FAULT RECOVERY PROGRAMS
SOFTWARE DESCRIPTION
1A PROCESSOR

CONTENTS PAGE

CONTENTS PAGE

Printed in U.S.A.

CONTENTS PAGE

CONTENTS PAGE

CONTENTS PAGE

CONTENTS PAGE

CONTENTS PAGE

CONTENTS PAGE

Figures

CONTENTS PAGE

1. GENERAL

1.01 This section provides a description of the programs used to recover from faults in the 1A Processor. Generally, the 1A Processor Fault Recovery Programs are those that recover faults within the 1A Processor System. However, since the 1A Processor is a common system and is used to provide stored program control for both local and toll switching systems, the 1A Processor Fault Recovery Programs must also direct program control to the appropriate applications program if the fault indicators point to the applications hardware. Since the 1A Processor common systems fault recovery programs and the application fault recovery programs work together and interact to provide a fault recovery capability for the total system, this section provides a description applicable to the complete system.

Note: Since the fault recovery programs interact closely with the hardware, an understanding of the hardware is important to understanding the fault recovery programs. For information on the 1A Processor System, refer to Section 254-200-001.

1.02 This section is reissued to incorporate CPR7 program changes. This includes:

(a) A description of Single Strategy Fault Recovery (SSFR) which is comprised of both Auxiliary Unit Fault Recovery (AUFR) and Attached Processor Fault Recovery (APFR) programs

(b) Numerous changes to the 1A Processor F-Level Fault Recovery Program (PFLR).

Revision arrows are used to emphasize the more significant changes.

1.03 Part 12 provides a defined list of abbreviations and acronyms used in this section.

1.04 This section describes the following programs and their associated program identifications (pidents). The program listings (PRs) may be referred to for further information.

(a) Central Control Fault Recovery Program—CCFR

(1) CCFRMAIN (PR-5A304)

(2) CCFRTEST (PR-5A305)

(b) Call Store Fault Recovery Program—CSFR

(1) CSFRNORM (PR-5A307)

(2) CSFRBASE (PR-5A306)

(c) Program Store Fault Recovery Program—PSFR

(1) PSFRCSPG (PR-5A325)

(2) PSFRPSPG (PR-5A326)

(d) Auxiliary Unit Fault Recovery Program—AUFR

(1) AUFRCNTL (PR-5A300)

(2) AUFRCPGM (PR-5A301)

(3) AUFRTEST (PR-5A302)

(4) AUFRDFOR (PR-5A303)

(5) ♦AUFRILEV (PR-5A365)♦

(e) File Store Fault Recovery Program—FSFR

(1) FSFRDGN (PR-5A318)

(2) FSFRDISK (PR-5A319)

(3) FSFRSTAT (PR-5A320)

(f) ♦Single Strategy Fault Recovery for the Attached Processor System

(1) AUFRILEV (PR-5A356)

(2) APFRILEV (PR-5A350)

(3) APMHCNTL (PR-5A520)

(4) APFRICON (PR-5A352)

(5) APFRBASE (PR-5A351)♦

(g) Data Unit Fault Recovery Program—DUFR

(1) DUFRDFOR (PR-5A308)

(2) DUFRDGNI (PR-5A309)

(3) DUFROFLN (PR-5A310)

(4) DUFRPCAU (PR-5A311)

(5) DUFRPCDU (PR-5A312)

(6) DUFRPCSB (PR-5A313)

(7) DUFRSUBR (PR-5A314)

(8) DUFRTADM (PR-5A315)

(9) DUFRTSTS (PR-5A316)

(10) DUFRTTYI (PR-5A317)

(h) 1A Processor F-Level Fault Recovery Program—PFLR

(1) PFLRPIIR (PR-5A339)

(2) PFLRBLMH (PR-5A324)

(3) PFLRDGNH (PR-5A323)

(4) PFLRRRCR (PR-5A342)

(5) PFLRPUMP (PR-5A341)

(6) IOPUMPPC (PR-5A343)

(7) IOPUMPBX (PR-5A350)

(i) 1A Processor Power Conversion and Distribution Frame Fault Recovery Program—PDFR

(1) PDFR (PR-5A327)

2. GENERAL APPROACH TO 1A PROCESSOR FAULT RECOVERY

2.01 The 1A Processor Fault Recovery Programs (Fig. 1) are normally entered as a result of a maintenance interrupt. However, they may also be entered on interject or via the input/output (IO) handler program, via routine exercise programs, via the maintenance control program for deferred (base level) fault recovery testing, or via manual requests from the TTY. But, the primary purpose of these programs is to restore the system to call processing in the face of system errors or faults.

2.02 Interrupts of levels A through K may occur at any time during system operation, but maintenance and operational interjects will occur only at times set by the software code. (Within the 1A Processor, file stores, data unit selectors [DUS], and tape unit controllers [TUCs] use the interject facility.) The hardware interrupt sequencers in central control may be triggered either manually (A level) or automatically by hardware.

2.03 As a part of the hardware sequencer action, a wired transfer is made to the appropriate entry point in the System Interrupt Recovery Program (SIRE). The SIRE program then stores a basic set of data that may be useful for determining the

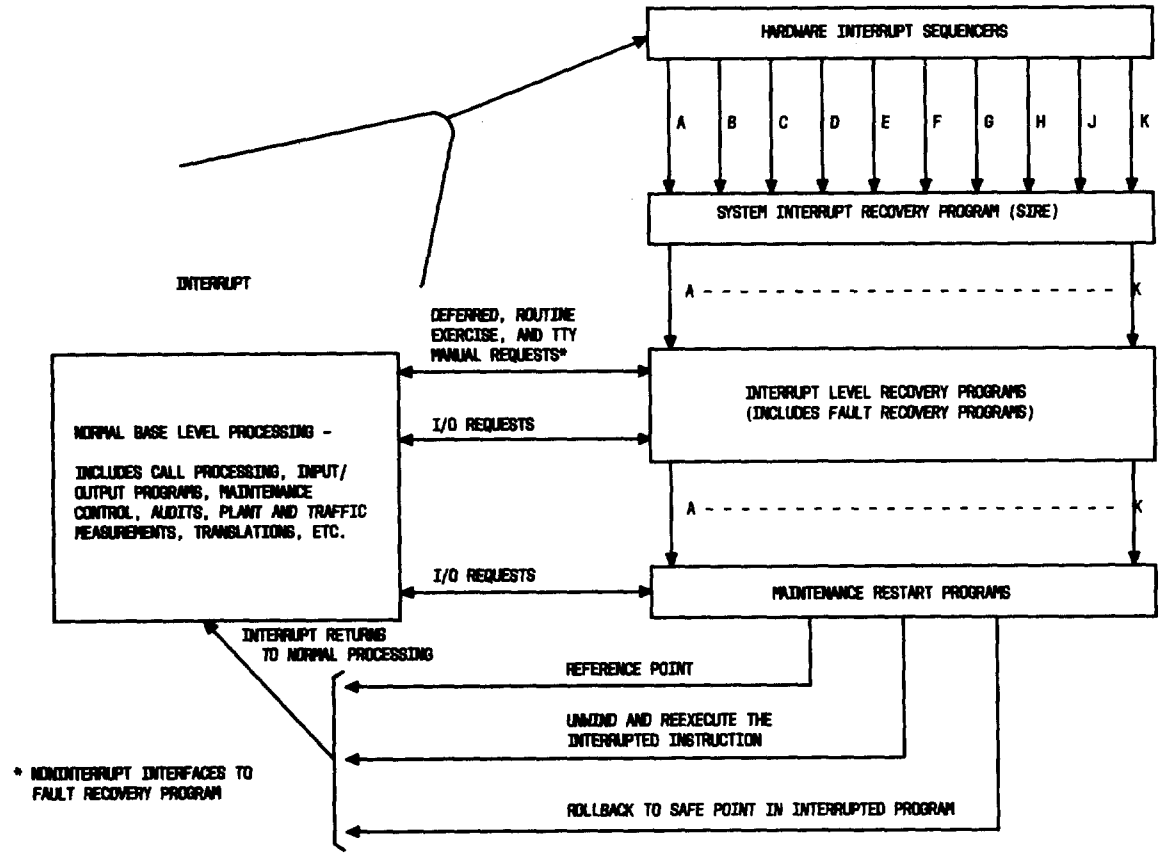


Fig. 1—General Program Flow for Fault Recovery

cause of the interrupt and also may be useful for restarting base level processing after an interrupt recovery. This data is stored at memory locations (interrupt bins) that are assigned to each interrupt level.

2.04 After SIRE has stored the required data in the appropriate interrupt bin, a program transfer is made to an interrupt-associated filter program. This program determines the primary source of the interrupt and also determines which unit or units are implicated. After the basic source of the interrupt is resolved, per unit or per source fault recovery programs are entered.

2.05 The fault recovery programs are designed to isolate faulty units or subsystems rather than to identify replaceable components. These programs recognize and isolate most call-affecting faults during a single interrupt interval.

2.06 The basic techniques of fault recovery strategy are centered around rapid resolution of problems and quick return to normal system operation. The fault recovery programs report error data to the error analysis programs. Error analysis maintains a history of interrupts and associated data.

2.07 After the fault recovery program has selected a working configuration of hardware, the program must do several "housekeeping" tasks. The program must set appropriate flags that will cause diagnostics and other testing to be scheduled on base level after the system has returned to call processing. Also, the program must record the actions it has taken in the appropriate error analysis data history.

2.08 Finally, the fault recovery program begins output messages to convey its actions to operating personnel. If several interrupts have failed to resolve a persistent problem, the output messages may be used to supplement the automatic error anal-

ysis. The operating personnel may analyze the output messages and select a working configuration of hardware manually.

2.09 After the fault recovery program has taken the basic recovery action, the program transfers to the Maintenance Restart Program (MARP). The MARP program then does several additional functions based on the highest level interrupt encountered during an interrupt interval. (Multiple interrupts may occur during an interrupt interval. Here, the lower level interrupts are recorded, but not analyzed.) Program MARP (using SIRE-stored data) restores or initializes system registers, including central control prevent error source transmissions (PESTS), as appropriate. It also sets flags for the appropriate audits to be scheduled on base level after the return to normal processing.

2.10 The MARP program also provides an interface to application restart programs to accommodate restart procedures that are application dependent. After program control is returned to MARP from the applications restart program, MARP starts the report messages and does the return to normal processing.

2.11 The return to normal processing is dependent on flags set by the fault recovery programs and the applications restart programs. However, MARP may alter the decision. The basic options for return to call processing are as follows:

(a) Unwind and reexecute the interrupted instruction. This option is used for interrupts lower than C level. (This option is the most common return.) The MARP program may alter this decision based on either the instruction type or on errors detected in automatic interrupt data storage (pre-SIRE).

(b) Roll back to a safe point in the interrupted program. This option is used if more than one interrupt has occurred during the interrupt interval, if a nonunwindable instruction such as an execute was the interrupted instruction, or if the interrupt was an out-of-range failure.

(c) Return to a reference point, either the beginning of the base level cycle or to the beginning of the interject task schedule. This option is used after a system phase, A-, B-, C-, and K-level interrupts, or after any interrupt lasting longer than 10 ms.

2.12 Many program flows other than for interrupt recovery exist for the fault recovery programs. A more complete description of fault recovery follows.

3. CENTRAL CONTROL FAULT RECOVERY PROGRAM—CCFR

INTRODUCTION

3.01 The primary purpose of the CCFR program is to verify the integrity of the active central control: (a) following a system reconfiguration, (b) following a system malfunction, (c) on a demand basis from other maintenance programs, or (d) on a demand basis from a TTY request.

3.02 In addition to doing the verification of the active central control, CCFR does a validity check of the standby central control on most requests (provided the standby central control can be put in step with the active). The CCFR program will not attempt to verify the standby central control on a B-level interrupt because of a processor configuration or a pulse source failure. Also, the standby is not tested on TTY requests for deferred testing of the active central control.

3.03 To perform its functions, CCFR interacts with many other system maintenance programs. The major program interfaces are shown in Fig. 2.

3.04 The CCFR program provides the necessary service routines to control the configuration of the central controls. These routines are used by CCFR and all other programs that change the central control configuration during normal system operation. The CCFR program service routines

(a) Remove the standby central control

(b) Restore the standby central control to service with routine matching

(c) Switch active and standby central controls

(d) Control the match mode and type of matching (if any) that is to be established during normal system operation.

3.05 Two additional routines are provided to set up routine matching without interrupts and to put the central controls in step without matching.

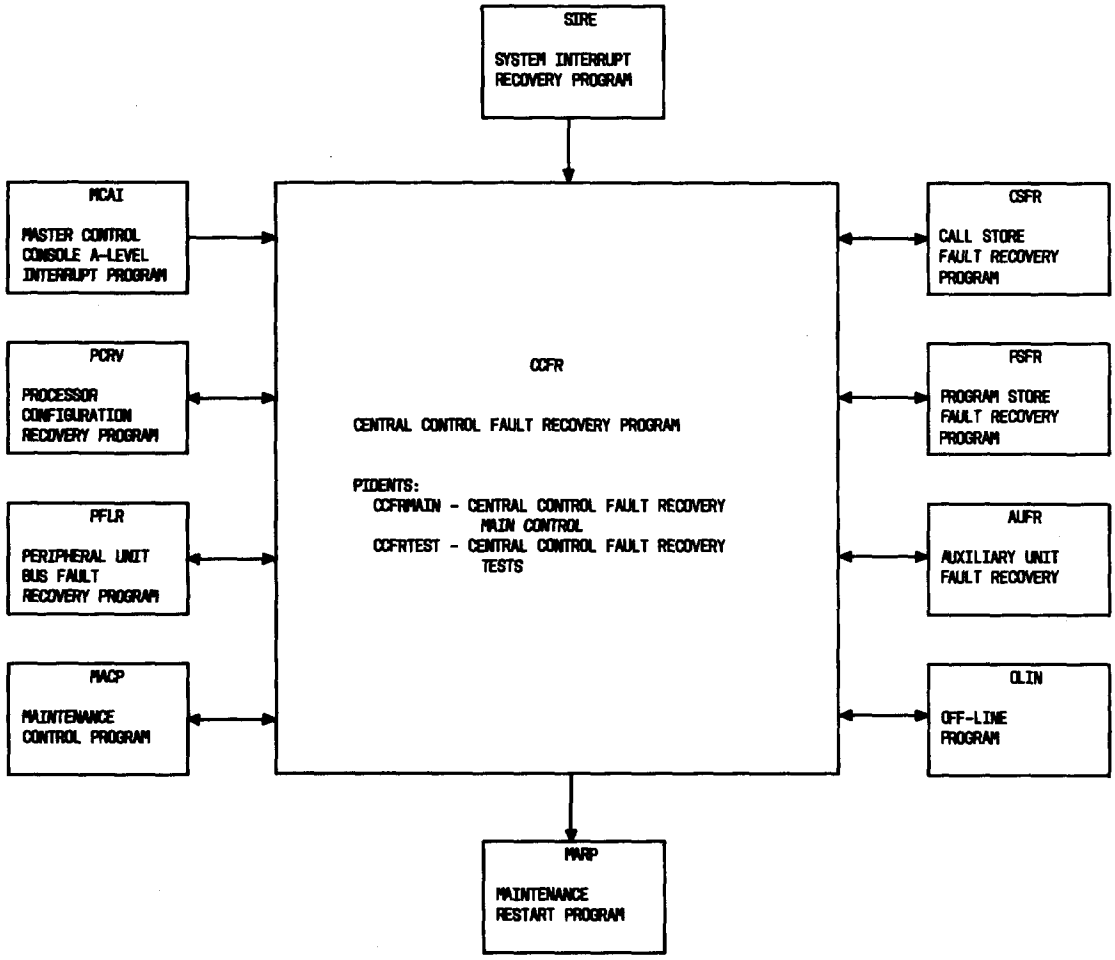


Fig. 2—Central Control Fault Recovery Program (CCFR)

These two routines are used by CCFR and other maintenance or recovery programs while testing the central controls or while testing access to associated subsystems using duplicate central controls.

3.06 The CCFR program maintains records of the number of C-level interrupts that are processed without detecting a machine malfunction. These are treated as errors or undetectable transient faults. Monitoring these errors is done by CCFR to determine whether the error levels (number of errors in a specified time period) are acceptable for system operation. When these error levels are exceeded, CCFR begins automatic diagnosing and/or switching of central controls in an attempt to isolate the source of the errors. Error messages and data are printed on the TTY whenever possible to assist the office personnel in identifying and isolating a faulty or marginally operating unit. Error data collected by CCFR is also made available to the 1A Processor Error Analysis Program (ERAP).

3.07 The CCFR program also serves as a test tool for the Processor Configuration Recovery Program (PCRV) to verify that the central control selected by PCRV is capable of basic order execution. Later in the recovery process, PCRV again uses CCFR to gain access to a good peripheral unit bus. However, CCFR does not configure any units on the peripheral bus system.

3.08 The CCFR program may also be used by other programs or by the office personnel to run a comprehensive or selective set of test routines on the central controls. In this mode, CCFR runs on base

level in the segmented mode so that normal call processing is not affected.

CENTRAL CONTROL ORGANIZATION

3.09 Duplicated central controls are the primary functional elements of the 1A Processor. The central controls interface with all internal and external signal and control buses and provide the processing capability for the 1A Processor. For reliability purposes, the two central controls are connected in parallel. Either one can control system operation. The normal system configuration provides for the two central controls to operate in step, each doing matching checks on the other.

3.10 One central control functions as the active unit and the other functions as the standby. During this normal mode of operation, both central controls are matched to ensure that they execute the same instructions, receive the same data, and make the same conditional decisions. If the active central control malfunctions, the standby central control is made active and assumes control of processor functions (the switch of active and standby central controls may be done automatically under program control, by the processor configuration hardware, or by manual activation from the control and display panel).

3.11 System troubles are normally detected by trouble detection circuits, and the call processing capability of the system is restored by the interrupt system with the associated fault recovery programs. The interrupt structure is a hierarchy of ten interrupt levels that are entered according to the severity of the problems encountered by the system. Central control malfunctions normally cause B- or C-level interrupts but may cause interrupts of a lower level that are intended to detect subsystem malfunctions (ie, E-level: failure to access program store). Processor configuration sequencer triggers, program requests to switch active central controls, and generate control pulse (GCP) failures cause B-level interrupts, whereas C-level interrupts are caused by a mismatch of cross-coupled data between central controls.

3.12 To ease recovery problems when a "sane" central processor is not available, the central control has a processor configuration sequencer that is started (triggered) when one or more conditions indicating loss of processing ability occur. The processor configuration sequencer establishes various combinations of central controls, base program stores, and program store buses without reliance on program instructions. The B-level interrupt programs are later used to verify the sanity of the assembled processor.

3.13 The central control has two timers that supply backup timing for interrupt recovery programs. The analog timer (started by the interrupt sequencer) supplies 100-ms timing for B-, C-, D-, and E-level interrupt programs. If the analog timer "times out," the processor configuration circuit is activated or reactivated. The processor configuration sanity timer enables detection of a nonworking configuration more quickly than 100 ms during B-level processor configuration recovery. This sanity timer times out after 496 central control cycles and will reactivate the processor configuration sequencer if not administered properly by the recovery programs.

> ***Note:*** For more detailed information about Central Control, refer to Section 254-201-031, Central Control—Theory.

CCFR—FUNCTIONS AND STRATEGY

A. General

3.14 Normally, CCFR operates on B, C, and base level. However, using the external entry, it can run on any other interrupt level or on interject. As stated earlier, its primary purpose is to verify the integrity of the active central control; however, the tests used to perform the verification vary and depend on the conditions under which the program is entered.

3.15 The CCFR program is organized around a common control program (Fig. 3) which calls one or more independent test routines. Control is based on a dynamic control word initialized by CCFR to specify which test routines to run on each entry. The control word is normally set to all zeros which is recognized by CCFR as an invalid entry condition. The control word consists of three fields of bits. Within the first field, each bit has a one-to-one relationship to a test routine that tests a portion of the central control. The second field consists of a single bit and signifies a special test procedure used only on B-level interrupts caused by a pulse source failure. The remaining bits make up the third field and show the origin and the termination of the request to run CCFR.

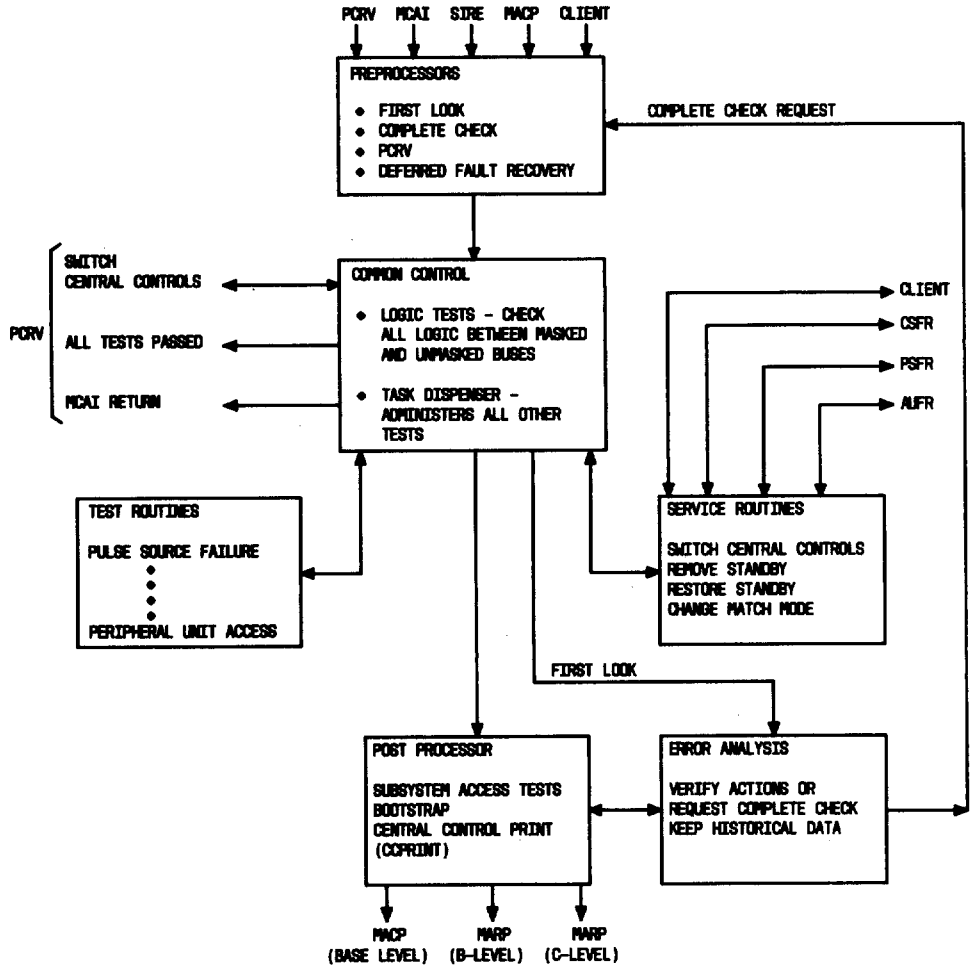


Fig. 3—Program Organization—CCFR

3.16 There are several different entry points in CCFR, each with its own unique requirements. A pre-processor program is provided for each input to prepare the necessary information and perform the required initialization before entering the common control program. At the conclusion of the common control program, a termination program is provided for each entry. There is also a special termination program if CCFR is entered invalidly (common control program finds the control word to contain all zeros). These terminating programs perform access tests on other subsystems, update status words, request subsystem normalization, update error counters, and in general do cleanup and housekeeping tasks before returning the system to call processing via MARP, PCRV, or the Maintenance Control Program (MACP).

