are calculated for each main memory address and the corrections are made to file store copies. Since file store correction is a much longer process, it must be interleaved with other system processing via segment breaks. This unavoidably leaves a period of time when main memory and file store are out of synchronization.

3.45 The noncorrecting mode will normally stop processing the tape for one of two reasons:

(1) The tape is completely processed or

(2) The correction block buffer has filled up.

If the correcting mode is requested and the tape finished, corrections will be made and the audit will terminate. If the correcting mode is requested when the correction buffer is full, all corrections contained in the correction area will be made as described. Then the audit will continue processing the tape. This cycle continues until the tape is completely processed.

C. Program Interfaces

Common Program Interfaces

3.46 SAST has software interfaces with the following common programs for the reasons indicated:

DKAD—File Store Administration Program. Standard DKAD routines are used to read and write file store. In addition, DKADCTDA is used to convert a main memory address to file store address.

DUAD—Data Unit Administration Program. SAST uses standard DUAD routines to secure a tape unit (DUADSECU), open and close nonstandard label file (DUADOPNL, DUADCLNL), read a data block (DUADREAD), advance to end of file (DUADFEOF), and to terminate the use of the Tape Unit Control (DUADFOFF).

IOCP—Input/Output Control Program. IOCP provides all routines necessary to interpret and acknowledge input messages (IOCPIMCK, IOCPIACK) and to print output messages (IOCPPRNT).

MACP—Maintenance Control Program. Since SAST is a MACP client, it uses many of the client interface MACP routines. MACPRJB1 is used to request a MACP entry, segment breaks are taken via MACPSRTT, and job termination is accomplished by transferring to MACPNEOJ. MACPABAD is used to establish a default abort address; MACPDKOP provides the interface required to handle delayed disk completion returns. MACPPTTY provides a delayed return point during printing of partitioned TTY messages. MACPPLT2 is used to provide a 1-second timing mechanism.

PAGS—Paging Program PAGSGFRA is used to calculate the address and size of the unused portion of the paging area after SAST has been paged in.

PATT—Processor Application Transfer Table. All transfers to application programs are done via PATT.

SAWS—Writable Store Audit. SAWS routines are used to establish and release the memory changing client lockout mechanism (SASSAIB, SASRAIB). In addition, "superhashes" are calculated by SAWSHASH.

Application Program Interfaces

Identifying Mismatches Caused by Recent Changes

3.47 Since SAST can be used to audit the office data assembler (ODA) tape, the probability is very high that SAST will detect data which has been altered by recent change procedures. This situation will arise if any recent changes have been introduced after the most current ODA tape has been written. To avoid an undesired rollback, all detected mismatches which are due to recent change activity should be deleted from SAST's correction blocks.

3.48 Whenever an ODA tape is being used by SAST, a transfer will be made to an application recent change routine via PATTRCTA. The application routine will scan the SAST error correction blocks and zero the mismatch address of any correction block which pertains to a recent change. When the application routine finishes, SAST will delete all correction blocks with a zeroed address and will repack the correction buffer.

Requesting SAWS Audit

3.49 After SAST has completed its auditing procedure, a request is made via PATTMALL to run SAWS. This verifies that the new system maps and superhash sums are correct and that any "uncorrectable" conditions reported by SAWS have been corrected by SAST.

4. AUXILIARY UNIT SYSTEM AUDIT PROGRAM (SADK)

GENERAL

4.01 SADK performs the following four functions:

(1) Initializes the file store system

(2) Audits memory, specifying:

(a) File store equipage

(b) Memory used by the File Store Administration Program (DKAD)

(c) Outstanding file store requests

(d) High and low priority queues.

(3) Times the file store requests

(4) Audits the Automatic Message Accounting (AMA) buffer pointers.