3.17 Entry into any of the CCFR control programs requires that some hardware and/or software actions be taken before the program is entered. These requirements vary based on the level of the interrupt, the cause of the interrupt, or the response expected by the user when the request is for a deferred CCFR. Likewise, after the common control program has completed, some means must be provided to return the central controls to normal operation. These functions are done by several pre-processor and post-processor control programs. However, in many instances, the actions are so similar that parts of the program are used for more than one entry condition. Only the control word or a bit in the status word identifies the entry condition.

3.18 The CCFR program's control structure can best be characterized in terms of C-level inter-

rupts, B-level interrupts, external program requests, and deferred fault recovery requests. Each of these control structures will be discussed along with entry requirements.

B. C-Level Interrupt

General

3.19 The C-level program is entered because of a mismatch between the two central controls. The occurrence of the C-level interrupt stops all matching and freezes the contents of all matchers. This is ensured by the way the match control registers in central control are set up during normal operation. To ensure a proper setting of these registers, only CCFR is allowed to set up or change a match mode that is to exist during call processing.

3.20 The interrupt sequencer stops the standby central control and sets the active central control to send on both program store buses and both call store buses. It also starts the analog timer as a precaution against a faulty central control becoming hung up in a loop in the interrupt recovery program. A software check of the direct transfer capability of the central control is done in SIRE by attempting a direct transfer to CCFR. If the transfer fails, central control performs an "EXC" instruction whose address is the address of the "EXC" instruction itself. This halts the central control until the analog timer times out and generates a B-level interrupt.

3.21 For C-level interrupts, there are two control programs, each consisting of a pre-processor program and a post-processor program. When these are combined with the common control program, they form a complete control structure for the processing and disposition of all C-level interrupts. Each of the programs was designed with a different philosophy in mind. The first-look program uses as little real time as possible and tests only a selected area of the central control. The complete check program runs all available test routines of CCFR in addition to interface tests with other subsystems.

First-Look Program

3.22 The term "first look" as used by CCFR is to define the degree of testing and the amount of real time to be used before any decision is reached about the sanity of either central control. In addition to conserving real time, the first-look control program performs four other functions.

(1) The first-look pre-processor program stores the data contained in the internal registers and buffer bus registers of both central controls at the time of the interrupt.

(2) The first-look pre-processor program does a series of tests on the decision-making capability of the active central control to determine if there is any need to continue testing.

(3) The first-look pre-processor analyzes the error indicators that were frozen at the time of the interrupt. From this analysis, it selects a set of test routines to run on the central controls that are most likely to uncover the cause of the interrupt.

(4) The first-look post-processor places the central controls in a state that is suitable for the resumption of call processing.

3.23 The CCFR first-look program is entered at global location CCFRFLE from SIRE. Its first task is to complete the storage of a selected set of registers [paragraph 3.22(1)] from the active and standby central control into a data table. This table is used to store data that the restart program, MARP, prints as a part of the system interrupt message. When the table is loaded, a flag is set for MARP to show that the table data is valid.

3.24 Before any test routines are set up, a test of the logic and decision-making ability of the active central control is made. This test only goes far enough to test the type of decision normally used to end the test routine. The test program is arranged in a maze requiring the central control to correctly execute a sequence of conditional transfers to reach the successful termination. A failure anywhere in this program causes control to be transferred to the complete check program.

3.25 After the conditional transfer tests have been successfully completed, CCFR loads two more data tables. The first table will contain 64 registers, shadow registers, and buffer bus registers, from the active central control. The second table will contain the same information from the standby central control. This data will also be printed by MARP at the conclusion of the interrupt.

3.26 The CCFR program uses many subroutines, most of which use the contents of the J regis-

ter as the return address. The first-look program does a crude check of the operation of the J register return option. If the J register fails the tests, the central controls are switched. Before doing a test of the transfer capability using the J register return, the processor configuration sanity timer is started to protect against severe program or data mutilation if the transfer fails. If the transfer fails, the timer will time out and cause a B-level interrupt in about 0.35 ms. If program control returns to either of the expected points, the sanity timer is turned off.

3.27 When all the previous tests have passed and all the interrupt associated data has been stored, the first-look program analyzes the error indicators to determine which test routines to run. A check is made to determine if routine matching was in progress in the active central control. If it was not, internal match error indicators are ignored. Likewise, if routine matching was not in progress in the standby central control, the external match indicators are ignored. If, after this filtering process is done, there are no valid match error indicators set, control is transferred to the complete check program. Also, one and only one instruction class indicator will have been set when the interrupt occurred. If not, the complete check program is run.

3.28 The interrupt sequencer stops matching and stops the standby central control for all maintenance interrupts except F-level interrupts. Therefore, it is possible for a C-level interrupt to occur while the F-Level Fault Recovery Program (PFLR) is running. Here, the indicators that are frozen when the C-level interrupt occurred may be misleading to the first-look program. Consequently, a check is made to see if the F-level activity flip-flop is set; and, if it is, the complete check program is run to ensure that the proper part of the central control is tested.

3.29 For the purposes of matching, the 1A Processor instructions are grouped into six classes. The classes were derived from the internal gating associated with the instruction and not necessarily with the function the instruction performs. The first-look program has six data tables that correspond to the six instruction classes. Each of the data tables contains four words that correspond to the four match error indicators. These words contain flag bits that show the test routines that are appropriate for a particular mismatch on a particular instruction class.

3.30 The first-look program determines the class of instruction being executed at the time of the C-level interrupt and also determines which matchers failed. The data table words that correspond to the class instruction and the matcher error indicators are then ORed to form a composite control word (Fig. 4).

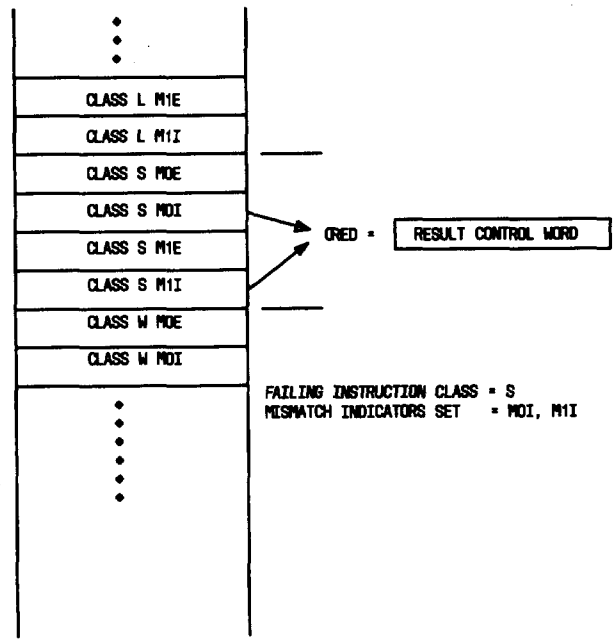


Fig. 4—CCFR—First-Look Control Word Generation

3.31 After setting up the control word, all units on the auxiliary unit bus (AUB) are inhibited so no interference occurs on the call store or program store bus systems between the central controls and some unit on the AUB. The first-look program also repeats some of the functions done by the interrupt sequencer. This includes setting the program and call store routing control flip-flops (PBO, PBT, CBO, and CBT) to make the active central control the controlling unit for all functions. The call store and program store status words are also updated.

3.32 The pre-processor portion of the first-look program completes its job by attempting to put the central controls back in step. If it can, routine matching without interrupts is initiated between the central controls. Otherwise, all matching is stopped, and a flag bit is reset to show the standby central control is not being tested. The first-look program

then transfers to the common control program to process the interrupt.

3.33 The first-look program can end in one of three ways. First, all selected tests might run but fail to find a fault. This could be due to a transient error causing the interrupt or the first-look program failing to select the proper set of tests to detect the fault. For this condition, the first-look post-processor program keeps a count of the number of times that the first-look program passes all tests. When this counter reaches a value of 2, a complete check is run during the same interrupt interval as the first look.

3.34 Second, the standby central control might fail one of the first-look tests. This is detected by a mismatch between the central controls. However, even though a mismatch occurred, it does not necessarily mean the standby was at fault. The mismatch could be caused by a fault in the active central control matchers. So, an additional test is made on the active central control matchers. If this test passes, the standby central control is removed from service and a diagnostic is requested.

3.35 Finally, the active central control may fail. An active failure may be detected anywhere in any of the test routines. Also, the active central control is faulty if a mismatch is detected and the active matcher test fails. Any active failure results in a transfer to the central control switch routine and subsequently the generation of a B-level interrupt. The only factor preventing the central controls from being switched is if the TCC flip-flop (central control in trouble—standby out of service) is set. This should never be the case in the first-look program.

3.36 If either a standby fail or a pass termination occurs, a special routine is provided to return the system to a call processing configuration. This routine is also used by the complete check program and the B-level program. This routine, called CCPRINT, first removes the standby central control from service. This is done as a precautionary measure to ensure that the standby central control is completely isolated from the active system and that all status words and the matcher status table are up to date.

3.37 During some of the peripheral tests, a noncritical test may have failed. In this case, a status word flag is set. When CCPRINT finds this flag set, it prints a TTY message identifying the failing test. If the standby central control failed a test, a failure message is printed to identify the point at which the failure was detected. A report action phrase is loaded into the MARP output message. For the central control, this phrase always states that CCFR removed the standby central control.

3.38 If the storage of the active and standby data tables was successful, a report is made to identify to MARP the beginning address and number of words in each table. This information is also used by error analysis. If the central controls were switched because of a failure during one of the tests, the standby fail data table is not meaningful. Here the active fail data is printed instead of the standby fail table.

3.39 After all prints have been dispensed with, the program clears all scratch memory used by CCFR to prevent possible conflicts on later interrupts. If the standby central control failed, it will be diagnosed unless the excessive error rate flag is set. The return to reference point flag is set so that MARP will not attempt to unwind or roll back to the interrupted program.

3.40 At this point in the program, the appropriate call store and program store bus routing flip-flops (CBO and PBO) are set so that the active central control writes into call store and program store over both buses. At the time of the interrupt, depending on the type of order and the matching in progress at that time, any matcher might contain the address of some memory location that mismatched. Rather than filter out the different possibilities, CCFR assumes that all matchers contained addresses. The CCFR program reads from the location shown by the contents of a matcher and writes back into the same address using a maintenance store instruction. This guarantees that both copies of duplicated memory locations, whether secure or nonsecure, are the same but not necessarily correct. This read and restore process is only done for addresses in the call store and program store address ranges. This same procedure is followed for the addresses contained in the active and standby central control save data address registers. After all memory has been updated, requests are made via MACP routines to normalize the call store and program store buses. This is done later as a low priority MACP job on base level after the maintenance restart.

3.41 The remainder of the CCPRINT routine is devoted to cleaning up any error indicators in

the central controls. At the end of the CCPRINT routine, there is a decision made on the highest level activity flip-flop that is set. If C level is the highest set, CCPRINT transfers to restart at MARPCLEV. If B level is set, a decision must be made on what to do with the standby central control. If either the TCC flip-flop or the CCHE (high error rate) flag is set, the standby is left out of service. If both are reset, a deferred request is made to restore the standby to service. The program then ends by transferring to MARPBLEV.

Complete Check Program

3.42 The complete check program operates on C level and serves as a backup to the first-look program. It can run all the test's routines provided by CCFR. It also does access tests on other subsystems. If C-level interrupts, for which no fault can be found, occur at a rate that is unacceptable for system operation, the complete check program will initiate a central control diagnosis and/or switching of the central controls.

3.43 The complete check program can be entered from the first-look program or from the external program request. The complete check program is entered from the pre-processor portion of the first-look program when some exceptional condition is encountered that the first look cannot handle. Here, the control words have not been set up. The complete check program loads these words to run all test routines and to indicate the complete check program. Bus control flip-flops CBO, CBT, PBO, and PBT are set to ensure that all duplicated memory remains updated.

3.44 The complete check program may also be entered from the first-look program when the first-look passed counter exceeds its threshold value. Here, the control words have already been set up by the first-look program. The complete check program will run only those routines that were not run by the first-look program. Since both the first look and the complete check are run on the same C-level interrupt, repetition of the tests in the complete check that have already passed the first look would only serve to consume more system real time.

3.45 After the control words are set up, the complete check program attempts to put the central controls in step. If successful, routine matching without interrupts is set up between the central controls; otherwise, all matching is stopped. The complete check program then stops all units on the AUB and transfers to the common control program to process the interrupt.

3.46 Like the first-look program, an active central control failure could be detected at any place in any of the test routines. If a failure occurs, the same actions take place as in the first-look program. Likewise, a standby central control failure is detected by a mismatch between the two central controls. And, as in the first-look program, an active matcher test is run before condemning the standby central control. The complete check could also pass. Here, control is returned to the complete check post-processor program. Routing flip-flops CBO, CBT, PBO, and PBT are set and their status words are updated.

3.47 If the complete check passed on the active or both central controls, a subsystem access test is run. If the standby central control failed, the subsystem access test is run using only the active central control. The subsystems involved in the access tests are the call store, program store, and the auxiliary unit (AU) subsystems. Peripheral system access tests are not done because of time limitations.

3.48 The call store community is accessed first. The Call Store Fault Recovery Program (CSFR) contains an access test that is used by CCFR to verify access to all call stores currently in use by the system. If the access tests pass, CCFR goes on to the program store access test. However, if the access test fails, immediate actions are taken to restore a complete copy of call store memory for system use. The CCFR program first attempts to bootstrap the call store community. A bootstrap routine is so named because it figuratively "raises a subsystem by its bootstraps" establishing a configuration based on limited access tests. A routine in CSFR is called to do the bootstrap. If the bootstrap passes, then CSFR has assembled a complete copy of all call stores and CCFR proceeds to the program store access test.

3.49 If the bootstrap fails, CCFR will try to switch central controls. If the switch is successful, a B-level interrupt is generated. If for some reason CCFR cannot switch central controls or if the central controls have already been switched, CCFR will transfer to PCRV to deliberately start the processor configuration circuits to try to assemble a working system.

3.50 The program store access test is identical to the call store test with one single exception

CCFR uses a Program Store Fault Recovery (PSFR) routine to do the tests rather than a CSFR routine. If the program store access test is successfully completed, the AU subsystem access test is run. Before running the access test, CCFR calls an Auxiliary Unit Fault Recovery Program (AUFR) subroutine to stop each individual unit connected to the AUB system. The CCFR program then uses another AUFR routine to do the access test. If the access test is successful, CCFR will end the subsystem access test routine. If the access test fails, a TTY message is printed identifying the AU failure.

3.51 If all the subsystem access tests pass and the standby central control was running, a test is made to see if there was a mismatch while running the access tests. If any mismatches were detected, an active matcher test is run to determine which central control is to blame. If no mismatches were detected and the B-level activity is set, control is transferred to CCPRINT to end CCFR's part of the interrupt. However, if all tests pass and the B-level activity is not set, an excessive error rate procedure is run.

3.52 Sometimes, due to either the test environment or to the extent to which CCFR can exercise a certain portion of the central control, faults undetected by CCFR can be detected by the Central Control Diagnostic Program (CCDG). Also, errors or marginal faults occur that neither CCFR nor CCDG can detect. To prevent the system from bogging down with C-level interrupts, CCFR maintains several counters to limit the number of C-level interrupts allowed.

3.53 When a C level occurs that does not result in a fault detected by the first-look program, a first-look all tests passed counter is incremented. The value of the counter is then checked, and if the value is equal to or greater than the threshold value for the counter, a complete check is run on the same C-level interrupt. If the complete check should also pass, a complete check of all tests passed counter is incremented. If the counter is less than its threshold value after incrementing, the complete check transfers to CCPRINT to complete the interrupt. However if the counter is equal to or greater than the threshold, then the frequency of C-level interrupts which yield no failures in CCFR is excessive. The CCFR program takes further actions to isolate the faulty unit.

3.54 The central control status word contains flags that show the history of actions that CCFR has taken when the rate of C-level interrupts was determined to be excessive. Based on prior actions, CCFR may take any one of the following actions:

(a) If the standby central control has not been diagnosed, CCFR removes the standby from service, requests a diagnosis on the standby, resets all error counters, and prints a message that the standby is being diagnosed due to a high rate of C-level interrupts.

(b) On the other hand, if the standby central control has already been diagnosed due to excessive errors, then CCFR checks the flag for the active central control. If the active central control has not been diagnosed because of excessive errors, CCFR resets all error counters, prints a message that the central controls are being switched because of a high rate of C-level interrupts so that the new standby can be diagnosed. The status words are updated to show the action.

(c) Finally, if both the active and the standby central controls have already been diagnosed because of excessive errors, CCFR updates the status words to show that the central controls will not run in step without a high rate of C-level interrupts. The CCFR program then resets all error counters, removes the standby central control from service, and prints a message that the central controls will not run without excessive errors. No diagnosis is requested on the standby. The complete check then ends by transferring to the CCPRINT routine.

3.55 The excessive error rate counters and flags are all reset after a specified time has elapsed regardless of the number of C-level interrupts that has occurred. However, the flag that shows the central controls cannot run in step without excessive C-level interrupts can only be reset by an unconditional restore message from the TTY. Also, any system reinitialization wipes out all error history to prevent inaccurate data in the error counters and status that may cause premature reactions by CCFR.

3.56 The above procedure handles excessive C-level interrupts that occur in a short period of time. However, two other conditions exist that affect system operation. The first condition occurs when CCFR fails to detect a fault that could be detected by CCDG and interrupts are not occurring often enough to trigger a high error rate procedure and thus a request for

CCDG. To overcome this problem, CCFR maintains a long-term C-level count that is incremented on a C-level interrupt whenever the first-look and complete check test passed counters are less than their threshold value. The first-look and complete check test passed counters are reset by a regularly scheduled external timing routine; however, the long-term C-level count is reset only once daily or by either a central control switch or a request to run CCDG. Whenever the long-term C-level count exceeds its threshold value, CCFR performs the same procedure that is used whenever high error rates are detected by the complete check test passed counter.

3.57 The second condition occurs when some other program requests that CCFR do a complete check. Although this is not a true C-level interrupt condition, the same tests are run and the complete check uses as much of the system's real time as for a C-level interrupt. The CCFR program maintains an external request complete check counter that is incremented each time a complete check is run as the result of an external request if no faults are found. Whenever the external request complete check counter reaches its threshold value, CCFR performs the same procedure as for the other excessive error rate conditions.

3.58 This combination of counters and flags allows CCFR systematically to limit the amount of system time devoted to handling C-level interrupts whether failures are detected.

C. B-Level Interrupts

General

3.59 The B-level interrupts are of three types:

(a) B levels caused by a time-out of the analog timer, long timer (program sanity timer), processor configuration sanity timer, or a trigger from the processor configuration circuits.

(b) B levels caused by a failure in the operation of the pulse source generation circuit or in the pulse source failure detection circuit. This can also be caused by a program that executes a generate control pulse (GCP) instruction with an invalid data field.

(c) B levels caused by pulse source switch of central control.

3.60 When a B-level interrupt occurs, the interrupt sequencer stops matching, stops the standby central control, sets the active central control to send on both program store buses and on both call store buses, starts the analog timer, and forces a transfer of program control to the B-level entry to SIRE. Also, if the B level was caused by a processor configuration trigger, the processor configuration sanity timer is started.

3.61 On any B-level interrupt, some part of CCFR is run as a part of the recovery process. Program PCRV receives program control from SIRE. The storage of interrupt data is done by SIRE and PCRV. The PCRV program then decides whether it will control the processing of the interrupt or whether CCFR will take control.

Processor Configuration

3.62 A processor configuration is caused by the time-out of the program sanity (long) timer, analog timer, or processor configuration sanity timer. It may also be caused by the processor configuration circuit if the circuit determines that both central controls are either active or standby. The processor configuration circuit then systematically configures the system until all possible combinations of K-code 20 program store buses, file store controllers, and central controls have been tried or until a usable configuration has been found.

3.63 After PCRV has found a usable configuration of hardware, PCRV tests the current system's sanity by calling other recovery programs to verify the processor subsystems. The CCFR program is the first subsystem program called by PCRV. For the sanity test, CCFR provides a control structure similar to the complete check program that runs a selective set of CCFR's test routines on the active central control only. The PCRV program then calls subsystem tests for program stores, call stores, and file store controllers. After these subsystems have been tested, PCRV returns to CCFR to select a good peripheral bus. A special entry directly into CCFR's test routine is provided for PCRV.

3.64 Since CCFR is entered on all B-level interrupts, CCFR must determine the cause of the particular interrupt. If the interrupt was caused by the processor configuration circuits, the processor configuration sanity timer will be enabled. If the sanity timer is enabled, CCFR uses this as an indication that it is to return to PCRV and to test only the basic order processing capability of the active central control.

3.65 Upon entry from PCRV, CCFR stops all units on the AUB and checks to see if the sanity timer is enabled. If it is enabled, CCFR loads the control words with the processor configuration control bit and with a selective set of test routine request bits. The standby central control is then removed from service. The removal is more of an initialization process than a configuration procedure since the standby was stopped by the interrupt sequencer.

3.66 The sanity timer is recycled and control is passed to the common control program to process the interrupt. Throughout the test routines, the sanity timer must be recycled about every 450 central control cycles to prevent another B-level interrupt. (The sanity timer actually times out after 496 cycles.)

3.67 If the active central control passes all tests, control is returned to a predetermined point in PCRV. However, if the active central control fails a test, control is transferred to the central control switch routine. For this entry into the switch routine, a program delay loop is entered which causes the sanity timer to time out and generate another B-level interrupt. This advances the processor configuration state counter to the next state and produces a new configuration to test. This procedure continues until the active central control passes all tests.

3.68 When PCRV uses CCFR to select a good peripheral bus, PCRV transfers directly to the peripheral bus loop-around test. The test routine is performed using the peripheral bus loop-around circuits in the peripheral interface frame and thus are independent of the peripheral units. The CCFR program will switch buses until it has tried all combinations before it will give up and return control to PCRV. Program CCFR, on this entry, will make no attempt to switch central controls. Two returns are provided in PCRV: one if CCFR was able to find a good configuration and another if it was not. On the fail return, CCFR passes information to PCRV so that the reason for the failure can be displayed at the master control console (MCC). During the loop-around test, the analog timer must be running and the sanity timer must be turned off. It is the responsibility of PCRV to meet these entry requirements.

Pulse Source Failure

3.69 The 1A Processor has 360 pulse sources which control configuration and status flip-flops within the 1A Processor System. These pulses are provided by a circuit in central control. The pulse source is activated when a generate control pulse (GCP) instruction is executed by the central control. An error detection circuit is connected to the outputs of the pulse source generator circuit to ensure that one and only one pulse source lead is pulsed on a GCP instruction and that no pulse sources are generated on a non-GCP instruction. When the detection circuit detects either none or more than one pulse on a GCP instruction or any number of pulses on a non-GCP instruction, it causes a B-level interrupt, if the inhibit pulse source failure flip-flop is not set.

3.70 The CCFR program contains the control program and test routines necessary to determine if the pulse source failure was caused by a fault in the active central control, by a programming error, or by a nonreproducible transient condition. A special pulse source failure routine is included in CCFR and is used with the logic test and the subsystem access test routines to make up the complete pulse source failure test sequence.

3.71 For pulse source failures, PCRV transfers program control to the B-level filter program in CCFR. If CCFR determines that the interrupt was not because of a processor configuration trigger, it checks to see if the pulse source failure indicator is set and also if its inhibit is reset. If they are, CCFR assumes the interrupt is because of a pulse source failure. The CCFR program then sets the pulse source failure inhibit to prevent any further B-level interrupts due to a GCP instruction used by CCFR. A control word is then built by setting the pulse source failure control bit and the pulse source failure test routine bit. The standby central control is removed from service. Again, this is for the purpose of initialization since the interrupt sequencer has already stopped the standby. Control is then passed to the common control program to process the interrupt.

3.72 The pulse source failure test routine is called by the common control program to test the central control for a failure. If central control was at fault, the test routine attempts to switch active central controls (an unsuccessful attempt would cause a processor configuration). If central control was not at fault, control is returned to the common control program. The control program transfers to the subsystem access tests. This is necessary since the GCP instruction that caused the interrupt may have been associated with a store routing flip-flop; consequently, there exists the possibility that the system does

not have access to a complete copy of memory. Also, the status of the stores and the actual configuration may not agree. This situation must be resolved by the subsystem access tests before returning the system to call processing. The access test transfers control to CCPRINT. The CCPRINT routine in turn transfers control to MARP.

Central Control Switch

3.73 When the central controls are switched using the GCP instruction, a B-level interrupt is generated; afterwards, CCFR is entered to verify the new active central control. This is because even though the new active central control may have been tested thoroughly while operating as standby, there is a chance that it will not run properly as the active central control. The B-level interrupt starts the analog timer if the central control clock does not operate properly.

3.74 Upon entry, if the B-level filter of CCFR does not find the processor configuration sanity timer enabled or the pulse source failure indicator set, it assumes that the interrupt was due to switching the central controls. The B-level test of the central controls is equivalent to the C-level complete check program when all tests are run. When the control program recognizes the interrupt as a central control switch, it loads the control words with the B-level control bit and all test routine request bits.

3.75 The standby central control is removed from service for initialization purposes. If the standby central control is available at this point, the central controls are put in step and routine matching without interrupts is started. The CCFR program then sets a flag to indicate to MARP that the B level was because of switching central controls. Control is then transferred to the common control program to process the interrupt.

3.76 If the active central control fails any of the tests, CCFR actions are the same as described for C-level interrupts. If the active central control passes all tests, the subsystem access tests are run. If a failure is detected anywhere in the access tests, the actions are the same as for the C-level complete check program. When the access tests are complete, control is transferred to the CCPRINT routine to complete CCFR's part of the interrupt.

D. External Program Request

3.77 The external program request is another way to enter the complete check program. The external program request entry point provides a means for any other interrupt program of a lower interrupt level, or an interject program, to run a complete check of the central control community. An external program request does not cause a C-level interrupt; however, CCFR treats it the same as it does a hardware-generated C level.

3.78 On entry to the external program entry point (CCFREXFR), the control program sets all interrupt inhibits except pulse source failures. (A-level interrupts and B-level processor configuration sequencer and central control switch interrupt sources cannot be inhibited by the program.) The control program also stops the AU sequencer, sets a flag in the status word indicating an external entry request, and inhibits the analog timer before entering the normal complete check program.

3.79 In the complete check termination routine, the normal complete check and the external request are differentiated by the status word flag. If a failure is detected, the actions of CCFR are the same as those described in the C-level complete check. However, if no errors are detected, the external program request test passed counter is incremented and checked. If the counter has reached its threshold value, the high error rate procedure is done to determine appropriate action.

3.80 Also, it should be stressed that once a program transfers to the external entry it has given up all program control. No return is made to that program and no information about the result of the central control tests is returned to it. The final disposition by CCFR will eventually be a return to the main program reference point.

E. Deferred Fault Recovery

3.81 The deferred fault recovery program can be considered a "mini" diagnosis of the active central control. It runs all the tests of the complete check program in a segmented mode that is administered through MACP on base level. Each of the program segments is limited to 2.5 ms of time. A termination bit in the control words and six control bits in the status word direct the actions of deferred fault recovery. The control word termination bit differentiates deferred fault recovery from all other entries, and the status word bits direct deferred fault recovery to the functions it should do.

3.82 Deferred fault recovery may be requested from the TTY or by a program request that

places an entry in the MACP job table request. The CCFR program provides the necessary routines for making the MACP job table request. Deferred fault recovery is not started until it has the highest priority of any pending MACP job request and does not interfere with an MACP job already in progress. The MACP program assigns deferred fault recovery a temporary block of memory (MACP scratch pad) and passes two words of information to deferred fault recovery at run time. The first word identifies the channel over which a TTY request was received or contains an unassigned channel identity when the request did not come from a TTY. The second word contains the option bits for the status word and test bits for the control words. The CCFR program supplies the deferred fault recovery termination bit for the control words.

3.83 Deferred fault recovery may be requested for one of two purposes. First, it can be requested to run tests on the central control community. If these tests pass, the central controls are restored to the configuration that existed when MACP started deferred fault recovery. This may not be the configuration that existed at the time of the original request. Secondly, deferred fault recovery may also be called to restore the standby to service without running any tests. This is the normal way of putting the central controls in step with routine matching for normal system operation. In either case, deferred fault recovery passes an abort address to MACP. If an interrupt occurs while central control fault recovery is an MACP client, MACP will transfer program control to the abort routine during MACP's first scheduled segment after the interrupt recovery. The abort routine cleans up any memory currently being used by deferred fault recovery, prints an appropriate output message, and ends the deferred fault recovery request.

3.84 A special segment routine is provided to interface between deferred fault recovery and MACP. This routine is used for all but one segment break. The exception is the segment break immediately before the AU access test. This access test requires that each individual AU be stopped before the access test is started. Generally, a test routine is run as a single segment and the segment time break is taken in the common control program. However, because of length, some test routines must take internal segment breaks. In these cases, the mechanism for determining when and if a segment break will be taken is embedded in the test routine itself. The test routines contain enough defensive checks to ensure that a segment break will not inadvertently be taken as part of an interrupt recovery procedure. Never is more than one test routine run during a single segment.

3.85 A program request automatically runs deferred fault recovery on both central controls if the standby is available. When finished, it prints either an all tests pass (ATP), active failed, or standby failed message on the TTY.

3.86 On the other hand, when deferred fault recovery is requested by a ("TEST:CC") TTY message, several options exist that are not available on a program request. A TTY request may specify that either both central controls or only the active be tested. Also, the request may specify the test routines that are to be run. This can be a single test, several isolated tests, a range of tests, or any combination of these.

3.87 Another feature of the TTY request allows deferred fault recovery to loop two or more test routines. The logic test is always run and the TTY message must specify at least one test routine to be accepted as valid. When looping, deferred fault recovery executes the entire set of specified test routines, except for the subsystem access tests before starting over again unless a failure is detected in one of the test routines. Here, deferred fault recovery executes all specified test routines until the first failure is detected and then starts over again. The looping mode is ended by a maintenance interrupt or by a TTY message that ends it as an MACP client.

3.88 It should be pointed out that the TTY request can select the test routines to be run, but it cannot specify the order in which they will be executed. Deferred fault recovery uses the same common control program as any other CCFR control program, and the task dispenser portion of the common control has a fixed sequence in which it calls test routines.

3.89 In addition to normal program requests and TTY requests, deferred fault recovery is run after every central control diagnosis and as a part of the daily routine exercise of the central control. For these two entries of deferred fault recovery, the Diagnostic Control Program (DCON) initializes a diagnostic buffer table. The table contains all the information about the origin of the request, the results of the diagnosis, and what action is to be taken.

These two entries are ended by the CCFR diagnostic final handler routine.

3.90 For looping, output messages are suppressed except for the abort message that occurs when the request is stopped. Also, deferred fault recovery passes data to the system error analysis program if either the active or standby central control fails. If the standby fails, it is left out of service and a diagnosis is requested. If the active central control fails, the standby is left out of service but no diagnosis is requested. Information about the failure is printed on the TTY, and it is up to the operating personnel to take the appropriate action.

CCFR—PROGRAM STRUCTURE

A. General

3.91 The CCFR program (Fig. 2) consists of two pidents: CCFRMAIN and CCFRTEST. Pident CCFRMAIN contains the CCFR control, service, and test routines that are essential to system recovery and is located in K-code 20. Pident CCFRTEST contains the remaining test and service routines that are not required for system recovery.

3.92 The common control routines have already been described in terms of C-level interrupts, B-level interrupts, external program requests, and deferred fault recovery requests. The test routine and service routine descriptions follow.

B. Test Routines

3.93 Special instruction sequences are required to test specific central control circuits for all input conditions. Each test routine tests as many of these input conditions as possible for a specific partition of central control hardware. Sometimes, it is impossible to test all the circuit input combinations. This is because the CCFR must run in the active central control and test it at the same time. Consequently, CCFR cannot test circuitry as completely as the central control diagnostic program that runs in the active central control and tests the standby.

3.94 Test routine failures in CCFR may in some cases be misleading. For example, the execution of the first instruction of CCFR depends on proper operation of the instruction fetch sequencer, order word decoder, system clock, various gating functions, etc. A fault in any one of these circuits could cause the improper execution of this first instruction. However, the objective of CCFR is to determine whether central control is operating properly, and identification of the exact cause of a failure can be determined by later tests performed by the diagnostic program (DCON). The CCFR program uses the following test routines with access tests in other fault recovery programs to test as much of the central control hardware as possible.

Logic Test

3.95 The logic test routine is entered at entry LOGIC_TEST to do a comprehensive test on the logic circuitry. These tests include the L register and combined mask homogeneity circuits, the compare circuits, the AND, OR, and EXCLUSIVE OR logic functions, the right-most one detector circuit, the insertion mask, the 16- and 24-bit rotators, and the size and displacement register and translator circuit. A large part of the central control is exercised during these tests, and troubles in other parts are expected to be manifested during these tests. If the tests pass, control returns to the common control program. If a failure is detected, the proper test is marked and an attempt is made to switch the central controls.

Register and Homogeneity Logic Test

3.96 This test is designed to test all the registers located between the masked bus and the unmasked bus of the central control. This routine is entered at the REGTEST entry to test the central control index registers, stack register, enable register, reply register, and the peripheral unit data register. These tests include the ability to set and reset each individual bit of a register and shadow register if so equipped. The ability to correctly set the sign and homogeneity flip-flops on a register test instruction is tested. The ability to select one and only one register is tested. Checks are made that detect leakage from the masked bus to a register, from a nonselected register to the unmasked bus, and between registers and shadow registers. If the tests pass, control returns to the common control program, and if not, an attempt is made to switch the central controls.

Buffer Register, Insertion Mask, and Program Store Access Test

3.97 This routine is designed to test the access from the buffer register to the program store,

and access to the buffer register. This routine is entered at entry BR_TEST. Both right and left halves of a program store word are used for this test and each half is checked for the expected results. The insertion masking is checked on a write into program store. If the tests pass, control returns to the common control program. If failures are detected, the central controls are switched and diagnostics are requested on the new standby central control.

Index Adder Test

3.98 This routine is designed to test registers and functions associated with the index adder. The registers tested include the index addend register, the index augend register, the data address register, and the save data address register. This routine is entered at entry point IA_TEST and checks the addition ability of the index adder, and the access to the save data address register. All addition instructions do not contain a full 24 bits of data for the index addend register and the sign bit may be in one of several positions. Therefore, on execution the data field is right-adjusted and the sign placed in the next least significant bit outside the data field and expanded to fill the remaining bits of the index addend register. The ability to load the index addend register with all combinations of data fields and sign bit locations is tested.

3.99 In the checks just done, the index augend register is loaded with stable data from an index register and is referred to as NO MIX operation. When the data for the index augend is to be gated from an index register that has not yet received the data, the data must be gated from the masked bus to the index augend at the same time it is being gated to the index register. This is referred to as MIXED operation. Checks are made to ensure that these gating paths are working properly. Attached to the index adder is an all zero detector. When this detector finds all zeros at the output of the index adder, and the instruction being executed is a store instruction, and no index register is specified, the gating from the index adder to the data address register is inhibited. This provides the "write at present address" function because the data address register contains the address of the last read operation. Tests are done on the all zero detector to ensure its proper operation. If the tests pass, control returns to the common control program. If failures are detected, the central control is removed from service and diagnosed if possible.

Add One Register and Logic Test

3.100 This routine verifies that the add one option is operating properly. This routine is entered at entry point AOL_TEST and makes a series of tests that check the add one option in the NO MIX environment. Additional checks are then done for the add one option in the MIXED environment. If all tests pass, control returns to the common control program; otherwise, the failing registers are saved, the central controls are switched, and failing data outputted via the TTY.

Parity Test

3.101 The parity test routine tests several parity generation and check circuits in the central control. The routine is entered at entry point PTY_TEST. The checks are made in a series of tests, the first of which is the data address parity generator, and LOKP flip-flop. Next is a series of checks that test the data parity generator without writing into memory, and the data parity check circuits. These circuits check the parity on all data reads from memory regardless of from where the data is read. Finally, the parity check circuits associated with fetching instructions from program store are tested. Both right-half and left-half parity check circuits are exercised. The program address parity generator is not tested except to verify that it is not generating bad parity that would cause the program store fetch parity check circuits to fail. If the tests pass, control returns to the common control program. If failures are detected, the active central control is removed from service for diagnostics if possible.

Protected Area Test

3.102 The protected area test routine is designed to check the upper and lower protected area registers and matchers. The routine is entered at entry PATST and initially checks the ability to set and inhibit the setting of the protected area D-level interrupt source. The lower and upper protected area address registers are then loaded with data, and regular store operations are done to locations within and outside the protected area to check for proper operations. The TOLL flip-flop is either set or reset for these tests depending on the office type and conditions under which the tests are run. On completion, the upper and lower protected area registers are restored to parameter values and the TOLL flip-flop restored to its original value. If the tests pass, control

returns to the common control program. If failures are detected, a request to switch the central controls is issued and failing tests are recorded.

Auxiliary Unit (AU) Test

3.103 The AU test routine is designed to test the registers and some of the circuits associated with central control AU communications. Only those functions that are internal to the central control are tested, and AUs are prohibited from running during the tests. The routine is entered at entry point AUTEST and does a read/write access test on all the AU associated registers. The ability to correctly select the AU register is tested as well as leakage from the buffer write bus into the registers as well as leakage from the registers into the buffer read bus. Tests are done to ensure that the AU matcher is functional. Also, tests are done to ensure that the AUs have the ability to gate data to the AU interject sources to set the correct indicator in the interject register. The operation of the AU interject inhibit flip-flops is also included in this test. If the tests pass, control returns to the common control program. If any failures are detected, an attempt is made to switch the central controls.

Stack Test

3.104 The stack test routine is designed to test the stack and its associated counter. The routine is entered at entry STCKTST and checks the software stack mechanism, the stack register, and the stack counter. The ability to generate and inhibit the D-level stack interrupt source due to a stack counter overflow or underflow is also tested. Only one test expects data to be stored into and retrieved from the call store stack. A special check is made on this test to ensure that call store 0 is really there and that data can be written into and read from it. If it cannot, then this test is skipped. If this test routine is run as part of the deferred fault recognition, the contents of the call store stack from the first address to the current stack counter is saved and restored after the test. If the test is not run as a part of the deferred fault recognition, the contents are not saved since CCFR returns to the reference point that resets the stack counter to zero.

Peripheral Loop-Around Test

3.105 The peripheral loop-around test routine is designed to check those peripheral circuits which can only be tested by sending data to the peripheral loop-around circuits in the processor peripheral interface frame. This routine uses the loop-around circuits for most of its tests and therefore the power must be on in the processor peripheral interface frame. This routine has entries from two sources. It may be entered from the common control program as are the other test routines and returns there if the tests pass or to the central control switch routine if failures are detected. It may also be entered from the Processor Configuration Recovery Program (PCRV) to configure a good peripheral bus to the processor pheripheral interface, with a pass or fail return directly to PCRV. The PU_LOOP entry is used by the common control program and CCFRPLAT is the PCRV entry.

3.106 The peripheral loop-around tests are done in three parts. The first part is a bus selection test in the active central control. In this part the test results are read from dc outputs of the peripheral bus selection circuits in the active central control. This test requires that either the coded enable or central pulse distributor (CPD) sequencer be active at the time the outputs of the selection circuits are read. The peripheral registers are initialized so that valid data will not be sent to the peripheral bus; thus, no peripheral unit is affected.

3.107 The second part of the test checks all the peripheral registers, gating paths, cable drivers, cable receivers, and buses associated with the coded enable sequencer. Data is such that no coded enable unit will respond to the peripheral operation. The pulse source polling pulses that do not change peripheral unit registers are also checked. The peripheral sequencer is started with a GCP rather than an IO order to allow full use of the maintenance mode facilities.

3.108 The third part of the test checks the gating, translation circuits, cable drivers, cable receivers, registers, and bus associated with the CPD sequencer. Also included in these tests is a check of the short binary parity circuit. The CPD sequencers are usually started with a GCP; but sometimes when the CPD is inhibited, normal CPD orders are used.

Pulse Source Failure Test

3.109 The pulse source failure test routine is called from the common control program on pulse source B-level interrupts only. The test procedure is

different from those that are used for other test routines. This routine analyzes the error indicators and data patterns and will possibly retry the failing instruction.

3.110 The test routine first reads the failing instruction (address was saved in the save current address) and determines from the operation code if it was a GCP instruction. If not, an incorrect instruction counter will be incremented. If the operation code was correct, the data field (which was saved in the save data address) is checked for the correct format. If the format is incorrect, an out-of-range flag is set for the Maintenance Restart Program (MARP). If the operation code and the instruction format are both correct, the instruction is retried with pulse source B levels inhibited. If the retry does not cause a pulse source failure, a retry passed counter is incremented. If the retry caused a pulse source failure, the central controls are switched. When the pulse source retry passed counter or the invalid instruction counter is found to be too high, an attempt is made to switch the central controls. If the central controls cannot be switched, a message is printed on the TTY that the pulse source error rate is too high but the central controls could not be switched by the program. A plant measurement counter is incremented whenever a retry is attempted, whether it is successful or not. Another plant measurement counter is incremented whenever an invalid format is detected.

3.111 The following test routines are contained in the CCFR TEST pident, which is not resident in K-code 20.

Matcher Test

3.112 The matcher test routine is designed to test all the functions of the matchers in the central controls. This routine is entered at entry point CCFRMCHK. Included is a test of the accessibility and selection of all the match registers. A leakage test from the buffer write bus into the match registers and from the match registers onto the buffer read bus is included. The matcher function is checked by loading data into the match registers and reading the dc output of the matcher for the expected match or mismatch condition. Checks are done to ensure that the central control can do all types of utility matching. If the standby is running in step with the active, tests are also made on routine matching.

3.113 This routine departs from the normal way of testing the standby central control. Normally the standby central control runs in step with the active, and matching between the central controls detects standby failures. Since the matchers themselves are being tested, this approach cannot be used. In this test routine, the active matchers are always tested. If the standby is being tested, the same tests that were run on the active are repeated on the standby. This is done by subroutines that test either the active or standby central control matchers, depending on a base address passed to the subroutine from the control portion of the test routine.

Decoder Test

3.114 The decoder test routine is designed to check the generation of the decoding and gating functions that control the instructions executed by the central control. Since no instruction in the central control will generate only one function, all or parts of many functions are tested while doing a test on a specific function. This routine is entered, from the common control program, at entry CCFRDCDT, and returns control to the common control program if the tests pass. If failures are detected, the central controls are switched if possible.

3.115 The decoder test routine uses two basic types of test procedures. The first consists of doing some initialization, executing a test instruction, and checking the results. A failure in this test procedure may suggest a failure in the initialization process, a failure in the test instruction, or a failure in the instructions used to check the results. Where the failure exists is not important to CCFR as it is designed to detect failures, not to resolve them. The second approach is to start with some piece of data and execute a series of instructions, each of which changes the data slightly. The data at the end of a successful run has a unique value. A failure in any of the instructions will cause the data to differ from the expected value. Usually it will not be possible to tell which instruction failed but again this is not important to CCFR. However, in the analysis of failing data in the decoder test, it should be remembered that the point of the detection of the failure generally will not be the point where the failure actually occurred.

Buffer Bus Access Test

3.116 The buffer bus access test routine checks the accessibility of most of the central control buffer bus registers via the buffer write bus and the

buffer read bus. The routine is entered at entry point CCFRBBAT. The test is done by dividing the address spectrum into groups, each group into subgroups, and each subgroup into its individual registers, and then checking for proper selection between each group, subgroup, and individual registers.

Interrupt Test

3.117 The interrupt test routine is designed to test the interrupt circuits. The routine is entered at entry point CCFRISQT and performs five major functions in this order:

(1) The read/write access and register selection of all the buffer bus registers in the interrupt group are checked. Those registers that directly affect interrupt generation are checked last to avoid the generation of unexpected interrupts during the test.

(2) The operation of the program-controlled interval timer is checked to ensure that it is decremented when the enable bit is set and not on reset, and the G-level source is set on the final decrement.

(3) The ability to set the H-, J-, and K-level interrupt sources and to check the inhibits associated with each source. The H and J interrupts are checked for both 5- and 10-ms operations. The ms clock is restored to its original condition at the end of the test to prevent program sanity timer timeout and a possible processor configuration change.

(4) The interrupt test routine also checks the interject mechanism that includes the proper operation of the instructions designed to check for interject work. This test checks the ability to generate all the interject sources except autonomous peripheral interjects, checks the operation of interject inhibits, and for proper program response to the setting of one or more interject sources.

(5) Finally, the operation of the interrupt circuits is checked, but only if all previous tests have passed and the modified interrupt A-level feature and K-code 20 are accessible. The test routine checks the ability of all inhibit flip-flops to inhibit interrupts, the ability to go back to normal from all interrupts, and the ability of higher level interrupts to override lower level interrupts. The final portion of this routine checks the read/write access to the registers directly involved in the generation of an interrupt. The modified interrupt A-level addresses for all unexpected interrupts are set to return to CCFR which allows CCFR to maintain control even though it generates all interrupt levels.

Fetch and Execution Sequencer Test

3.118 The fetch and execution sequencer test routine is designed to verify that the fetch and execution sequencers can make all their valid transitions. The routine is entered at entry point CCFRFEQT and if the tests pass, control returns to the common control program. If any failures are detected, an attempt is made to switch the central controls, if possible. The routine is composed of three parts. The first part is designed to test the state transitions for slow fetching (1400 ns per fetch) from program store. The second part checks the instruction execution sequencer for slow and fast operation. The last part checks the state transitions for fast fetching (700 ns) from program store. The test routine is run in the following way. First, all operations are forced to be slow and the first two sections of the test are run that check slow fetching and execution operation. Then the force slow is removed to allow the central control to run at the speed of memory, and all three sections of the test are run.