4.02 SADK is executed under the control of an application audit control program. The application audit control program first enters an application interface program which supplies preliminary information required by SADK. Control is then passed from the interface program to SADK for execution of the auxiliary unit system audits. If any errors are detected during the audits, the error information is returned via the PATT table to the application audit control program. When SADK has completed, control is returned to the audit control program via the PATT table. Figure 2 is a functional block diagram which depicts SADK's relationship with other programs.

FUNCTIONAL DESCRIPTION OF SADK ROUTINES

A. Initializing the File Store System

4.03 This routine initializes the memory which is used by DKAD and idles all file store requests. In order to idle the file store requests, this routine:

(a) Zeroes the File Store Request Blocks (DRBs)

(b) Zeroes the Client Answer Blocks (CABs)

(c) Zeroes the queues

(d) Writes all File Store Request Registers (DRRs) in the file store to the idle state.

This routine is normally called during a phase of memory initialization.

B. Memory Audit

4.04 The Memory Audit routine performs the following functions:

(1) Audits office data specifying file store equipage against memory initialized by the File Store Fault Recovery Program (FSFR)

(2) Audits memory used by DKAD such as:

(a) The Call Store Last-Look (CSLL) words

(b) File Store Request Block (DRB) address

(c) Client Answer Block (CAB) address

(d) The number of DRRs used for each file store.

(3) Audits outstanding disk requests by comparing the DRB data with the DRR data

(4) Audits the high and low priority queues.

C. Timing of File Store Requests

4.05 Each request for file store data transfer which is entered into the system is given a specified number of seconds in which it must complete. A client does not have to time each outstanding file store request. SADK/DKAD will inform the client of all requests which have timed out. However, if a client must complete a function in a given amount of time, an overall sanity timer is used. The file store requests are timed in the following manner:

DRR Requests

4.06 For each active request, a TIME flag is set. This flag is the first word of the Client Answer Block (CAB). On the next entry the time flag is checked. If the flag is set, then the request is considered to have timed out. The File Store Fault Recovery Program (FSFR) is then entered to resolve the timeout.

4.07 If FSFR reconfigures the file stores, all requests which have the TIME flag set in the CAB are given an additional second in which to complete. If the FSFR determines that the timeout occurred as the result of a software error, the request which initiated the FSFR will be cancelled. The client will then be given a completion failure message with a return code specifying timeout.

Queued Requests

4.08 The mechanism for timing queued requests is similar to the DRR timing in that a TIME flag is set in a queue word. If a queue request times out, the request is unloaded from the queue and the client is given a completion failure message with a return code specifying timeout. There is no interface with FSFR for requests on queue which time out.

D. Automatic Message Accounting (AMA) Buffer Audit

4.09 This routine audits the consistency of the AMA buffer pointers. These pointers are used by the AMA data transfer. If any inconsistent pointers are found, they are corrected.

PROGRAM UNIT DESCRIPTIONS

A. General

4.10 SADK contains four program units which accomplish the functions described above. The following is a brief description of the four PUs, their entry points, and the functions performed at each entry point. A detailed description of the program units may be obtained from the current SADK program listing.

B. File Store Audit PU

Purpose:

Audit file store software

ENTRY POINTS	FUNCTIONS
SADKENT (global)	(a) Audits DRB
	(b) Audits CAB base addresses
	(c) Verifies the word which indicates the number of DRRs
FSADDRR (global)*	(a) Audits DRB data and a corresponding DRR
	(b) Audits the state of the CSLL.

*If the File Store Fault Recovery Program (FSFR) is copying a file, this portion of the audit is inhibited.

STEP	ACTION	VERIFICATION
	FSADQUE (global)	Audits high and low priority queues and the various queue head cells.

Execute tables and error routines in this PU are entered as required during the audit.

C. File Store Initialization PU

Purpose:

(a) Initialize memory used by DKAD

(b) Idle all requests in the file store busy/idle registers

(c) Zero the DRB and CABs

(d) Initialize high and low priority queues to zero

(e) Initialize the queue head cells.