Miscellaneous Peripheral Test

3.119 The miscellaneous peripheral test routine is designed to check the remainder of the peripheral circuits that do not require data to be sent or received over the peripheral buses as a part of this test. Test results are obtained from dc readout points on the buffer bus, the contents of the buffer bus registers, or from the interrupt sources. Like the peripheral loop-around test some sections are devoted to coded enable tests and some to CPD test. The coded enable circuits tested in this routine are the coded enable parity generation circuits, the peripheral answer parity check circuits, the coded enable matchers (PM0 and PM1), the proper setting of the coded enable error sources, and the interaction between D- and E-level interrupts and the coded enable sequencer. The CPD circuits checked in this routine are the interaction between the D- and E-level interrupts and the CPD sequencer, the CPD echo detector circuit, the CPD reply matcher, and the setting of the CPD peripheral error sources. The routine is entered at entry CCFRMPUT from the common control pro-

gram and returns there if the tests pass. If errors are detected, the central controls are switched if possible.

C. Service Routines

General

3.120 Service routines are routines used by CCFR and other programs to actually change or to request a change in the configuration of the central controls. These routines usually return program control to the calling (client) program when the requested function is completed. However, because of their nature, some functions will result in a transfer of control to another program or will result in the generation of an interrupt.

Stop (Remove) the Standby Central Control Routine

3.121 This routine does the two basic functions of (1) removing the standby central control from service and (2) initializing the central control complex for simplex operation. This routine has three different entry points. The first point is used only by CCFR and is used to remove the standby central control from service and to set a diagnostic request bit in the status word. The actual diagnostic request is made later in the program by another CCFR routine. The second entry (CCFRSTBL) is a global entry and is used by other programs to remove the standby central control and to request that it be diagnosed. The actual diagnostic request is made within this routine. The third entry (CCFRREM) is also a global entry and is used by CCFR and other programs to remove the standby central control without requesting a diagnostic. All three entry points, after doing some initially different tasks, join to form a common program.

3.122 To stop the standby central control without interfering with the rest of the system operation and without causing an interrupt, the remove routine does the following sequence of events. Call store D levels and program store E levels are inhibited to prevent a transient generated by stopping the standby central control. The AUB transmission may be inhibited for about 20 central control cycles and then restored without affecting AU jobs. This routine sets all AU inhibits, clears all activity flip-flops in the standby central control, stops matching, divorces (inhibits cross-coupled error and control signals from adversely affecting the operation of the simplex central control) the central controls, stops the standby central control, sets the bus control flip-flops CBO, CBT, PBO, and PBT, and restarts the AUs by restoring the original inhibits.

3.123 This sequence of events stops the standby central control, isolates it from the rest of the system, and establishes a memory bus configuration compatible with simplex central control operation. The status words for the call store and program store bus configuration are updated according to the hardware status.

3.124 This routine also establishes a peripheral configuration for both the CPD bus system and the coded enable bus system. For the CPD bus system, the only initialization required is to set the appropriate control flip-flop; no status word update is required. The coded enable bus system can communicate over a split bus system; ie, central control 0 over bus 0 and central control 1 over bus 1 or vice versa. If neither bus is marked out of service in the status words, then bus control flip-flops PUBO and PUBR are set and PUBT is reset. If either bus is marked out of service, PUBO and PUBT are set and PUBR is reset. The status word is then updated to the hardware status. This procedure provides a peripheral unit bus configuration that is compatible with a simplex central control and either a simplex or duplex peripheral unit bus.

3.125 The matcher status table is updated to show that no routine matching is in progress. Any possible call store or program store errors are cleared and inhibits are restored to their previous setting. A lamp status update is also requested. The processor configuration sanity timer is recycled if the routine is called as a part of the processor configuration recovery.

3.126 A routine is then called to set up and verify the current match mode. This ensures the consistency of the match status table and sets up the match control registers in the active central control. The match control registers in the standby central control are then set to a no matching condition.

3.127 The remove routine then ends by returning control to the calling program or routine.

Restore the Standby Central Control to Service Routine

3.128 The restore routine is composed of a subroutine within a subroutine. The outer routine is

normally used to restore duplex central control operation for system use. The outer routine establishes routine matching in at least one central control and does some checks to ensure that the central controls will remain in step. The inner routine (entry point CCFRSTPQ) does the actual operations that put the central controls in step. The inner routine is used by CCFR and other maintenance programs to do tests on the central controls or other subsystems with duplicate central controls. In addition, it is used by the outer routine to restore duplex central controls operation for system use.

3.129 The inner routine uses pulse points to stop and start the standby central control clock, clear stop the standby central control, clear the standby central control, and reset any auxiliary activity flip-flops that might be set in the standby. This should leave the standby central control in a state that allows the active central control to have read/write access to internal standby central control locations. The interrupt inhibit flip-flops of the active central control are blindly copied into the standby if adequate access exists.

3.130 Before going any further, it is necessary to check the read/write access to the standby central control. Two data patterns are written into standby central control registers and read back for verification. If either data pattern is read back incorrectly, the routine is ended and a fail return is made to the calling routine.

3.131 Before attempting to start the standby, several internal buffer bus locations must be initialized. Some are initialized by copying the corresponding active register into the standby register, some are simply zeroed, and some are written with a predetermined data pattern. This initialization puts the standby in a listen-only mode on the call store, program store, peripheral and AUB systems. All addressing is done by the active central control. Also, all AU activity is stopped by latching the AU sequencer in both central controls. Since precise timing is required to get the central controls in step, no AU interference can be tolerated.

3.132 The instruction fetch sequencer is then set to the proper start-up state. The instruction stack is loaded with a FILL instruction followed by a long transfer instruction to a fixed address. The active central control then executes a GCP instruction to start the standby central control. At the beginning of the second cycle of the GCP instruction, the standby's clock starts and the standby executes the FILL instruction. At the end of the GCP instruction in the active central control and the FILL instruction in the standby, both central controls begin the execution of the long transfer instruction. This brings the central controls into step with the active fetching instructions and data for both central controls.

3.133 Verifying that the active and standby are in step is done by having each central control write a specified data pattern into a buffer bus location. The active central control then reads the test location from the standby central control and checks it for correct data. If it is not correct, then the standby central control did not stay in step with the active and the restore routine returns control to the calling routine with a failure indication.

3.134 If, however, the central controls are running in step, it is necessary to initialize all registers and buffer bus locations that were not initialized before starting the standby central control so that they are the same in both central controls. During this process, registers are initialized in groups according to their relationship to each other. After all registers have been initialized, control is returned to the calling routine.

3.135 The outer routine is used to completely restore the standby central control for system use. This routine is always run as a part of the deferred CCFR and as an MACP job. This means that the standby central control is always restored to service on base level and never on interrupt level. On entry into this routine, the status word is checked to determine whether to abort the restore. If the standby was removed because of excessive C-level interrupts, then only an unconditional restore from the TTY is accepted by this routine. Also, if the standby was removed via the TTY, it must be restored via the TTY.

3.136 If the status is all right, the standby is removed from service. The removal is for initialization purposes and to provide a stable starting point for the restore procedure. The inner routine is then called to put the central controls actually in step. If the central controls are put in step, control is returned to the outer routine to set up routine matching without interrupts. A maze test is then run to determine if the central controls will stay in step. The

maze is composed of miscellaneous instructions with different options and many conditional transfer instructions. Some of the conditional transfers are expected to be executed; others are not. There is only one valid path through the maze test, and any departure from the path will result in a mismatch that will freeze the matchers and stop the standby central control. If either the maze test fails or if the inner routine is unable to start the standby in step, the restore routine prints a message that the central controls will not run in step and returns to deferred CCFR to end the job request.

3.137 If the central controls pass the maze test, the central control status word is updated to show that the central controls are running in step. This includes a lamp update request to update both the central control frame and the master control console (MCC) lamps. The matcher status table is updated to show routine matching in the standby central control; and, if no utility matching is in progress, routine matching is also indicated for the active central control. The divorce and standby trouble flip-flops are both reset. A routine is called to update the matchers to the match status table and to enable C-level interrupts. If the restore request was not part of the routine exercise, a message indicating that the standby has been returned to service is printed on the TTY. The restore then returns to deferred CCFR to do cleanup tasks and end the request.

Switch the Central Controls Routine

3.138 The switch routine does either a conditional or an unconditional switch of the active and standby central controls by using the pulse point switch mechanism. This routine has five entry points, three of which are used exclusively by CCFR. The remaining two entry points are used by CCFR as well as other maintenance programs.

3.139 The only difference between the three exclusive entry points is whether the B, L, and K registers have been stored in the active fail bin and whether the failing test bit has been set. Program control is transferred to one of these three entries only if a CCFR test routine detects a failure in the active central control.

3.140 All three entries merge into a common routine after the initial differences have been resolved. The common routine stores all the remaining active central control registers (in addition to B, L, and K) that are required for the active central control fail TTY message. Indicators are then set for error analysis to show the starting address and the number of words that have been saved. A code word is then initialized to indicate to CCFR that an active central control failure has been detected.

3.141 If the switch routine was entered because of a processor configuration test failure, it is possible that the active central control does not have enough sanity to do a pulse point switch of central controls. A 512-cycle delay is built into the switch routine. If the switch routine is entered because of a processor configuration test failure, the processor configuration sanity timer (a 496-cycle timer) will time out before the delay is completed and cause a B-level interrupt. The switch is then done by the processor configuration sequencer circuits rather than CCFR. If program control passes beyond the delay, then additional checks are made to determine whether to do the switch.

3.142 The program checks to see if the failure occurred during a deferred CCFR test segment. If it did, no attempt is made to switch the central controls and program control is returned to the deferred CCFR. Also, the program checks the central control trouble flip-flop. If the trouble flip-flop is set, no attempt is made to do the switch and control is returned to the calling routine. On the other hand if the flip-flop is not set, the switch procedure continues.

3.143 The fourth entry (CCFRRACC) is a conditional switch entry that is used by CCFR and other programs to switch central controls if the trouble flip-flop is reset. If the flip-flop is set, control is returned to the calling routine. Otherwise, this entry merges with the other three entries into one common switch procedure.

3.144 The final entry point (CCFRSWCC) is an unconditional switch entry that is used by CCFR as well as other maintenance programs. This entry, after resetting the trouble flip-flop, merges with the other entries into the common switch procedure.

3.145 Once the final decision has been made to switch, the common switch procedure configures the system to run with only one central control and clear stops the standby central control since the central controls cannot be switched while running in step. An attempt to switch in step would

result in a hardware race condition which would make each central control partially active and would trigger a processor configuration.

3.146 The J-, K-, and interrupt-level activity registers of the active are copied into the standby, and the analog timer in the standby is recycled to prevent a processor configuration on the switch. The central controls are switched by a GCP instruction that should cause a B-level interrupt during the second cycle of the GCP. A 3-cycle delay is provided in the program after the GCP to allow time for the switch and the B-level interrupt to occur. If the switch fails and the B-level interrupt does not occur, the program will continue after the 3-cycle delay and transfer to a routine in PCRV to cause a deliberate activation of the processor configuration circuit since the active central control is faulty and a system reconfiguration is necessary.

Match Mode Administration

3.147 The match mode administration routine is responsible for checking the validity of the matcher status table, making changes to the matcher status table, and initializing the match control registers in the central controls according to the matcher status table. There are two entry points to this routine, CCFRSVMM and CCFRMREQ. The first verifies the matcher status table and initializes the match control registers according to the status table. The second entry is used to change a portion or all the matcher status table and establish a new match mode. The new match data is contained in a buffer table in exactly the same format as the matcher status table. This data is checked while in the buffer and replaces the matcher status table data only if it passes all tests.

3.148 Match mode administration contains two routines in addition to the administration routine. The first routine is used to check the validity of a utility match request and may be called to check the current contents of the matcher status table or the buffer table. A return address is supplied by the using program. The second routine is called when both routines and utility matching are specified at the same time, and it ensures that no conflicts occur between the two modes.

Request Execution of Deferred CCFR

3.149 This routine is used by other interrupt programs to enter a request for deferred CCFR in MACP's job request tables. It is usually called after an interrupt that stopped the standby central control to restore the standby central control to its previous configuration. The routine is entered at entry point CCFRRQDF. If the request is granted, all tests are run on both central controls and the standby is restored to service. The request is not granted unless the central controls were previously running in step. In addition, a request is always made to normalize the call store, program store, and peripheral system.

Set Up Routine Matching Without Interrupts

3.150 This routine is used by other maintenance programs while performing tests with duplicate central controls. The routine is entered at entry point CCFRRTMA and requires a return address to the user. This routine completely initializes the match registers in both central controls for matching with options, halt match and freeze stop standby central control on mismatch. The C-level enable bit is reset to prevent interrupts while running in this mode. This routine is not used for normal system operation.

Record Standby Error Data

3.151 This routine is used to store data for the standby fail print message whenever the standby central control fails a test. This routine saves the contents of all match registers, the standby central control operational registers, the standby current address register and data address register, and a failing test word. The failing test word shows which test was being run when the failure occurred. This routine is entered only from CCFR and a return address is required; its entry point is CCFRRSYD.

4. CALL STORE FAULT RECOVERY PROGRAM—CSFR

INTRODUCTION

4.01 The CSFR program has four basic functions in the 1A Processor. First, CSFR is normally entered from the System Interrupt Recovery Program (SIRE) after a D-level interrupt; later, the program acts as a filter for the several sources of D-level interrupts. The CSFR program steers noncall store related problems to the appropriate recovery program. Second, CSFR resolves call store related problems. Third, CSFR provides a "bootstrap" capability for the call store community. Finally, CSFR provides several service routines for use by other programs.

CALL STORE ORGANIZATION AND FEATURES

4.02 The 1A Processor call store community is made up of several individual call stores. The number of call stores varies according to the type and size of the switching office installation.

4.03 The store community may consist of 64K (1K= 1024 words) core-type stores (two per frame), 64K semiconductor-type stores (up to six per frame), or 256K semiconductor-type stores (up to eight call stores and six program stores per frame) or a combination of these frames. All three frame types can operate at 1400 ns (slow); in addition, the 256K store can operate at 700 ns (fast) when not mixed with other types of stores in the same community. The maximum number of call stores is 46 members (0-45) of the 64K size or its equivalent in 256K size.

4.04 The central controls access the call stores via duplicated call store buses that interconnect every call store frame with both central controls (Fig. 5). (The buses may have as many as four branches.)

4.05 Each call store word location is identified by a unique address (Fig. 6). This address consists of a K-code and a data location address. The K-code portion of the address identifies the specific 64K

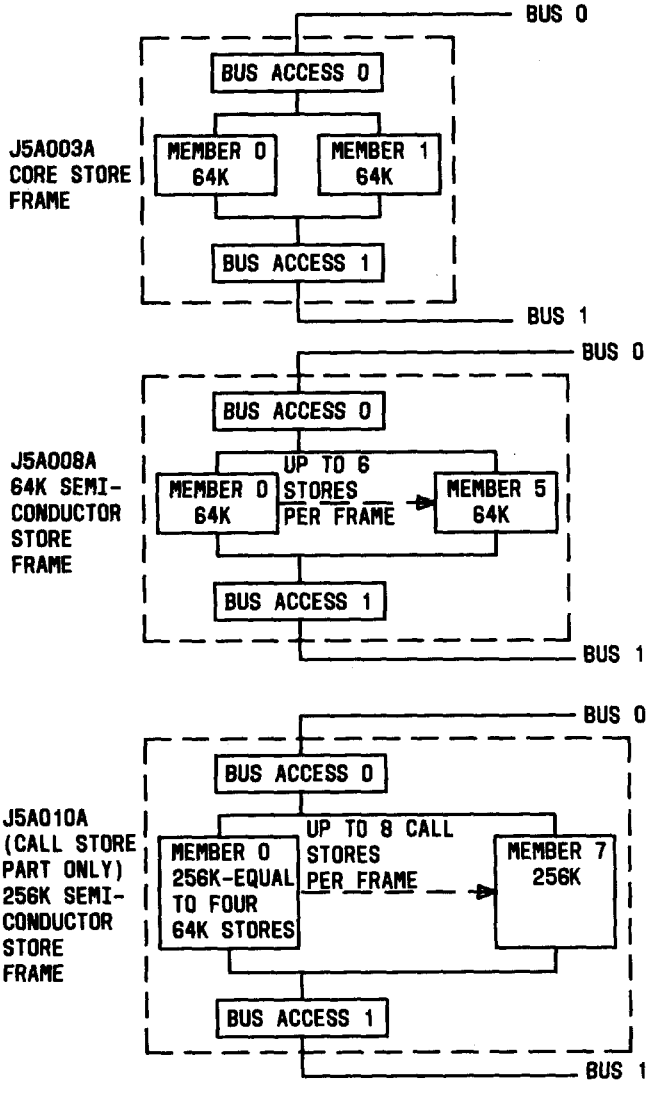


Fig. 5—Functional Layout of Three Frame Types Used for Call Store

call store memory block to be addressed. The K-code also identifies the specific call store to be addressed (ie, one K-code identifies a 64K call store, but any one of four consecutive K-codes identifies a single 256K store). The data location identifies the specific data location to be accessed within the 64K call store memory block. Call stores that contain duplicate copies of information located in other call stores are assigned the same K-codes. The K-code of any call store may be changed under program control by CSFR. The changeable K-codes allow CSFR to substitute one of the duplicated call stores for another in the event of a failure (four 64K stores may be used to replace a single 256K store). The CSFR program can uniquely identify any member of the call store community regardless of its K-code by pulse point access using GCP instructions.

> ***Note:*** When 64K and 256K stores (1400 ns only) are mixed within a call store community, several combinations are possible for duplicate store arrangements. Duplicate stores may be a pair of 64K stores, a pair of 256K stores, or a 256K store and four 64K stores. The fault recovery programs support all possible combinations of stores except only 256K stores are supported when the 1A Processor has the Attached Processor System (APS) feature.

4.06 Bus routing of communications between central control and the call stores is controlled by bus selection flip-flops located in central control and in the call stores. The duplicated buses are referred to as 0 and 1. Always, one pair is called the active bus and the other is called the standby bus. Central control bus selection flip-flop functions (Table A) are as follows:

(a) CBA—When CBA is reset, bus 0 is the active bus and bus 1 is the standby. This results in the active central control transmitting and receiving over bus 0 and the standby central control using bus 1. When CBA is set, the active central control uses bus 1 and the standby uses bus 0.

(b) CBO—When CBO is set, the active central control transmits on both the active and the standby buses, but receives on only the active bus. The standby central control does not transmit data, but does receive on the standby bus.

(c) CBT—When CBT is set, the active central control sends and receives on the active bus and the standby central control receives on the active bus, but does not send on either bus.

Table A summarizes the configuration capabilities of the bus from the central control end of the bus.

4.07 The controls located in the call stores are as follows:

(a) RO—Selects the bus over which the store receives information from central control. When reset, the store receives data on bus 0. When set, the store receives on bus 1.

(b) ANS0—When set, the store sends data to central control over bus 0. When reset, the store does not send on bus 0.

(c) ANS1—When set, the store sends data to central control over bus 1. When reset, the store does not send on bus 1.

When the store is in a maintenance or control mode, the store returns data to central control on the bus designated by RO regardless of the state of ANS0 and ANS1. Table B summarizes the controls located in call store.

4.08 The various combinations of settings of these controls allow the program to isolate most bus faults with a minimal effect on the system operation.

4.09 The 256K store has two additional access controls that are used by the maintenance and recovery programs. The first control is the SLOW flip-flop. When set, the store operates at a 1400-ns rate. The store must operate at this rate when mixed with 64K stores in the same community. When the slow flip-flop is not set, the store operates at the 700-ns rate. The second control is a set of four flip-flops: OKBR, 1KBR, 2KBR, and 3KBR. Each of these flip-flops controls the read access to the corresponding 64K block of memory. When any of these are set, the corresponding 64K block of memory will not respond to a read access and will not return error signals for either reads or writes. These four flip-flops are used during copy operations to update a store in 64K increments.

4.10 Bus access to the call stores may be done in any one of three modes of operation: normal, maintenance, or control mode. The mode of operation depends on the setting of the maintenance flip-flop

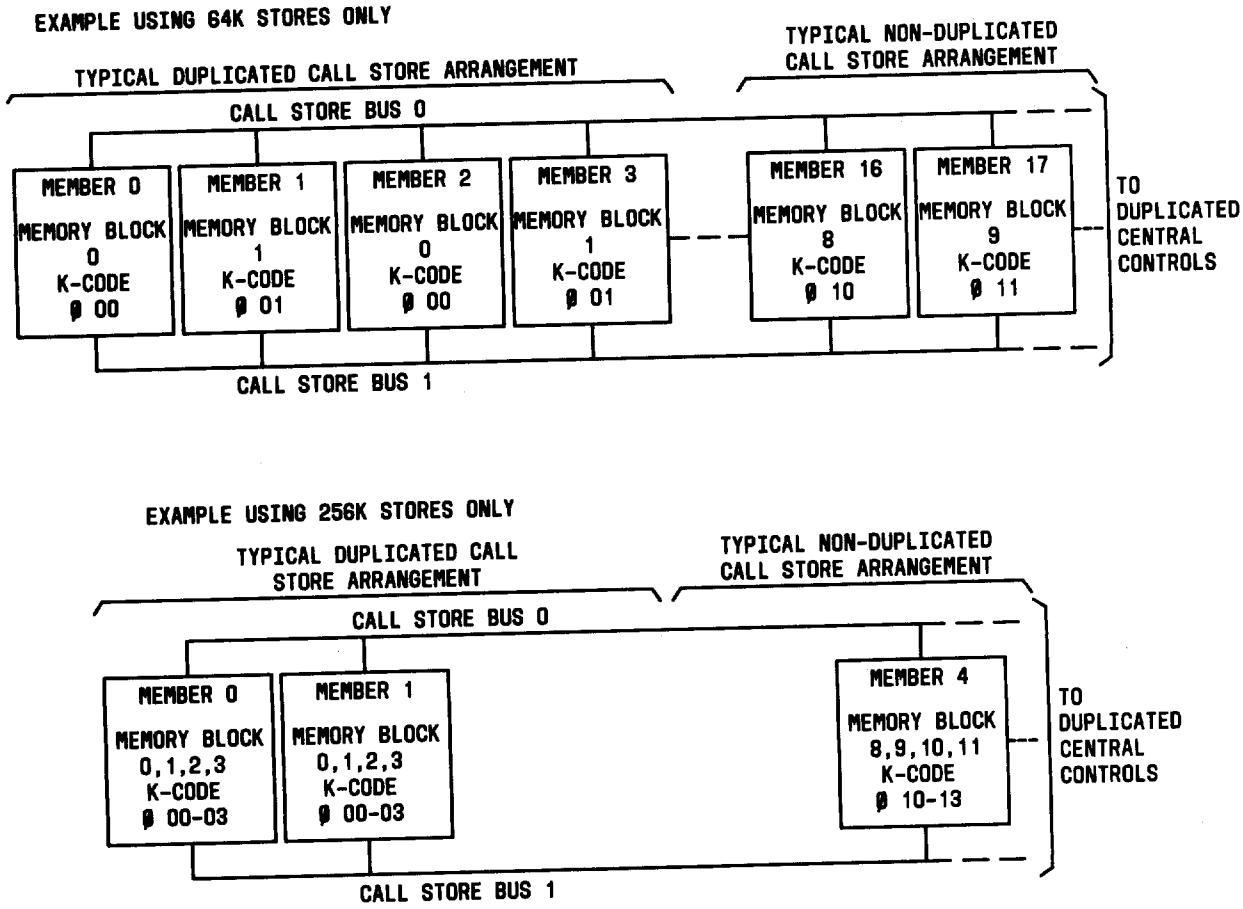


Fig. 6—Call Store Organization—Simplified

in the call store and mode bits which accompany the address information for each access of the store.

4.11 Normal mode operation allows central control to access those call stores whose K-code matches the one sent via the bus, provided the maintenance flip-flop is not set in the store. Maintenance mode operation allows central control to access a store whose K-code matches the one sent, provided the maintenance flip-flop is set. Control mode operation enables central control to alter the status of control and maintenance flip-flops within a store without affecting the data stored in the memory module. Control mode operation depends on the mode bits matching the setting of the maintenance flip-flop (set or reset).

TABLE A

CALL STORE BUS CONTROLS LOCATED IN CENTRAL CONTROL

CONTROL FLIP-FLOPS (NOTE 1)			CENTRAL CONTROL		
CBT	CBO	CBA	SENDS ON CALL STORE BUS (NOTE 2)	RECEIVES ON CALL STORE BUS	
0	0	0	0	0	
0	0	1	1	1	
0	1	0	0 & 1	0	Active
0	1	1	0 & 1	1	Central
1	0	0	0	0	Control
1	0	1	1	1	
1	1	0	0 & 1	0	
1	1	1	0 & 1	1	
0	0	0	1	1	
0	0	1	0	0	
0	1	0	X	1	Standby
0	1	1	X	0	Central
1	0	0	X	0	Control
1	0	1	X	1	
1	1	0	X	0	
1	1	1	X	1	

Note 1: 0 = Reset, 1 = Set

Note 2: X = No Bus Transmission

4.12 In the control mode of operation, the store recognizes maintenance load and maintenance store orders. These orders provide access to maintenance, status, and control flip-flops located within the call store.

4.13 Memory within the call store community provides storage for data required by central control during program execution. (In addition to data, call store also stores the program store fault recovery program.) Some of the data, such as main program constants, data tables, and translation information, is also stored on disk. On the other hand, data of a transient or temporary nature relating to call progress or equipment status and subject to change by the program resides only in the call store community (call store status is kept in program store).

4.14 The call store community consists of both duplicated and nonduplicated memory. Information that is not backed up by a file store copy, or information that must be immediately accessible if a call store fails, is duplicated within the call store community. Full duplication is used for this type of data so that in normal operation each call store in duplicated memory contains a complete copy of data stored in a mate or duplicate call store. The duplicate

TABLE B

CALL STORE BUS CONTROLS LOCATED IN CALL STORE

CONTROL FLIP-FLOPS (NOTE 1)			CALL STORE	
RO (NOTE 2)	ANS0	ANS1	SENDS ON CALL STORE BUS (NOTE 3)	RECEIVES ON CALL STORE BUS
0	0	0	X	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0 & 1	0
1	0	0	X	1
1	0	1	1	1
1	1	0	0	1
1	1	1	0 & 1	1

Note 1: 0 = Reset, 1 = Set

Note 2: On maintenance and control orders, the call store answers the bus designated by the RO flip-flops regardless of the state of the ANS0 and ANS1 flip-flops.

Note 3: X = No Bus Transmission

stores are assigned the same K-code (or K-codes if the store is 256K). A 256K call store may have four 64K stores as a mate.

4.15 Information in the call store community that is also contained in file store is ordinarily not duplicated in call store. Duplicated memory covers the lower part of the call store address range and nonduplicated memory the upper call store address range. Stores within duplicated memory are used as functional replacements in case another member of the call store community (duplicated or nonduplicated) becomes disabled. If a duplicate call store is disabled, another call store would not be used to substitute for it unless its mate store is also disabled.

4.16 In a normal 1A Processor configuration with both central control and call store buses in service, duplicate stores are configured to opposite buses; ie, one store of the pair receives from and sends to the active call store bus while the other store of the pair receives from and sends to the standby call store bus. The nonduplicated stores are configured to receive from the active bus and send to both buses.

4.17 When one call store bus is out of service, one store of each pair of duplicated stores is configured to receive from and send to the active call store bus. The other stores of the duplicated pairs are configured to receive from the active bus but are not configured to send on either bus. When one central control is out of service, one store receives and sends on the active bus and the other store receives and sends on the standby bus.

4.18 The primary mechanisms for trouble detection in the call store and related equipment are as follows:

(a) The call store returns an all-seems-well (ASW) signal to central control for every read or write operation that appears valid at the store. Failure to return ASW indicates a trouble condition.

Note: The 256K stores operating fast (700 ns) cannot detect error conditions in time to return an all-seems-well failure (ASWF) during the 700-ns cycle. For those error conditions that normally return ASWF during slow (1400-ns) operation, the store returns the data parity or write enable failure signal (DPWEF) during the next 700-ns cycle instead.

(b) The DPWEF signal is used only by 256K stores operating fast. This signal is returned during the next 700-ns cycle after the error is detected.

(c) The call store returns an (ASWF) signal on every read or write operation that appears invalid at the store.

(d) Each address received by a call store is accompanied by a parity bit covering the address (data word location) and the K-code. The call store checks the parity. Incorrect parity causes the store to read at the indicated address (no writing is done), and to suppress the ASW signal while generating the ASWF signal.

(e) All data written into call store is accompanied by two parity bits for each 24 bits of data. These parity bits that cover the address (including K-code) and data are checked by the store itself, and are stored with the data. If the parity is incorrect, the data is still written in the store, but ASW is suppressed and ASWF is sent to central control. The parity bits are transmitted back to central control whenever the data is read out. The central control then checks the received parity against the computed parity over data received and the address.

(f) Semiconductor stores autonomously perform a periodic data parity check of every address. If this autonomous data parity check fails, the store sets a flag flip-flop and will return ASWF and no ASW the next time it is addressed.

(g) All data to be written into call store must be preceded by a write enable pulse timed to precede the readout strobe. The write enable prepares the store to receive information from the data bus. If the pulse is not received on a write order, the cycle is completed as a read operation and an ASWF signal is generated while ASW is inhibited.

(h) Each store contains an error summary register. Setting an internal error indicator in the error summary register sets the maintenance flip-flop, enables the ASWF signal, and inhibits ASW. Errors which occur after time for transmission of ASWF and ASW are detected by central control on the next operation to the store. Internal errors are recorded in the error summary register when conditions such as access circuit trouble are detected. Also, certain access circuit points are monitored and checked for validity. These error indicators can be used by CSFR to localize the fault or error.

(i) Several types of program errors can be detected by central control. These errors include protected address range write violation, underflow/overflow of the program stack counter, write protect error (256K store only), and transfers to an address outside the program store range without the call store program flip-flop set.

Detection of any of the above error conditions will cause a D-level interrupt by the central control.

4.19 For a more detailed description of call store, refer to the following sections:

SECTION	TITLE
254-201-010	Call Store/Program Store—Description
254-201-011	Call Store/Program Store—Theory
254-201-012	Call Store/Program Store, 1400-ns Semiconductor Store—J5A008A—Description
254-201-013	Call Store/Program Store, 1400-ns Semiconductor Store—J5A008A—Theory
254-201-014	Call Store/Program Store, 256K Semiconductor Store—J5A010A—Description
254-201-015	Call Store/Program Store, 256K Semiconductor Store—J5A010A—Theory.

CSFR—FUNCTIONS AND STRATEGY

A. General

4.20 The primary purpose of CSFR is to return the system to normal call processing (Fig. 7) as quickly as possible after a fault or error has been detected in the call store community. Therefore, whenever possible, CSFR will remove the faulty store from service on a "first-look" basis. The first-look approach uses error indicators in the central control, the bus configuration, and the store status to identify the faulty store.

4.21 The first-look approach replaces the faulty unit with a duplicate if one is available. If a duplicate is not available, the recovery selects a store (from the duplicated call stores) and initiates a copy of the suspected unit into the selected unit. If tests are unable to detect trouble within the faulty store, it is restored to service. When the update is completed, the updated store is placed into service and the suspect store is removed and diagnosed. If the faulty unit fails again before the update is completed, the faulty unit is removed from service and the system must wait for the successful completion of the substitute store's update from a backup copy stored on file store.

4.22 However, when the failing store is not duplicated and a store is not available for selection as a substitute, the first-look approach is not used. Call store fault recovery uses the error history of the failing store to determine the action to be attempted. If the store's error history is acceptable, call store fault recovery corrects the failing word, does a complete access test of the store, and restores it to service.

4.23 An unacceptable error history or a failure of the complete access test can cause call store fault recovery to do a bootstrap of the call store com-

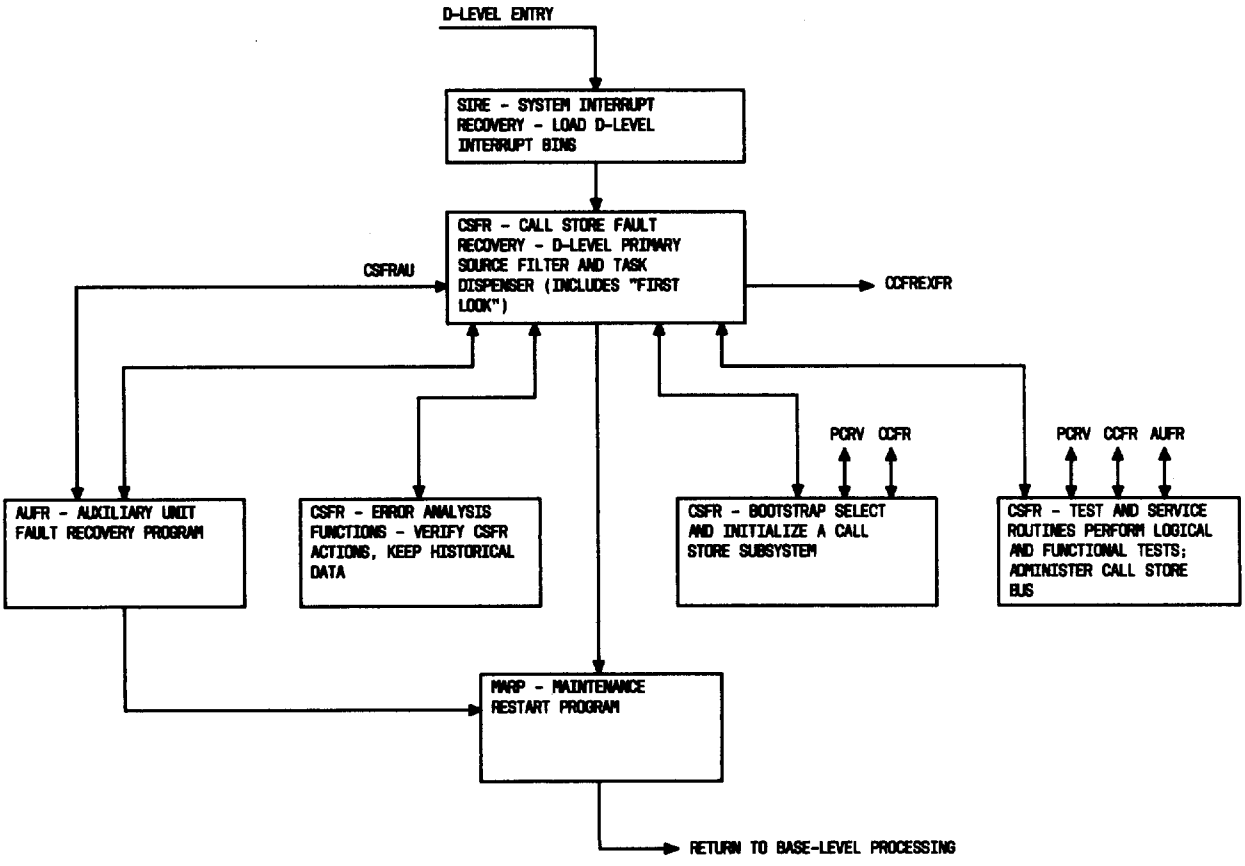


Fig. 7—Call Store Fault Recovery—Program Flow and Interfaces—Simplified

munity. The bootstrap routine attempts to assemble a complete copy of the call store information by using only stores that pass the bootstrap tests. If necessary, call stores will be used regardless of their status before the failure. Should the bootstrap fail to establish a valid copy of call store, a program transfer is made to the Processor Configuration Recovery Program (PCRV) to switch central controls (B-level interrupt). After a successful fault recovery, control is returned to normal processing by transferring to the Maintenance Restart Program (MARP).

4.24 The call store service routines and bootstrap routines may also be entered from the CCFR, AUFR, and the PCRV programs. These programs may use the service routines to verify access of call stores. In addition, CCFR or PCRV may enter the call store bootstrap routine if either determines that one or more call store memory blocks were not error-free or not provided in the configuration that has been established. Therefore, call store bootstrap is entered to recover a valid call store configuration. Control is then returned to the calling program, either PCRV or CCFR, which judges the success of the bootstrap.

B. Interrupt Recovery

4.25 Following a D-level interrupt, the interrupt sequencers transfer program control to the System Interrupt Recovery Program (SIRE) which stores the appropriate data in the D-level interrupt bins. The SIRE program then transfers control to CSFR which then performs a D-level source filtering function.

4.26 The filter takes one of the following actions:

(a) For software type sources (out-of-range address error, protected address violation, underflow/overflow of stack counter, and program transfers to call store without setting call store program flip-flop , the filter sets the appropriate flags and transfers program control to MARP. Write protect failures only apply to the 256K store. Write protect failures are treated as access failures, [paragraph 4.26(c)], but are transferred to the Write Protect Administration Program (WPAD) before exiting fault recovery via MARP. For software type restarts, MARP either rolls back to a safe point in the interrupted program or returns control to the base-level reference point (stack counter errors).

(b) For AU access failures, the filter transfers program control to the Auxiliary Unit Fault Recovery Program (AUFR).

(c) For call store access failures, the filter transfers program control to the recovery routines within CSFR.

(d) Finally, if no sources are found by the filter, program control is transferred to the Central Control Fault Recovery Program (CCFR) to verify the central control.

4.27 If the D-level interrupt was caused by a call store access failure, further action is required by CSFR.

> ***Note:*** The AUFR program may encounter access problems with the call store community. These problems are referred to CSFR at entry point CSFRAU. These problems may include a write protect failure. Recovery from this type failure is essentially the same as for a D-level call store access failure. However, for a successful recovery, control is returned to AUFR rather than exiting to MARP.

4.28 First, CSFR gets the failing call store address and verifies its validity. The CSFR program determines whether the addressed module is actually equipped. If the address is invalid, the problem is treated as an out-of-range program error. Appropriate flags are set and the program exits to MARP. Also, if the failing address is not available, CSFR performs an access test on the entire call store community to verify its integrity.

4.29 However, if the failing address is a valid call store address, CSFR continues the recovery process by testing the call store bus. The program tests the bus by sending several patterns of test data to the call stores to be returned to central control and verified. (The test data is not written into memory, but is simply looped back to central control via the reply portion of the bus.) If the bus test fails, an attempt is made to remove the failing bus from service.

4.30 If the failing bus is the only bus in service, CSFR will not remove the bus but will transfer to the call store bootstrap routine. The bootstrap routine will then attempt to assemble a valid call store community.

4.31 On the other hand, if the calling bus can be removed from service, CSFR reconfigures the bus control flip-flops in central control (Table A) as well as those in all call stores (Table B) to the new

configuration. A full access test of the call store community is then done to assure that central control can access a complete copy of all call stores.

4.32 If the bus test passes, CSFR determines whether the failure occurred in a duplicated call store. If the failing store is duplicated, it is removed from service (unless the mate is out of service or if its error count is greater than zero but less than five) and its mate is set to operate as a nonduplicated store. The store error count is incremented by one on each interrupt that is caused by the store. The error count is decremented by one every half-hour. After removal of the suspect call store, the access test is performed on the call store community. If the store error count is greater than zero but less than five, CSFR will attempt to keep the suspect store in service as the standby unit. The CSFR program performs a gross access test on the suspect store to qualify it for service. Successful completion of the access leads to restart.

4.33 If both copies of a duplicated block of call store are not available or if the suspect store contains one of the nonduplicated blocks of memory, CSFR searches for a duplicate call store which can be used as a replacement for the suspect store. Before searching for a substitute, CSFR checks to determine if, during a previous interrupt, a substitute was found for this block of memory and is in the process of being updated. If an update is in progress, CSFR goes to a routine (FINFILL) to finish updating the substitute store on interrupt. Here, the system must wait until the update is complete before call processing can be resumed via MARP.

4.34 If no duplicate store is available for use as a replacement, the error history of the suspect store is checked. An acceptable error rate causes CSFR to attempt to use the suspect store. Successful completion of the access test for the suspect store, including a retest of the failing location, is followed by an access test of the entire call store community. Unless a failure is encountered, CSFR exits to MARP. An unacceptable error rate would cause CSFR to attempt to bootstrap the call store community.

4.35 If the search for a call store available for use as a substitute is successful, then CSFR will attempt to use the suspect call store while the substitute store is being updated. The CSFR program performs a gross access test on the suspect store to qualify it for use, then takes the following actions:

(1) The maintenance flip-flop in the suspect store is reset (if it was set by an internal error) and the RO flip-flop is set to the active bus.

(2) The selected substitute store is set to receive from the active bus, but not to transmit. Its K-code flip-flops are set to the same K-code as the suspect store. The maintenance flip-flop of the substitute store is reset, but its out-of-service lamp is lighted until its update is complete and it is brought into service.

(3) The update of the substitute store is attempted by copying from the suspect store (not from file store). The copy is done as an MACP job on base level after call processing has resumed.

(4) If the substitute store is the last available for the community, the major alarm is sounded.

(5) The error history for the store and the community is updated.

(6) The call store status table is altered to reflect the present situation and the "error analysis in progress" flag is set.

4.36 If the substitute's data is copied from the suspect store and no errors are detected during the update, the suspect store is removed from service and diagnosed. If it passes diagnosis, it is put back into service and the substitute is returned to its normal status within the call store community.

4.37 Before program control is returned to call processing, the access test is done to verify that a complete copy of call store is available. Detection of an error during the access test leads to further evaluation of the problem. Various actions can be taken to correct failures:

(a) The standby store is removed if the failing K-code is duplicated and the failure was detected on the standby bus.

(b) The standby bus is removed if the failing K-code is unduplicated and the failure was detected on the standby bus.

(c) A parity correction is done if a Trap Refresh Data Parity Failure is detected. (Maximum three times).

(d) A switch of the buses and a removal of the standby bus is done if the maximum parity corrections were done (see step 3 above).

(e) Bootstrap is entered if the access test failed too often (six times) or if a standby bus problem is detected but no standby bus or store exists.

After any one action the community access test is again entered.

4.38 If the second failure is determined to be from the same store as caused the original failure, a check for a store in update is made and the update completed. Otherwise bootstrap is entered.

C. Bootstrap Functions

4.39 A bootstrap of the call store can be undertaken for any of the reasons that have been described to this point. In addition, PCRV or CCFR may enter the bootstrap routines. The occurrence of a call store bootstrap and its reason is reported via the master control console (MCC) and, if possible, also via the TTY.

4.40 On entry into the bootstrap routines, a check is made to determine whether bootstrap has been passed an excessive number of times. If bootstrap has passed excessively, then the fundamental strategy is varied. The fundamental strategy is described first.

4.41 The call store bootstrap routines attempt to assemble a complete copy of the call store memory blocks. Entry into bootstrap implies that there is a major problem in the call store community since the first-look routines were unable to come up with a valid configuration of stores. The bootstrap routine starts by selecting one call store bus (the active) over which the bootstrap is to be tried.

4.42 All maintenance flip-flops in the call store community are set and the RO flip-flops are set to the standby bus. The bootstrap routines then attempt to bring the individual call stores into the system one at a time.

4.43 The maintenance flip-flop for the store under test is pulsed to toggle the RO flip-flop to the active bus. The bootstrap tests are then done using the active bus. Maintenance and control tests are done with the maintenance flip-flop set. The maintenance flip-flop is reset, and the bootstrap tests are completed with the store in the control and normal modes. At the conclusion of the tests, the maintenance flip-flop is set. The status in program store is then updated to reflect the bootstrap results, and tests are started on the next call store.

4.44 When all stores have been tested, the bootstrap results are evaluated by analyzing the status table. If a complete copy of call store is available, the bootstrap was successful. When a complete copy is not available, the ability of the normally duplicated call stores to complete the copy is checked. Substitutes are flagged by the program to be updated before the return to call processing.

4.45 When a complete copy cannot be assembled even with the normally duplicated stores acting as substitutes, the bootstrap has failed via the active bus. The active and standby buses are switched and the bootstrap is attempted again on the new active bus.

4.46 If the bootstrap fails on both buses, a check is made to determine whether the bootstrap was requested on D-level by CSFR or by another program. Control is returned to the requesting program with a failure indication if the entry is not a D-level request. D-level bootstrap requests that fail cause a program transfer to the CCFR central control switch routine CCFRRACC. Later, a switch of the active and standby central controls occurs followed by a B-level interrupt.

4.47 If the bootstrap is successful, the status of the individual stores is updated in program store. The stores then are configured into or out of the active system according to the updated status.

4.48 If the bootstrap was entered from PCRV, program control is returned at this point. All substitute stores are flagged, but the data has not been updated. The PCRV program controls the update of these stores.

4.49 If the bootstrap was entered from CCFR or on D level, the substitute stores are updated by pumping a copy of the data block from file store. If a copy is not available, the substitute is zeroed. On conclusion of the update, program control is transferred to the full access test for D-level entries or to CCFR.

4.50 Successful completion of this access test results in a return to call processing via MARP.

4.51 If the bootstrap is reentered after having passed bootstrap previously, a check is made

to determine whether bootstrap has passed an excessive number of times. Excessive passing of bootstrap tests causes CSFR to change the fundamental strategy in a prescribed way for this entry and each later reentry into bootstrap.

4.52 The first change of the bootstrap actions switches the active and standby call store buses before the bootstrap tests. A later reentry to bootstrap causes the suspect call store to be forced out of the bootstrap screening tests, regardless of whether it has passed bootstrap tests. Each subsequent reentry causes another store to be forced out. If this process of isolating each store in the community one at a time fails to isolate the cause of the problem, another reentry causes the program to go directly to the bootstrap failure leg of the program. This would cause a B-level interrupt and the recovery would be controlled by PCRV.

D. Noninterrupt Level Functions

4.53 The CSFR program performs several noninterrupt level functions. The CSFR program administers all changes in the configuration or status of the call stores and the call store buses. Some of these functions are also done on interrupt level by the same routines or subroutines. However, all base level functions must conform to the base level segmenting requirements. Consequently, these functions are normally done as clients of MACP or are done as a subroutine of other maintenance programs.