ENTRY POINTS	FUNCTIONS
DKADINIT (global)	(a) Initializes memory for DKAD Idles DRRs located in each of the in-service FSCs.
ICØMMUNITY0 (local)	(b) Idles all DRRs located in each file store for Community A
ICØMMUNITY1 (local)	(c) Idles all DRRs located in each file store for Community B
INITCNST (local)	(d) Reads file store equippage for a file store community.

D. File Store Request Timing PU

Purpose:

Entered on base level for the purpose of timing outstanding file store requests which are in the DRRs or on queue.

ENTRY POINTS	FUNCTIONS
SADKTIME (global)	When a request in a DRR times out: (a) Prints a report message indicating a timeout (b) Cancels the request indicating a timeout (c) Transfers to FSFR to analyze the timeout. When a high or low priority request which is on queue times out: (a) Prints a report message indicating a timeout (b) Cancels the request indicating a timeout (c) Dispenses the queued request and the client's fail address.
SOFT_ERROR (local)	When a software error occurs: (a) Cancels requests in the input register (b) Resumes search of DRR requests.

ENTRY POINT	FUNCTION
FS_RESTART (local)	When file stores have been reconfigured or reinitialized: (a) Allows all requests one additional second to complete.
TØ_MSG (local)	Prints a message indicating that a file store request has timed out.
PRØCQUE (local)	(a) Audits high and low priority requests which are on queue to determine if any requests have timed out (b) If a request is found which has timed out, the request is removed from the queue and the client is given a completion failure return.

E. Automatic Message Accounting Buffer Audit PU

Purpose:

Audit the consistency of the Automatic Message Accounting (AMA) buffer pointers used by the Automatic Message Accounting Data Transfer Program (AMDX). Inconsistent pointers are corrected.

ENTRY POINTS	FUNCTIONS
AMDXAUD (global)	Audits the AMA buffer pointers.

5. REFERENCES

5.01 More detailed information about the programs described in this section may be found by referring to the appropriate program listing (PR) of the following programs:

SAWS - 1A Processor Writable Store Audit

SAWSCMMN PR-5A239

SAWSSUBR PR-5A240

SAWSBASE PR-5A238

SAST - System Audit of Stores Using Tape

SAST PR-5A237

SASR PR-5A236

SADK - Auxiliary Unit System Audit PR-5A235.

5.02 The introductory BSP to the application programs provides a complete list of 1A Processor and application programs and the sections in which they are described. More detailed information about all programs referenced in this section may be found by referring to that BSP.