4.54 These base level functions are primarily related to hardware audits, diagnostic requests, and routine exercise tests.

4.55 The CSFR program administers audits of all call store/call store bus-related lamps and indicators located on call store frames and the MCC. The CSFR program uses subroutines located within the Master Control Console Common Control and Monitor Program (MCCM) to check or change the state of the lamps. The CSFR program assures that the lamps show the correct state as indicated by CSFR status tables. Stores whose state is found to be indefinite are removed for a diagnosis to determine their correct status before updating the lamps and status tables.

4.56 The CSFR program performs both prediagnostic and post-diagnostic functions for all diagnostic requests for the call stores/call store buses. For prediagnostic functions, CSFR assures that the store to be diagnosed is available and replaces the store with a substitute copy of the memory block if necessary. The store to be diagnosed is then configured out of the system and made available to the diagnostic program. For bus diagnosis, CSFR must assure that the system has access to a complete copy of call store over the remaining bus. This involves changing the bus routing controls in central control as well as controls in all call stores.

4.57 Post-diagnostic functions include verification of the success of the diagnostic before returning the unit to service. The CSFR program administers the return of the unit to service.

4.58 The CSFR program also does configuration functions for diagnostics of the central control.

4.59 Removal or restoration to service of a store usually involves more than a simple alteration of the configuration of the hardware. In many instances, an update of a substitute store must be completed before a substitute store may be removed from service. Always, an out-of-service store must be updated to the present contents of its assigned memory block before it can be placed in service. If a substitute store had been serving in its place, the substitute must also be updated to its normal block thus restoring duplication as soon as possible.

4.60 Routine exercise requests for store diagnosis (midnight routine) are also administered by routines located within CSFR.

4.61 Finally, CSFR also contains a set of routines that are used by the Systems Update Program (SYUP). Included are routines to remove, configure, and restore call stores and call store buses for system updating.

CSFR—PROGRAM STRUCTURE

A. General

4.62 The CSFR program (Fig. 8) is divided into two pidents, CSFRNORM and CSFRBASE. It contains the CSFR code that is essential to the system recovery programs. Pident CSFRBASE is located in program store memory block 0 along with other system recovery programs. Pident CSFRNORM contains all other CSFR code including the control

routines for D-level recovery as well as the noninterrupt recovery routines. Pident CSFRNORM uses routines located within CSFRBASE and CSFRNORM to accomplish the call store fault recovery functions.

4.63 The CSFR program also uses numerous subroutines located within other programs. The following programs contain important subroutines that are used by CSFR:

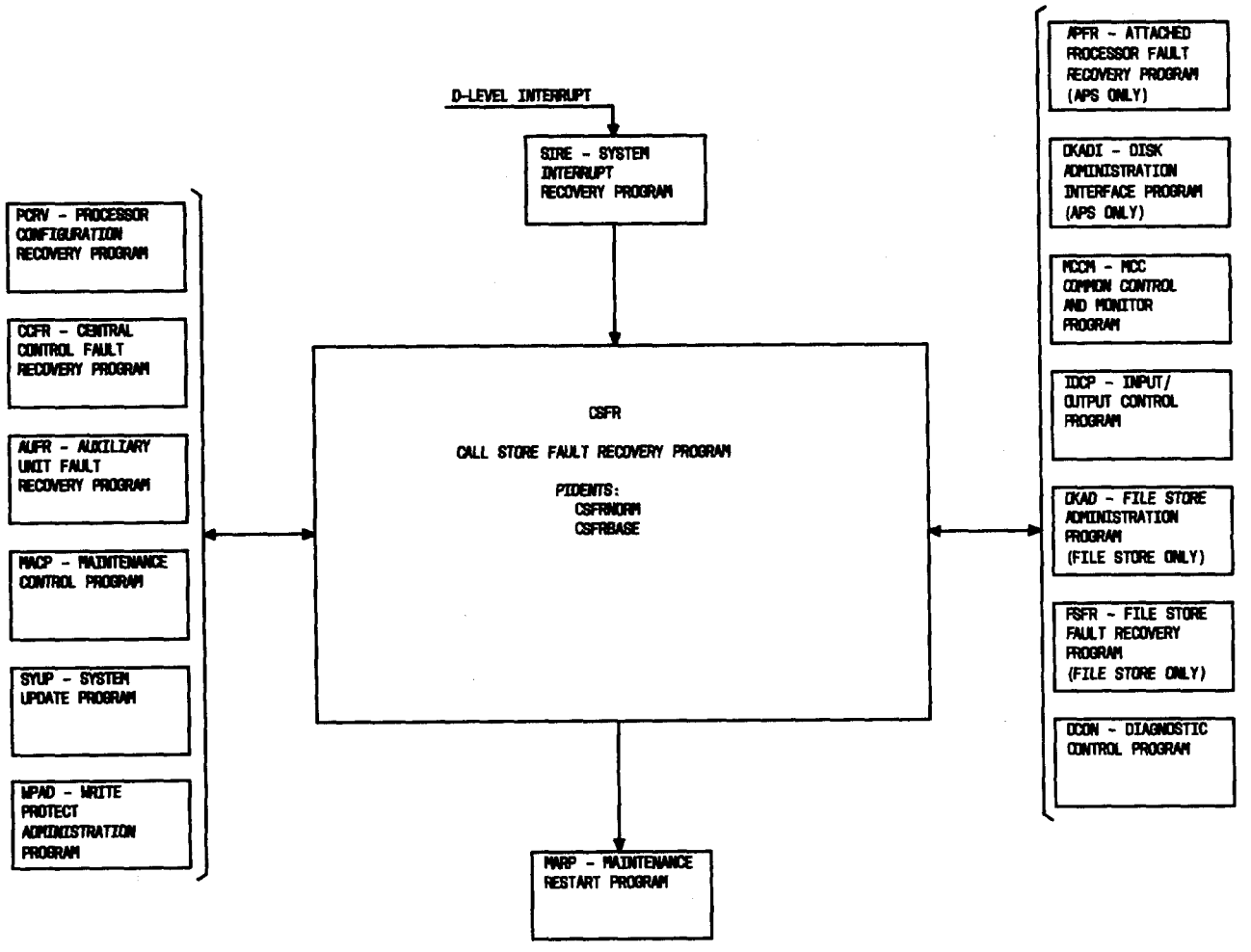
(a) MACP—Maintenance Control Program subroutines are used to set up base level tasks.

(b) MCCM—Master Control Console Common Control and Monitor Program subroutines are used to verify and update the states of various lamps and control indicators on the call store frame and the MCC.

(c) IOCP—Input/Output Control Program subroutines are used to initiate output messages.

(d) DKAD—File Store Administration Program subroutines are used to perform "pumps" (file store to call store copy) of store memory blocks. (File store system only.)

(e) ♦DKADI—Subroutines are used to perform "pumps" of store memory blocks. (Attached Processor System [APS] only.)♦



♦Fig. 8—Call Store Fault Recovery Program (CSFR) Interfaces♦

(f) AUFR—Auxiliary Unit Fault Recovery Program subroutines are used to stop AUs during CSFR recovery functions.

(g) CCFR—Central Control Faulty Recovery Program subroutines are used for all central control configuration changes during testing.

(h) FSFR—File Store Fault Recovery Program subroutines are used to restore the file store controllers to their previous state after a pumping operation if possible. (File store system only.)

(i) ⧫APFR—Attached Processor Fault Recovery Program subroutines are used to restore the APIs to their previous states after a pumping operation. (APS only.)⧫

(j) WPAD—Write Protect Administration Program subroutines are used to verify a write protect error before restart.

4.64 There are several other important program interfaces in addition to the subroutine interfaces.

(a) SIRE—The CSFR program is always entered from the System Interrupt Recovery Program after a D-level interrupt.

(b) PCRV—The CSFR program has several interfaces with the Processor Configuration Recovery Program during system recoveries, call store bootstraps, and PCRV initiated pumps of call store.

(c) DCON—The CSFR program has several exits to diagnostic control to start or stop a diagnostic of call store.

(d) SYUP—The CSFR program interfaces with the System Update Program to remove, configure, and restore call stores and call store buses for system updating.

(e) MARP—The CSFR program interfaces with the Maintenance Restart Program to set up the restart and the output data and then normally exits to restart.

4.65 The CSFR program control structure and bootstrap operations were described as a part of Interrupt Recovery. A description of the remaining major routines follows.

B. Full Access Test

4.66 Before returning to call processing, CSFR must verify that a complete copy of call store is available. The full access test routine may be entered to verify a single call store block of memory or to verify the entire call store community. The single block verify function is used as a part of the recovery function, whereas the successful completion of the complete copy verify on D level leads to restart.

4.67 This test routine is also used by such non D-level programs as PCRV and CCFR to assure that a complete copy of call store is available. These programs use entry point CSFRACC and, on completion of the tests, control is returned to the calling program with a pass/fail indication.

4.68 For D-level recovery, the initial entry to the access tests determines whether the central controls were operating in step at the time of the interrupt. If they were, an attempt is made to start the standby in step with the active before performing the access tests. If the standby is placed in step with the active, then routine matching is established without interrupts. Also, the active central control is set to halt matching and stop the standby if a mismatch is detected. The CSFR program always uses CCFR service routines to make any required changes in the configuration of central control. The access tests are then done sequentially on each K-code assigned to the call store community or the single specified K-code for a one block verify.

4.69 If the tests pass, the following actions are taken. If the central controls were in step during the access tests, the standby is stopped and a check is made for mismatch errors. If the central controls mismatched, an ASWF is detected, or an DPWEF is detected in the standby central control during the tests; a request is made for a diagnosis of the standby central control. This diagnosis is done on base level after interrupt recovery is completed. The standby call store bus is then removed from service and the full access test is reentered. However, if no errors occurred, CSFR requests a restore of the standby central control on base level.

4.70 If the tests were performed successfully with no mismatches or with only the active central control, report data is prepared for output. The CSFR program then requests diagnosis for all units removed by CSFR or, in the event a bootstrap was en-

tered, a diagnostic is requested for all out-of-service call stores and call store buses.

4.71 At this point, control is returned to AUFR for D levels caused by AU access failures, and call store D-level failures are transferred to MARP for restart. Write protect errors are transferred to WPAD before restart.

4.72 If the access tests fail, checks are made of the number of previous failures. The results of the checks depend on whether bootstrap was entered before and whether the buses were switched. If either of these actions occurred, bootstrap is entered. If not, further analysis of the failure is done. If the failure was detected in the standby central control, the standby store is removed. If no standby store existed, the standby bus is removed. If neither standby bus nor standby store exist, bootstrap is entered. If the failure was detected in the active central control, a check is made for a Trap Refresh Data Parity Failure which if detected is corrected. After the first failure, the first fail flag is set as an indicator for possible later failures.

4.73 After the fourth detected failure of the active central control or active bus, a switch of the active and standby bus is done followed by a removal of the standby bus.

4.74 If the first fail flag is set and if error analysis is set, a check is made to determine whether an update of a substitute call store is in progress. If an update is in progress, a transfer is made to the FINFILL subroutine to allow the update to complete on interrupt level. Later, the access tests are reentered to assure that a complete copy of call store is available after the substitute store is placed in service.

4.75 If an update is not in progress, CSFR transfers to the call store bootstrap.

C. Call Store Remove—Replacement Ready

4.76 Upon entry, this routine removes the specified call store member from service and replaces it with another call store whose member number is also supplied to the routine. The replacement call store may or may not have been in service before entry; however, it is assumed to contain the correct memory block.

D. Find Duplicated Call Store to Serve Another Memory Block

4.77 Before a nonduplicated call store can be removed from service (except during bootstrap), a substitute store must be selected for it from the set of duplicated stores. If a substitute is found, it is removed from service and set for updating to its new memory block. A control write is used to write the unit to its new status: K-code changed to replace unit being removed, RO set to equal the active bus, maintenance flip-flop reset, the ANS0 and ANS1 flip-flops are reset, and the communications reply inhibit is set.

> ***Note:*** Four 64K stores may be used to replace a single 256K store.

4.78 A Maintenance Control Program (MACP) routine job request is then made to copy (update) the store to its new memory block. The update is done on base level by reading from the store being removed from service and writing back into it and the new substitute.

4.79 After completion, the routine returns control to the client routine with a success or fail indication.

E. Qualify Suspect Store for Update

4.80 Upon entry, an attempt is being made to use a suspect call store and therefore avoid delays because of pumping from file store or bootstrapping the call store community. This routine qualifies the store before the attempt is made to use it.

4.81 This routine does a test of the suspect store. The test includes an access test as well as failing address tests. If the test passes, the store is considered to have encountered an error (as opposed to a fault). The error analysis flag is set in the unit's status table, and the full access test for the call store community is entered in preparation for the resumption of call processing.

4.82 If the test fails, the store is immediately removed from service. If a store had previously been selected as a substitute, its update is completed on the interrupt level; or, if the store was duplicated, its mate is set up as the active unit. Otherwise, the bootstrap routines are entered.

F. Complete Recovery After an Update Error

4.83 This routine (entry point FINFILL) is entered when an error is encountered in a memory block that is being updated; ie, the contents of one store is being copied into another. The source of the error must be isolated to either the store being copied or the store being updated.

4.84 If the error originated in the store being updated, it is removed from service as if it were a faulty store in a duplicated pair and the update in progress flag is reset. On the other hand if the error originated in the store being copied, the update is completed in one of two ways, depending on the category of information in the memory block. Information backed up on file store is updated from file store, whereas information not backed up on file store is copied unconditionally; ie, it may be no good. This would be an abnormal case (a duplicated store failing when its mate is unavailable).

4.85 The update is completed on interrupt (only a 64K block of memory is updated as described in paragraph 4.77, Note). Failure to complete the update results in a bootstrap and/or entry into AUFR if the file store controller shows a trouble condition. Also, the store status is marked to show that it may have been zeroed. The MARP program may later initiate an office phase.

G. Update a Call Store by Copying Its Assigned Block

4.86 Routine CSFRCOPY is used to update a store to the present contents of its assigned memory block by reading from the in-service store and writing the results into both stores. This routine is normally done on base level. The initial entry is through the routine request table of MACP.

4.87 The request for a copy update is made via the control write new status (CWNEWST) routine when:

(a) A store is to be updated to replace one already in service for which there is a removal or diagnostic request.

(b) A store is to be updated to serve as an alternate to a nonduplicated store that has encountered an error.

(c) A store is to be updated before restoring it to service as a nonduplicated store or as one of a duplicated set of stores.

H. Call Store Community Status Update

4.88 A copy of the call store community status is maintained in a program store table. The status table is assumed to be correct when the store community's configuration is altered by a routine other than bootstrap. The routine is entered to update the status table to agree with the actual store community configuration when there is reason to believe that it may not be accurate; ie, after a pump of program store block 0. The routine may be entered under program control; ie, from PCRV.

4.89 The correct status is determined by pulsing the maintenance flip-flop for each call store and updating the status according to the answer. If no answer is received, the call store is assumed to be out of service.

I. Call Store Restore

4.90 On entry, this routine verifies that the specified call store can be returned to service. If the store can be returned to service, the store is configured to be updated to its assigned memory block. The store is updated to its assigned memory block by the CSFRCOPY routine as a base level task. On completion of the update, program control is returned to this routine. The store is then placed into service, and a message is printed to show that the restore is complete.

4.91 The restore routine may be entered as a result of a TTY request to restore or as a result of a program request to restore after a successful diagnosis of the store.

J. Call Store Removal Routine

4.92 On entry into this routine, the specified call store is removed from service if its duplicate member is available. Also, in the case of manual TTY or MCC requests for removal, the specified store is removed if a call store is available to substitute for the requested unit. This routine is also used when a diagnostic has been requested for a store that is in service. A portion of this routine is used as a subroutine to remove a store from service for which a copy has just been updated.

4.93 This routine has several entry points, each of which removes the specified store under different conditions as follows:

(a) A global entry (CSFRSRMV) is used by TTY, call store control panel, or power alarm requests to remove a call store.

(b) Another entry is used for diagnostic requests for an in-service store. The store is removed from service, and the diagnosis is started. If the diagnosis is being run as a restoral request, the call store restore routine is entered if the diagnosis is completed with an all tests pass (ATP).

(c) Another entry is used to "swap units," ie, a store has been updated to replace the store being removed. The replacing store is made the active store and the requested removal is performed. The out-of-service lamps are updated as for all removals and restores.

K. Configuration Change Routine

4.94 As call store buses are restored to or removed from service or are switched from serving the active central control, etc, the call stores listening to and responding to those buses must have their routing flip-flops altered to conform to the new conditions. The configuration change uses two subroutines. One subroutine alters the call store status table to agree with the new bus status, whereas the other subroutine sets the call store routing flip-flops to conform with the configuration found in the status table. When all stores conform to the status table, control is returned to the routine or program that requested the configuration change.

4.95 This routine removes the standby bus from service, switches the active and standby buses, returns the standby bus to service, or establishes a configuration based on the status table.

5. PROGRAM STORE FAULT RECOVERY PROGRAM—PSFR

INTRODUCTION

5.01 The PSFR program has three basic functions in the 1A Processor. First, PSFR is entered from the System Interrupt Recovery Program (SIRE) for program store access failures that result in E-level interrupts. Later, PSFR resolves program store related problems and steers any nonprogram store problems to the appropriate recovery program such as the Central Control Fault Recovery Program (CCFR) or the Auxiliary Unit Fault Recovery Program (AUFR). (Some central control and AU problems may initially appear to be program store problems.) Second, PSFR provides a "bootstrap" capability for the program store community. Finally, PSFR provides several service routines for use by other programs.

PROGRAM STORE ORGANIZATION AND FEATURES

5.02 The 1A Processor program store community contains several individual program stores. The number of program stores varies according to the type and size of the switching office installation.

5.03 The store community may consist of 64K (1K= 1024 words) core-type stores (two per frame), 64K semiconductor-type stores (up to six per frame), or 256K semiconductor-type stores (up to eight call stores and six program stores per frame) or a combination of these frames. All three frame types can operate at 1400 ns (slow); in addition, the 256K store can operate at 700 ns (fast) when not mixed with other types of stores in the same community. The maximum number of program stores is 24 members (0 through 23) of the 64K size or its equivalent in 256K size.

5.04 The central controls access the program stores via duplicated program store buses that interconnect every program store frame with both central controls (Fig. 9) (the buses may have as many as two branches). A unique address identifies each program store word location. This address consists of a K-code and a data location address (Fig. 10). The K-code portion of the address identifies the specific 64K program store memory block to be addressed. The K-code also identifies the specific program store to be addressed (ie, one K-code identifies the specific 64K program store, but any one of four consecutive K-codes identifies a single 256K store). The data location identifies the specific data location to be accessed within the 64K program store memory block.

5.05 Two program stores are normally designated as spares (called rovers) and can be assigned to replace any program store that malfunctions. During normal operation, the spare 64K program stores contain duplicate copies of information stored in program store memory block 0 and memory block 1. Consequently, these spare stores are normally assigned the same K-codes as those containing memory block 0 (K-code 20) and memory block 1 (K-code 21). Program store memory block 0 contains the system recovery programs. However, if the spare stores are 256K, the first eight blocks of program store memory are duplicated. Here, the spare stores are assigned K-codes 20 through 23 and 24 through 27.

5.06 The K-code of any program store may be changed under program control by PSFR. The

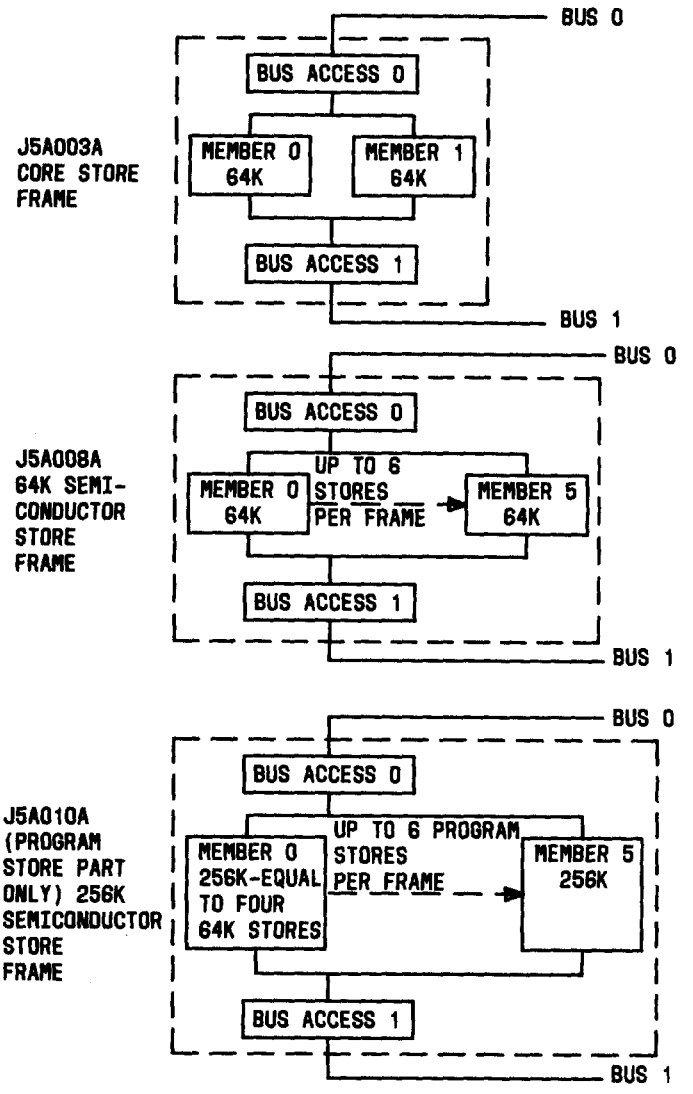
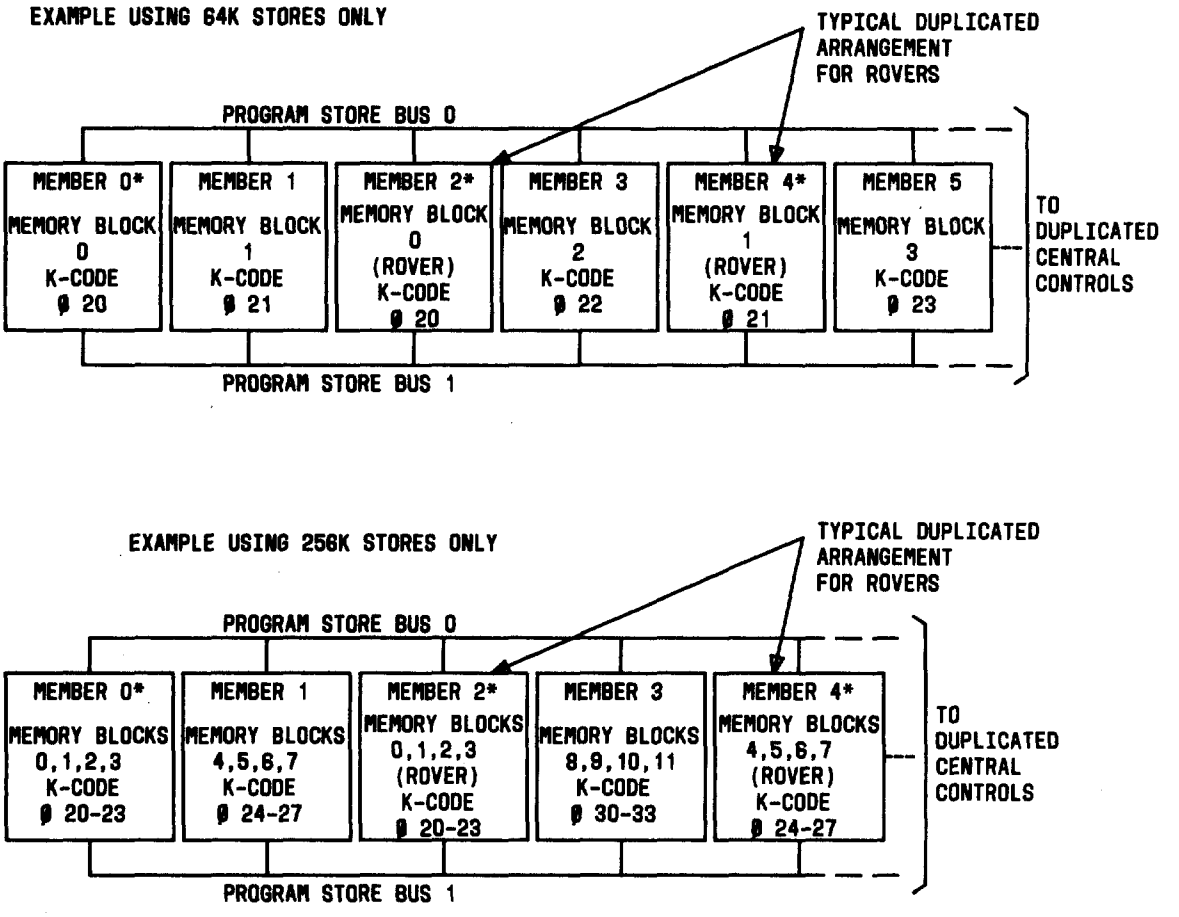


Fig. 9—Functional Layout of Three Types Used for Program Store

changeable K-codes allow PSFR to substitute a spare duplicate program store for another store if a failure occurs (four 64K stores may be used to replace a single 256K store). The PSFR program can uniquely identify any member of the program store community regardless of its K-code by pulse point access using generate control pulse (GCP) instructions.

> ***Note:*** When 64K and 256K stores (1400 ns only) are mixed within a program store community, several combinations are possible for duplicate store arrangements. Duplicate stores may be a pair of 64K stores, a pair of 256K stores, or a 256K store and four 64K stores.

5.07 Three program stores are designated as processor configuration controlled stores. These are the stores containing program store memory block 0 (K-code 20) and the two spare (rover) program stores normally assigned as backup for program store memory block 0 and 1 (for 64K stores) or memory blocks 0 through 3 and 4 through 7 (for 256K stores). For a gross system malfunction, these processor configuration stores can be accessed by central control via the processor configuration data interface. This interface can be used to force the selected program store to change its K-code to K-code 20 (20 through 23 for a 256K store). This interface also provides direct access to the RO flip-flops and mainte-



LEGEND: * = PROCESSOR CONFIGURATION SEQUENCER CONTROLLED STORES (0,2,4)
K = 1024 WORDS
ø = OCTAL NUMBER

NOTE: 64K AND 256K STORES MAY BE MIXED, BUT ONLY IF THE 256K STORES ARE OPERATED "SLOW." THE PROGRAM STORE COMMUNITY MAY NOT EXCEED THE EQUIVALENT OF 24 64K STORES.

Fig. 10—Program Store Organization—Simplified

nance flip-flops of the processor configuration program store.

5.08 The bus selection flip-flops located in central control and in the program stores control the bus routing of communications between central control and the program stores. The duplicated buses are referred to as 0 and 1. Always, one pair is called the active bus and the other is called the standby bus. Central control bus selection flip-flop functions (Table C) are as follows:

(a) PBA—When PBA is reset, bus 0 is the active bus and bus 1 is the standby. This results in the active central control transmitting and receiving over bus 0 and the standby central control using bus 1. When PBA is set, the active central control uses bus 1 and the standby uses bus 0.

(b) PBO—When PBO is set, the active central control transmits on both the active and the standby buses but receives on only the active bus. The standby central control does not transmit data, but does receive on the standby bus.

(c) PBT—When PBT is set, the active central control sends and receives on the active bus and the standby central control receives on the active bus but does not send on either bus.

Table C summarizes the configuration capabilities of the bus from the central control end of the bus.

5.09 The controls located in the program stores are as follows:

(a) RO—Selects the bus over which the store receives information from central control. When reset, the store receives data on bus 0. When set, the store receives on bus 1.

(b) ANS0—When set, the store sends data to central control over bus 0. When reset, the store does not send on bus 0.

(c) ANS1—When set, the store sends data to cen-

TABLE C

CALL STORE BUS CONTROLS LOCATED IN CENTRAL CONTROL

CONTROL FLIP-FLOPS (NOTE 1)			CENTRAL CONTROL		
PBT	PBO	PBA	SENDS ON PROGRAM STORE BUS (NOTE 2)	RECEIVES ON PROGRAM STORE BUS	
0	0	0	0	0	
0	0	1	1	1	
0	1	0	0 & 1	0	Active
0	1	1	0 & 1	1	Central
1	0	0	0	0	Control
1	0	1	1	1	
1	1	0	0 & 1	0	
1	1	1	0 & 1	1	
0	0	0	1	1	
0	0	1	0	0	
0	1	0	X	1	Standby
0	1	1	X	0	Central
1	0	0	X	0	Control
1	0	1	X	1	
1	1	0	X	0	
1	1	1	X	1	

Note 1: 0 = Reset, 1 = Set

Note 2: X = No Bus Transmission

tral control over bus 1. When reset, the store does not send on bus 1.

When the store is in a maintenance or control mode, the store returns data to central control on the bus designated by RO regardless of the state of ANS0 and ANS1. Table D summarizes the controls located in program store.

5.10 The various combinations of settings of these controls allow the program to isolate most bus faults with a minimal effect on the system operation.

5.11 The 256K store has two additional access controls that are used by the maintenance and recovery programs. The first control is the SLOW flip-flop. When set, the store operates at a 1400-ns rate. The store must operate at this rate when mixed with 64K stores in the same community. When the SLOW flip-flop is not set, the store operates at the 700-ns rate. The second control is a set of four flip-flops: 0KBR, 1KBR, 2KBR, 3KBR. Each of these flip-flops controls the read access of the corresponding 64K block of memory. When any of these are set, the corresponding 64K block of memory will not respond to a read access and will not return error signals for either reads or writes. These four flip-flops are used during copy operations to update a store in 64K increments.

5.12 Bus access to the program stores may be accomplished in any one of three modes of operation: normal, maintenance, or control mode. The mode of operation depends on the setting of the maintenance flip-flop in the program store and mode bits which accompany the address information for each access of the store.

5.13 Normal mode operation allows central control to access those program stores whose K-code

TABLE D

PROGRAM STORE BUS CONTROLS LOCATED IN PROGRAM STORE

CONTROL FLIP-FLOPS (NOTE 1)			PROGRAM STORE	
RO (NOTE 2)	ANS0	ANS1	SENDS ON PROGRAM STORE BUS (NOTE 3)	RECEIVES ON PROGRAM STORE BUS
0	0	0	X	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0 & 1	0
1	0	0	X	1
1	0	1	1	1
1	1	0	0	1
1	1	1	0 & 1	1

Note 1: 0 = Reset, 1 = Set

Note 2: On maintenance and control orders, the program store answers the bus designated by the RO flip-flops regardless of the state of the ANS0 and ANS1 flip-flops.

Note 3: X = No Bus Transmission

matches the one sent via the bus, provided the maintenance flip-flop is not set in the store. Maintenance mode operation allows central control to access a store whose K-code matches the one sent, provided the maintenance flip-flop is set. Control mode operation enables central control to alter the status of control and maintenance flip-flops within a store without affecting the data stored in the memory module. Control mode operation depends on the mode bits matching the setting of the maintenance flip-flop (set or reset).

5.14 Maintenance load and store instructions (orders) provide the flexibility to send maintenance, control, or normal mode orders to the stores. These orders provide access to maintenance, status, and control flip-flops located within the program store. In addition, these orders control the sending of valid/invalid parity, clock pulses, etc.

5.15 Memory within the program store community primarily provides storage for central control program instructions. However, office dependent data, which must be readily available to the central control but which is infrequently changed, is also stored in program store. Except for memory blocks 0 and 1, or 0 through 7 for 256K stores, program store data is not duplicated because a backup copy of the data is available on file store. If a software or hardware fault causes mutilation of the data stored in program store, the backup data can be "pumped" into the faulty store or its replacement from file store.

5.16 Since the program store community is suspect when an E-level interrupt occurs, the basic sanity of the program store community is established by a program executed from call store. Therefore, an E-level interrupt causes control to be passed to a program located in the protected range of call store memory. The status of the program store community is also checked by a program executed partially from call store. Before transferring to a call store program, the CSPGM flip-flop must be set in the central control. The CSPGM flip-flop is set by the hardware interrupt sequencer on E-level interrupts.

5.17 All tables and scratch pad words associated with the call store program for program store recovery reside in the duplicated area of call store. The program store recovery itself and associated translations may be located in the nonduplicated area of call store since it is backed up on file store.

5.18 In a normal 1A Processor configuration with both central controls and both program store buses in service, duplicate stores are configured to opposite buses; ie, one store of the pair receives from and sends to the active program store bus while the other store of the pair receives from and sends to the standby program store bus. The nonduplicated stores are configured to receive from the active bus and send to both buses.

5.19 When one program store bus is out of service, one store of each pair of duplicated stores is configured to receive from and send to the active program store bus. The other stores of the duplicated pairs are configured to receive from the active bus but are not configured to send on either bus.

5.20 The primary mechanisms for trouble detection in the program store and related equipment are as follows:

(a) The program store returns an all-seems-well (ASW) signal to central control for every read or write operation that appears valid at the store. Failure to return ASW shows a trouble condition.

Note: The 256K stores operating fast (700-ns) cannot detect error conditions in time to return an all-seems-well failure (ASWF) during the 700-ns cycle. For those error conditions that normally return ASWF during slow (1400-ns) operation, the store returns the data parity or write enable failure signal (DPWEF) during the next 700-ns cycle instead.

(b) The DPWEF signal is used only by 256K stores operating fast. This signal is returned during the next 700-ns cycle after the error is detected.

(c) The program store returns an ASWF signal on every read or write operation that appears invalid at the store.

(d) Each address received by a program store is accompanied by a parity bit covering the address (data word location) and the K-code. The program store checks this parity. Incorrect parity causes the store to read at the indicated address (no writing is done) and to suppress the ASW signal while generating the ASWF signal.

(e) All data written into program store is accompanied by two parity bits for each 24 bits of data. These parity bits that cover the address (including K-code) and data are checked by the store

itself and are transmitted back to central control whenever the data is read out. If the data parity check at the store fails, the write is completed (parity check results are not completed in time to suppress the write) but ASW is suppressed and ASWF is generated. The central control then checks the received parity against the computed parity over data received and the address.

(f) Semiconductor stores autonomously perform a periodic (about every 184 ms) data parity check of every address. If this autonomous data parity check fails, the store sets a flag flip-flop and will return ASWF and no ASW the next time it is addressed.

(g) All data to be written into program store must be preceded by a write enable pulse timed to precede the readout strobe. The write enable prepares the store to receive information from the data bus. If the pulse is not received on a write order, the cycle is completed as a read operation and an ASWF signal is generated while ASW is inhibited.

(h) Each store contains an error summary register. Setting an internal error indicator in the error summary register sets the maintenance flip-flop, enables the ASWF signal, and inhibits ASW. Errors which occur after time for transmission of ASWF and ASW are detected by central control on the next operation to the store. Internal errors are recorded in the error summary register when conditions such as access circuit trouble are detected. Also, certain access circuit points are monitored and checked for validity. These error indicators can be used by PSFR to localize the fault or error.

Central control causes an E-level interrupt when any of the above error conditions are detected.

5.21 For a more detailed description of program store refer to the following sections:

SECTION	TITLE
254-201-010	Call Store/Program Store—Description
254-201-011	Call Store/Program store—Theory
254-201-012	Call Store/Program Store, 1400-ns Semiconductor Store—J5A008A—Description
254-201-013	Call Store/Program store, 1400-ns Semiconductor Store—J5A008A—Theory
254-201-014	Call store/Program Store, 256K Semiconductor Store—J5A010A—Description
254-201-015	Call Store/Program Store, 256K Semiconductor store—J5A010A—Theory.

PSFR—FUNCTIONS AND STRATEGY

A. General

5.22 The primary purpose of PSFR is to return the system to normal call processing (Fig. 11) as quickly as possible after a fault or error condition has been detected in the program store community. Therefore, whenever possible, PSFR will remove the faulty store on a first-look basis. The first-look approach works in the same way as call store fault recovery first look. The first-look approach uses error indicators in the central control, the bus configuration, and the store status to identify the faulty store.

5.23 When the trouble is located in duplicated program store, the suspect unit is removed from service, and the remaining unit is set to operate as if it were not duplicated. An unduplicated block of memory requires further analysis. Program store fault recovery attempts to find a rover store that can be loaded with a copy of the suspect memory block. If it is able to select a rover store and start the copy of the suspect store, it then checks the error history (kept by PSFR) of the suspect store. If the history is acceptable, the store is left in service until the rover update is completed.

5.24 If the error history is unacceptable, the suspected unit is removed from service and the system must wait until the rover is filled from file store. Also, the program checks to see if a previous error has resulted in a rover store being prepared as a duplicate for the suspected block of memory. If a rover has been updated, the rover is placed in service and the suspect unit is removed. If a rover is in the process of being updated, the system waits for the update to be completed and replaces the suspect unit with the rover store.

5.25 After a configuration of program stores has been selected, an access test is done on each

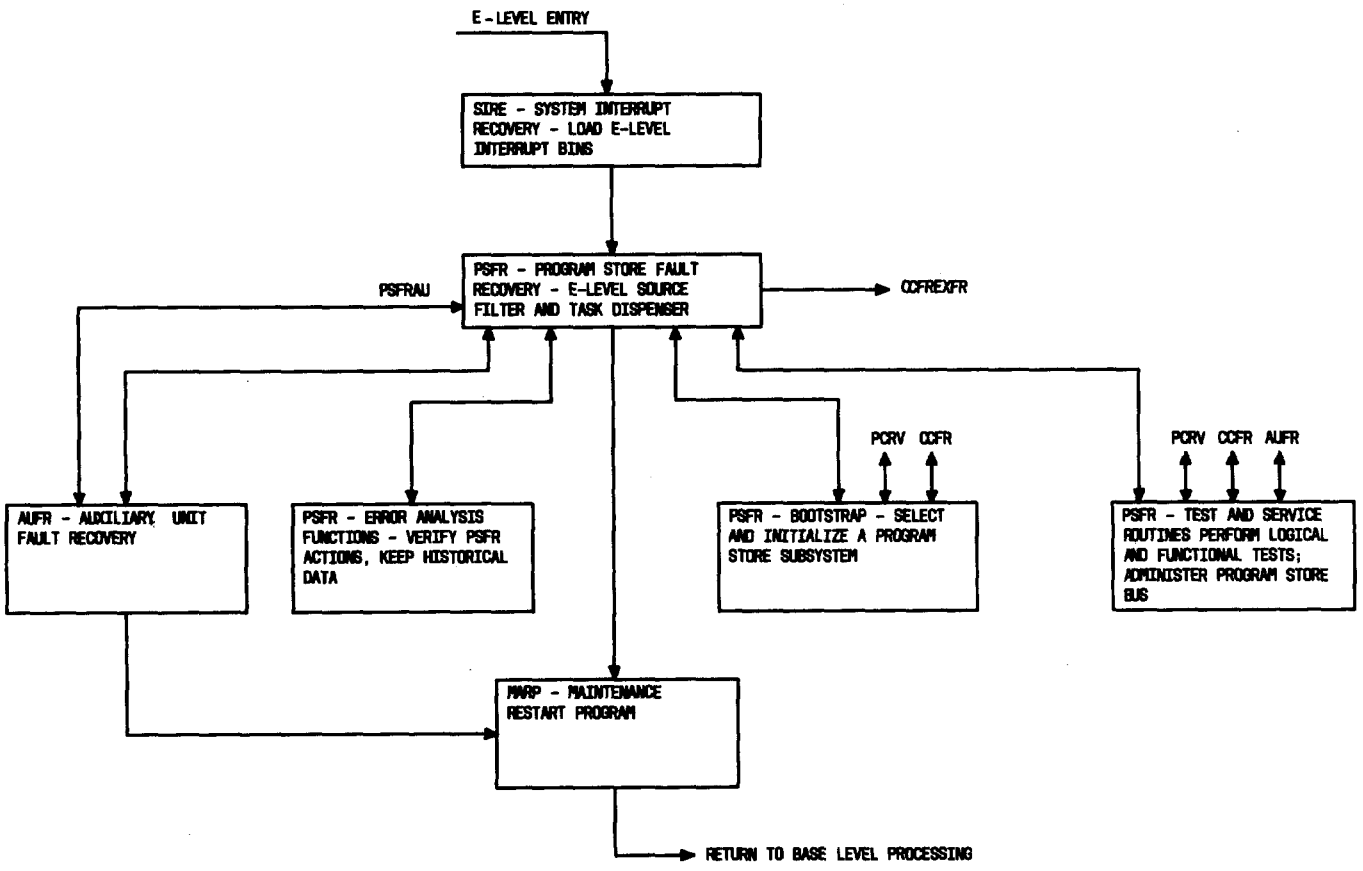


Fig. 11—Program Store Fault Recovery—Program Flow and Interfaces—Simplified

memory block to verify the integrity of the program store community. Failure of the access test after the suspect store has been removed from service can cause the program to transfer to the program store bootstrap routine.

5.26 Bootstrap attempts to assemble a complete copy of program store using only stores which pass the bootstrap qualifying tests. The bootstrap is considered successful if a full copy of program store (with or without the use of rover stores) has been assembled. Failure of bootstrap results in a transfer to the Processor Configuration Recovery Program (PCRV) to switch central controls (B-level interrupt). After a successful fault recovery, control is returned to normal processing by transferring to the Maintenance Restart Program (MARP).

5.27 In addition to the normal interrupt recovery, program store fault recovery may be entered for access tests from the CCFR or AUFR programs. Also, the program may be entered by PCRV to select a configuration of program stores or by PCRV or CCFR to bootstrap the stores. For these entries, program control is returned to the requesting program at a successful or unsuccessful return point after completion of the function.

5.28 Finally, PSFR contains many service routines that are used on base level to perform noninterrupt level maintenance functions on the program stores. These service routines are normally run as clients of MACP in conjunction with routine exercise, diagnosis, or off-line requests.

B. Interrupt Recovery

5.29 An access failure of a program store will cause an E-level interrupt. After the interrupt, control is passed to the System Interrupt Recovery Pro-

gram (SIRE) which stores the appropriate data in the E-level interrupt bins. The SIRE program then transfers control to PSFR where PSFR does a filter function before attempting the recovery.

5.30 There is only one E-level interrupt source (program store access failure).

> ***Note:*** The Auxiliary Unit Fault Recovery Program (AUFR) may encounter access problems with the program store community. These problems are referred to PSFR by AUFR at entry point PSFRAU. These problems may include a write protect failure. Recovery from this type failure is essentially the same as for an E-level program store access failure. However, for a successful recovery, control is returned to AUFR rather than exiting to MARP.

5.31 First, PSFR examines the failing address and verifies its validity. If the failing address is an unequipped program store memory block, then a flag is set to show an out-of-range addressing error has been made by the program. The PSFR program then exits to MARP to restart normal processing.

5.32 However, if the failing address is a valid program store address, PSFR continues the recovery process by testing the program store bus. The program tests the bus by sending several patterns of test data to the program stores to be returned to central control and verified. The test data is not written into memory but is simply looped back to central control via the reply portion of the bus. If the bus test fails, control is transferred to the bus switch routine.

5.33 The bus switch routine attempts to switch the active and standby program store buses. However, if the standby bus is already out of service or if the active bus has been forced (manually selected) from the master control console (MCC), the buses cannot be switched and program control is transferred to the bootstrap routines. If the bus switch is successful, the new standby bus is removed from service and the routing flip-flops in the stores are updated to reflect the change. Program control is then transferred to the PSFR access tests to verify that a complete copy of program store is available before exiting to MARP.

5.34 If the program store bus passes the tests, a check is made to determine whether the failing store is duplicated. If the store is duplicated, it is removed from service (unless its error count is greater than zero but less than five) and its mate is configured to operate alone. The store error count is incremented by one on each interrupt that is caused by that store. The error count is decremented by one every half hour. If the store error count is greater than zero but less than five, PSFR will attempt to keep the suspect store in service as the standby unit. The PSFR program does a gross access test on the suspect store to qualify it for service. The access tests are entered to verify that a complete copy of program store is available before exiting to MARP.

5.35 If the failing store is not a duplicated store, any one of several recovery actions is possible. First, if a previous error had resulted in selection of a rover store for updating to the contents of the failing memory block, then the recovery is transferred to the FINFILL routine. The FINFILL routine determines whether the failure is in the original failing store or if the rover is being updated to act as a substitute. If the rover has failed, it is removed from service and access tests are used to verify that a complete copy of program store is available before exiting to MARP.

5.36 However, if the original failing store has failed again, it is removed from service and the rover is configured to take its place. Routine FINFILL then checks the progress of the update. When the update has been completed, program control is transferred to the program store access tests to verify that a complete copy of program store is available before exiting to MARP.

5.37 If the update has not been completed, FINFILL determines whether the failing memory block is block 0 or another. If block 0 has failed, the rover is "pumped" with a copy of the memory block from file store. If any other memory block has failed, the program must determine whether a complete copy of the failing memory block is available in file store. Any part of the memory block that is backed up in file store is pumped and any part that is not backed up is zeroed. If either pumping or zeroing of the failing memory block is unsuccessful, the processor configuration circuit is activated (B-level interrupt occurs). Successful completion of the pumping or zeroing is followed by a transfer to the program store access tests before exiting to MARP.

5.38 If the calling nonduplicated store was not in the process of having a rover updated to take

its place, its error history is checked. If the error history is unacceptable, an attempt is made to find a rover store for update. If no rover is available, program control is transferred to bootstrap. If a rover is available, it is set up to receive update. An access test is then done on the failing store in an attempt to qualify it for use during the update procedure. If the store fails the qualifying test, it is removed from service and the program transfers to FINFILL to complete the update of the rover on interrupt level.

5.39 If the store passes the qualifying test, the store is marked for error analysis and an update request is made for base level. (On base level, the update is done by reading from the suspect store and writing the data back into the suspect and the rover store.) Before exiting to MARP, the program store access tests are entered to verify that a complete copy of program store is available.

5.40 If a failing nonduplicated program store is found to have an acceptable error history, an attempt is also made to find a rover store for update. However, failure to find a rover does not immediately result in a bootstrap. Instead, PSFR attempts to qualify the suspect store for system use by performing an access test on the single suspect K-code. Failure to pass the qualifying test would then result in a transfer to bootstrap since a complete copy of program store is not available.