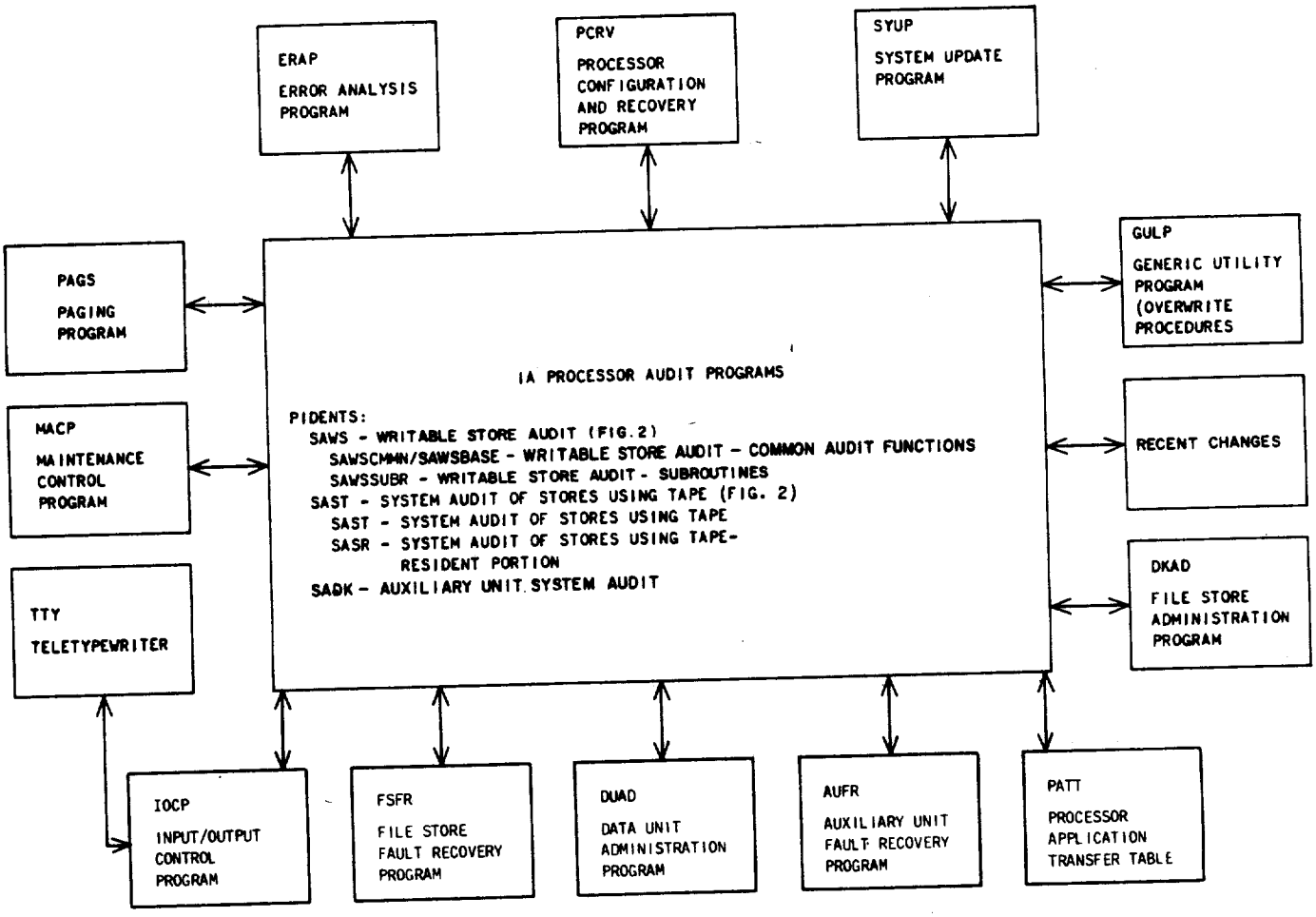

Fig. 1—1A Processor Audit Programs—Program Interface Block Diagram

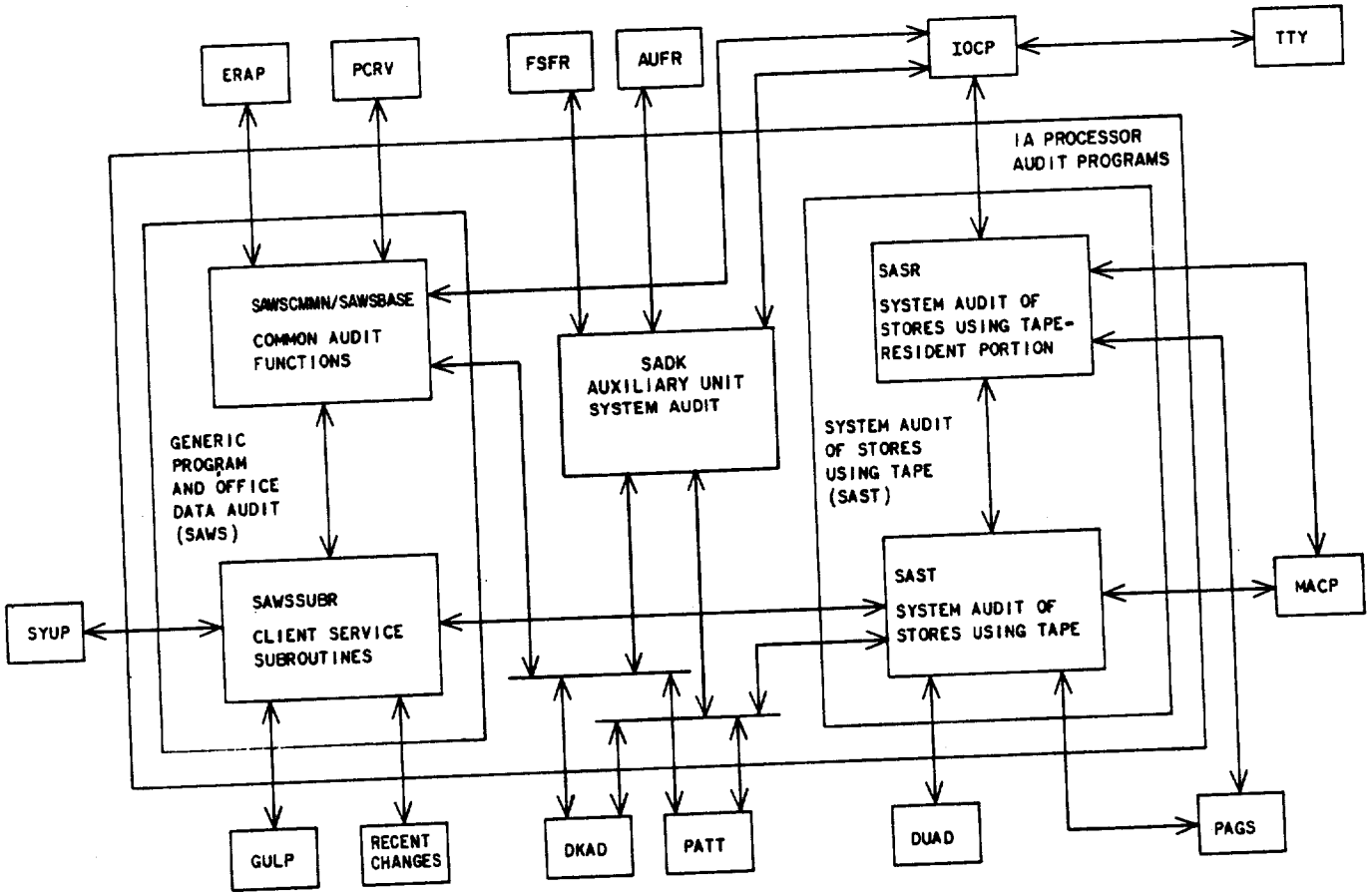

Fig. 2—1A Processor Audit Programs—Functional Block Diagram

TABLE A

ABBREVIATIONS AND ACRONYMS

ABBREVIATION	TERM
ACB	Audit Control Block
AMA	Automatic Message Accounting
AMDX	Automatic Message Accounting Data Transfer
BINK	BINary One-thousand (K). This is a quantity of 1024, the closest binary number to 1000.
CAB	Client Answer Block
CSLL	Call Store Last-Look
DDB	File Store Descriptor Block
DKAD	File Store Administration Program
DRB	File Store Request Block
DRR	File Store Request Register
DUAD	Data Unit Administration Program
ESS	Electronic Switching System
FSC	File Store Community
FSFR	File Store Fault Recovery Program
IDtag	Identification Tag
IOCP	Input/Output Control Program
MACP	Maintenance Control Program
ODA	Office Data Assembler
PAGS	Paging Program
PATT	Processor Application Transfer Table
PR	Program Listing
SADK	Auxiliary Unit System Audit
SASR	System Audit of Stores Using Tape — Main Memory Resident Portion
SAST	System Audit of Stores Using Tape
SAWS	1A Processor Writable Store Audit
SYSR	System Reinitialization Program
SYUP	System Update Program.
TTY	Teletypewriter