5.41 If the store passes the qualifying test, the store is marked as an error analysis store. The program then transfers to the program store access test to verify that a complete copy of program store is available prior to exiting to MARP.

C. Bootstrap Functions

5.42 A bootstrap of the program store community can be undertaken for any of the reasons that have been described to this point. In addition, global entry points are supplied for both PCRV and the Central Control Fault Recovery Program (CCFR). The occurrence of a program store bootstrap and the reason for it are reported via the master control console (MCC) and, if possible, also via the TTY.

5.43 On entry into the bootstrap routines, a check is made to determine whether bootstrap has been passed an excessive number of times on previous interrupts. If bootstrap has been passed excessively, the fundamental strategy is varied. The fundamental strategy is described first.

5.44 The program store bootstrap routines attempt to assemble a complete copy of the program store memory blocks. Entry into bootstrap implies that there is a major problem in the program store community since the first-look routines were unable to come up with a valid configuration of stores. The bootstrap routine starts by selecting one program store bus (the active) over which the bootstrap is to be tried.

5.45 All maintenance flip-flops in the program store community are set, and the RO flip-flops are set to the standby bus. The bootstrap routines then attempt to bring the program stores into the system one at a time.

5.46 The maintenance flip-flop for the store under test is pulsed to toggle the RO flip-flop to the active bus. The bootstrap tests are then performed using the active bus. Maintenance and control tests are done with the maintenance flip-flop set. The maintenance flip-flop is then reset, and the bootstrap tests are completed with the store in the control and normal modes. The maintenance flip-flop is then set at the conclusion of the tests. The results of the bootstrap qualifying tests are then placed in a scratch status table, and tests are started on the next program store.

5.47 When all stores have been tested, the bootstrap results are evaluated by examining the scratch status table. If a complete copy of program store is available, the bootstrap was successful. When a complete copy is not available, the ability of the rovers to complete the copy is checked. Rovers that are selected to complete the copy are flagged by the program to be updated to the appropriate memory block before returning to call processing.

5.48 When a complete copy cannot be assembled even by using the rovers as substitutes, the bootstrap has failed via the active bus. If bootstrap was entered from PCRV, then program control is returned to PCRV with a failure indication. Otherwise, the active and standby buses are switched, and a new bootstrap is attempted on the new active bus. If bootstrap fails on both buses, the processor configuration sequencer is activated and a B-level interrupt occurs. The recovery is then controlled by PCRV.

5.49 If the bootstrap is successful, the software status and store hardware are updated to prepare for a return to call processing. If bootstrap was

entered from PCRV, program control is returned to PCRV at this point. The PCRV program must then assure that all flagged substitute program stores are updated. If the entry was not from PCRV, then PSFR updates the substitute stores to the appropriate memory blocks. If any update fails, the processor configuration sequencer is activated and a B-level interrupt occurs.

5.50 Upon conclusion of the updates, program control is transferred to the access tests for E-level entries or is returned to CCFR for CCFR entries.

5.51 Successful completion of the access tests results in a return to call processing via MARP.

5.52 If bootstrap is reentered after having passed bootstrap previously, a check is made to determine whether bootstrap has been passed an excessive number of times. Excessive passing of bootstrap causes PSFR to change the fundamental strategy in a prescribed way for this entry and each later reentry into bootstrap.

5.53 The first change of the bootstrap actions switches the active and standby program store buses before the bootstrap tests. A later reentry to bootstrap causes the suspect program store to be forced out of the bootstrap screening tests, regardless of whether it has passed PSFR access tests or not. Each later reentry causes another store to be forced out of the testing. If this process of isolating each store in the community one at a time fails to isolate the cause of the problem, another reentry causes the program to go directly to the bootstrap failure leg of the program after ensuring that the processor configuration state counter is greater than zero so that a central control switch will occur. This would cause a B-level interrupt and the recovery would be controlled by PCRV.

D. Noninterrupt Level Functions

5.54 The PSFR program performs several noninterrupt level functions. The PSFR program administers all changes in the configuration or status of the program stores and program store buses. Some of these functions are also performed on interrupt level by the same routines or subroutines. However, all base level functions must conform to the base level segmenting requirements. Consequently, these functions are normally done as clients of MACP or are done as a subroutine called by other maintenance programs.

5.55 These base level functions are virtually identical to those done on the call store community by the Call Store Fault Recovery Program (CSFR). These base level functions are primarily related to hardware audits, diagnostic requests, and routine exercise tests.

5.56 The PSFR program administers audits of all program store/program store bus-related lamps and indicators located on program store frames and the MCC. The PSFR program uses subroutines located within the Master Control Console Common Control and Monitor Program (MCCM) to check or change the state of the lamps. The PSFR program assures that the lamps show the correct state as shown by PSFR status tables.

5.57 The PSFR program performs both prediagnostic and post-diagnostic functions for all diagnostic requests for the program store/program store buses. For prediagnostic functions, PSFR assures that the store to be diagnosed is available and also replaces the store with a substitute copy of the memory block. The store to be diagnosed is then configured out of the system and made available to the diagnostic program. For bus diagnosis, PSFR must assure that the system has access to a complete copy of program store over the remaining bus. This involves changing the bus routing controls in central control as well as controls in all program stores.

5.58 Post-diagnostic functions include verification of the success of the diagnostic before returning the unit to service. The PSFR program administers the return of the unit to service.

5.59 Removal or restoration to service of a store usually involves more than a simple alteration of the configuration of the hardware. In many instances, an update of a substitute store must be completed before a substitute store can be removed from service. Always, an out-of-service store must be updated to the present contents of its assigned memory block before it can be placed in service. If a substitute store had been serving in its place, the substitute must also be updated to its normal memory block, thus restoring duplication as soon as possible.

5.60 Routine exercise requests for store diagnosis (midnight routine) are also administered by routines located within PSFR.

PSFR—PROGRAM STRUCTURE

A. General

5.61 The PSFR program (Fig. 12) is divided into two pidents, PSFRCSPG and PSFRPSPG. Pident PSFRCSPG contains all PSFR program code that may be used during E-level interrupt recovery until program store memory block 0 is proven to be valid. It also contains all PSFR code that may be used when PSFR is called by PCRV. This pident PSFRCSPG is located in call store since the program store community is suspect under the above conditions. Pident PSFRPSPG contains all the remaining PSFR code and is stored in program store. It shares functional responsibility with PSFRCSPG and contains many subroutines that are used by E-level recovery. However, PSFRPSPG is not entered until program store block 0 is verified by access tests executed from the call store program.

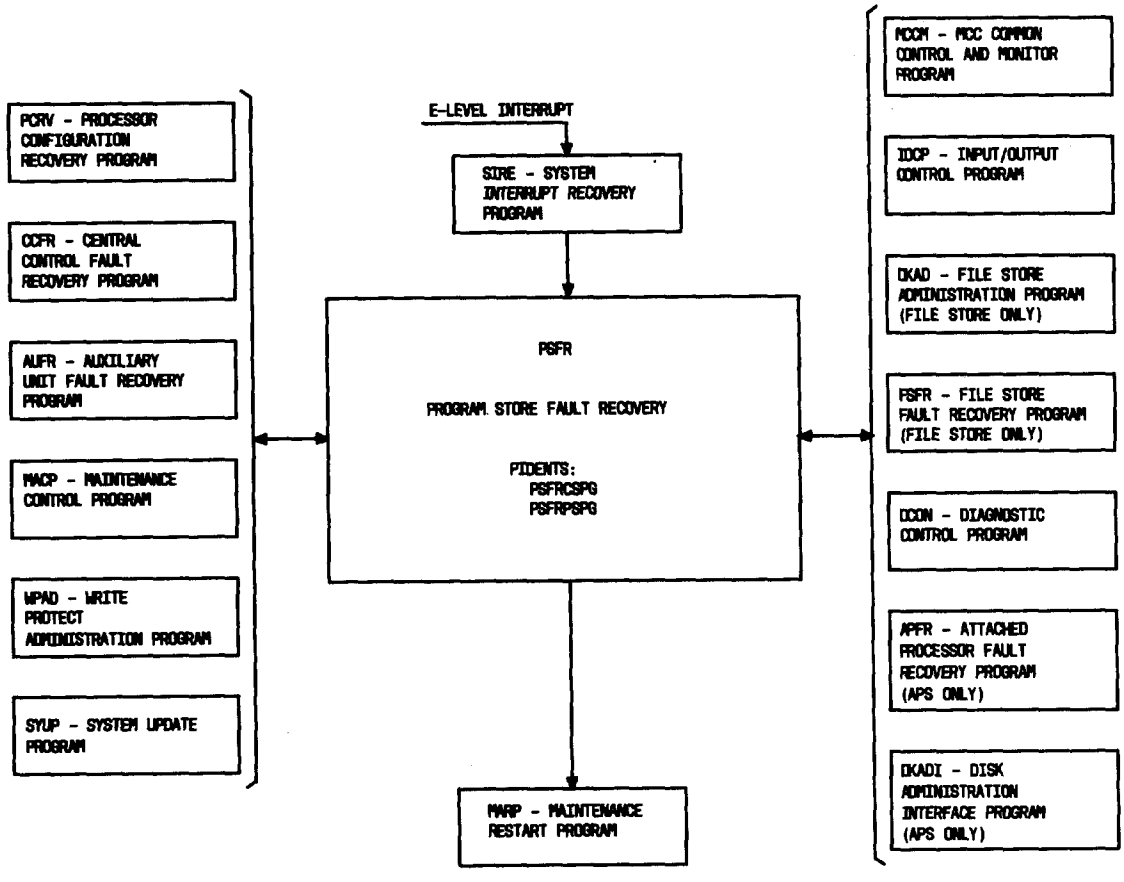
5.62 The PSFR program also uses numerous subroutines located within other programs. The following programs contain important subroutines that are used by PSFR:

(a) MACP—Subroutines are used to set up base level tasks.

(b) MCCM—Subroutines are used to verify and update the states of various lamps and control indicators on the program store frame and the MCC.

(c) IOCP—Subroutines are used to initiate output messages.

(d) DKAD—Subroutines are used to perform "pumps" (file store to program store copy) of store memory blocks. (File store system only)



♦Fig. 12—Program Store Fault Recovery Program (PSFR)—Interfaces♦

(e) ⧫DKADI—Subroutines are used to perform "pumps"of store memory blocks. (APS only)⧫

(f) AUFR—Subroutines are used to stop AUs during PSFR recovery functions.

(g) CCFR—Subroutines are used for all central control configuration changes during testing.

(h) FSFR—Subroutines are used to restore the file store controllers to their previous state after a pumping operation, if possible. (File store system only)

(i) ⧫APFR—Subroutines are used to restore the APIs to their privious states after a pumping operation. (APS only)⧫

(j) WPAD—Subroutines are used to verify a write protect error before restart.

5.63 There are several other important program interfaces in addition to the subroutine interfaces.

(a) SIRE—Program PSFR is always entered from SIRE after an E-level interrupt.

(b) PCRV—Program PSFR has several interfaces with PCRV during system recoveries, program store bootstraps, and PCRV initiated pumps of program store.

(c) DCON—Program PSFR has several exits to diagnostic control to start or stop a diagnostic of program store.

(d) SYUP—The PSFR program interfaces with the System Update Program to remove, configure, and restore program stores and program store buses for system updating.

(e) MARP—Program PSFR interfaces with MARP to set up the restart and the output data and then normally exits to restart.

5.64 The PSFR control structure and bootstrap operations were described as a part of Interrupt Recovery. A description of the remaining major routines follows.

B. Full Access Test

5.65 The full access test routine may be entered to verify a single program store block of memory or to verify the entire program store community. This routine is used by PSFR E-level recovery to verify that a complete copy of program store is available before returning to call processing via MARP. The PCRV and CCFR programs also use this routine for the same purpose.

5.66 Entry point PSFRACC is used by CCFR to call for an access test of the program store community using the existing program store configuration. For this entry, no changes are made in the configuration. Program control is returned to CCFR with a pass/fail indication on completion of the tests.

5.67 Entry point PSFRACPC is used by PCRV. The first 16 states of the processor configuration sequencer circuitry in central control try to build a valid program store configuration (without the use of file store pumps) by selecting an active program store bus and a program store containing memory block 0. This action is undertaken only under major system malfunction conditions. Later, PCRV enters the program store access tests at PSFRACPC to verify the configuration selected by the processor configuration sequencer.

5.68 First, the access tests identify the selected program store. (The store is a processor configuration store described in paragraph 5.07.) The store selected is the processor configuration store that has its maintenance flip-flop reset. The remaining program stores are then configured according to the selected active bus and memory block 0 store. Next, the access test is done and a success/failure return is made to PCRV.

5.69 For E-level recovery, the initial entry to the access tests determines whether the central controls were operating in step at the time of the interrupt. If they were, an attempt is made to start the standby in step with the active before doing the access tests. If the standby is placed in step with the active, then routine matching is established without interrupts. Also, the active central control is set to halt matching and stop the standby if a mismatch is detected. The PSFR program always uses CCFR service routines to make any required changes in the configuration of central control. The access tests are then done sequentially on each K-code assigned to the program store community or the single specified K-code for a one block verify.

5.70 If the tests pass, the following actions are taken. If the central controls were in step during

the access tests, the standby is stopped and a check is made for mismatch errors. If the central controls mismatched or the standby central control detected either an ASWF or an DPWEF during the tests, a request is made for a diagnosis of the standby central control. This disgnosis is done on base level after interrupt recovery is completed. The standby program store bus is then removed from service and the full access test is reentered. However, if no errors occurred, PSFR requests a restore of the standby central control on base level.

5.71 If the tests were performed successfully with no mismatches or with only the active central control, report data is prepared for output. The PSFR program then requests diagnostics for all units removed by PSFR, or if a bootstrap was done during this interrupt interval, a diagnostic is requested for all out-of-service program stores.

5.72 At this point, program control is returned to AUFR for program store AU access failures detected by AUFR, whereas E-level program store access failures are transferred to MARP for restart. Write protect errors are transferred to WPAD before restart.

5.73 If the access tests fail, checks are made of the number of previous failures. The results of the checks depend on whether bootstrap was entered before and whether the buses were switched. If either of these actions occurred, bootstrap is entered. If not, further analysis of the failure is done. If the failure was detected in the standby central control, the standby store is removed. If no standby store existed, the standby bus is removed. If neither standby bus nor standby store exist, bootstrap is entered. If the failure was detected in the active central control, a check is made for a Trap Refresh Data Parity Failure which if detected is corrected. After the first failure, the first fail flag is set as an indicator for possible later failures.

5.74 After the first detected failure of the active central control or bus, a switch of the active and standby bus is done followed by a removal of the standby bus.

5.75 If the first fail flag is set and if error analysis is set, a check is made to determine whether an update of a substitute rover store is in progress. If an update is in progress, a transfer is made to the FINFILL routine to complete the update on interrupt level. Later, the access tests are reentered to assure that a complete copy of program store is available after the updated rover store is placed in service.

5.76 If an update is not in progress, PSFR transfers to the program store bootstrap routines.

C. Program Store Status Update

5.77 On entry at PSFRUSTA, the program store status table located in the duplicated protected range of call store is updated. The status table is normally assumed to be correct when the program store community's configuration is altered by a routine other than bootstrap. However, the status update routine is entered to make the status table agree with the actual store community configuration when there is reason to believe that it may not be accurate, ie, after a bootstrap or if both copies of the call store memory block it resides in are lost. The routine may be requested manually or may be entered under program control, ie, from PCRV.

5.78 The correct status is determined by pulsing the maintenance flip-flop for each program store and updating the status according to the answer.

D. Program Store Configuration Change Routine

5.79 As program store buses are restored to or removed from service or are switched from serving the active central control, etc, the program stores listening to and responding to those buses must have their routing flip-flops (Tables C and D) altered to conform to the new conditions. There are several entries to the routine to accommodate the different types of changes. The configuration change routine uses two subroutines. One subroutine alters the program store status table to agree with the new bus status, whereas the other subroutine sets the program store routing flip-flops to conform to the configuration found in the status table. When all the stores conform to the status table, program control is returned to the routine or program that requested the configuration change.

5.80 This routine may be used to remove the standby bus from service, switch the active and standby buses, return the standby to service, or to establish a configuration based on the status table.

E. Program Store Removal Routine

5.81 This routine is entered to remove a selected store from service. If the store is already out

of service, the routine reports a successful removal, turns on the out-of-service lamp at the store, checks for a primary trouble condition (if necessary, the primary trouble lamp at the MCC is lighted) and returns program control to the calling program. If the store is active and no rover is available, then the removal request is denied.

5.82 If the store is duplicated, the duplicate copy is set up to operate as a nonduplicated store. The selected store is then removed from service and the request is completed as for a store that is already out of service when the request is made. If a rover store is needed and is available, it is first updated from the file store or by copying from the unit to be removed. The removal is then completed as for a duplicated store.

5.83 The routine may be entered at global entry PSFRSRMV in response to a remove (RMV) TTY request, a control panel request, or a power alarm request. Other entry points are also supplied to swap units or to remove and diagnose a store.

F. Program Store Restoral Routine

5.84 This routine is entered to restore a program store to service. It may be entered because of a restore (RST) TTY request or a call from another program. On entry, this routine must determine if the specified store is already actively supplying a memory block. If so, the store is removed and a diagnosis of the store is requested to be followed by a restoral. Secondly, if the store is not actively supplying a memory block, a diagnosis is requested asking for a restoral of the store to service after a successful diagnosis.

5.85 If an all tests pass (ATP) diagnosis has been done, the store is updated by copying from the now active store. When the update is completed, the store is configured back into service. The primary trouble lamp at the MCC is extinguished if it was lighted. A successful restoral is reported to the TTY, and program control is returned to the calling routine or program.

6. 1A PROCESSOR AUXILIARY UNIT FAULT RECOVERY PROGRAM—AUFR

INTRODUCTION

6.01 The AUFR program performs all fault recovery tasks which are common to all auxiliary units (AUs) in the AU bus (AUB) system. The AUFR fault recovery approach consists of finding a set of AU and central control hardware that is capable of carrying out the normal AU communication functions. This determination is made by a detailed set of tests which are run either on interject or interrupt level priority.

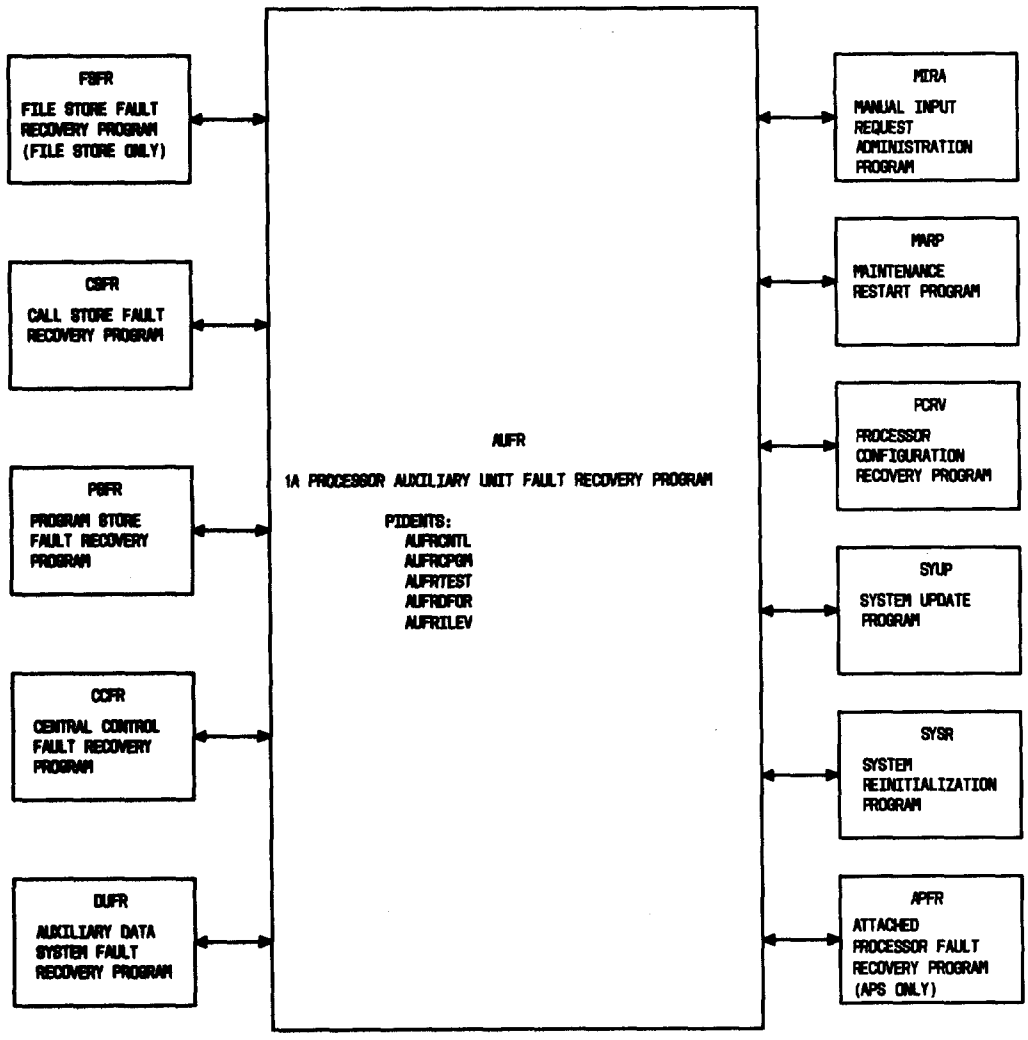
6.02 To perform its functions, AUFR interfaces with several other programs. The major AUFR program interfaces are illustrated in Fig. 13.

AUB SYSTEM ORGANIZATION

6.03 The 1A Processor System has a busy system which enables autonomous processing units to access the call store and program store bus system of the central control. The autonomous processing units are referred to as AUs and there may be as many as 16 AUs on the AUB. The AUB is linked to the call store and program store buses by special hardware in the central control that is called the AUB sequence (AUBSQ). The AUBSQ resolves bus occupancy conflicts among AUs on the AUB and resolves bus, store, or AU occupancy conflicts between the central control or any AU on either the AU, call store, or program store bus. This document will refer to the AUs and the AUBSQ as the AUB system. (See Fig. 14 and 15)

6.04 The AUB system, in the file store environment, will have at least two and a maximum of four file store controllers. A file store controller may control from one to four disk files on which a large amount of data can be stored. A file store controller is a wired logic processor that will process central control request(s) to transfer a data block between the slow serial access memory of a disk file and the fast random access memory (RAM) of a call store or program store. For reliability, the memory content of one set of disk files associated with a file store controller will be duplicated on an identical set of disk files associated with another file store controller. Collectively, all file store controllers with associated disk files are referred to as the file store system. The file store system serves as a primary data backup and bulk data storage facility for program, translation, and other information for the 1A Processor System.

6.05 The AUB system, in the Attached Processor System (APS) environment, will have at least 2 and a maximum of 16 Attached Processor Inter-



♦Fig. 13—Auxiliary Unit Fault Recovery Program (AUFR)—Interfaces♦

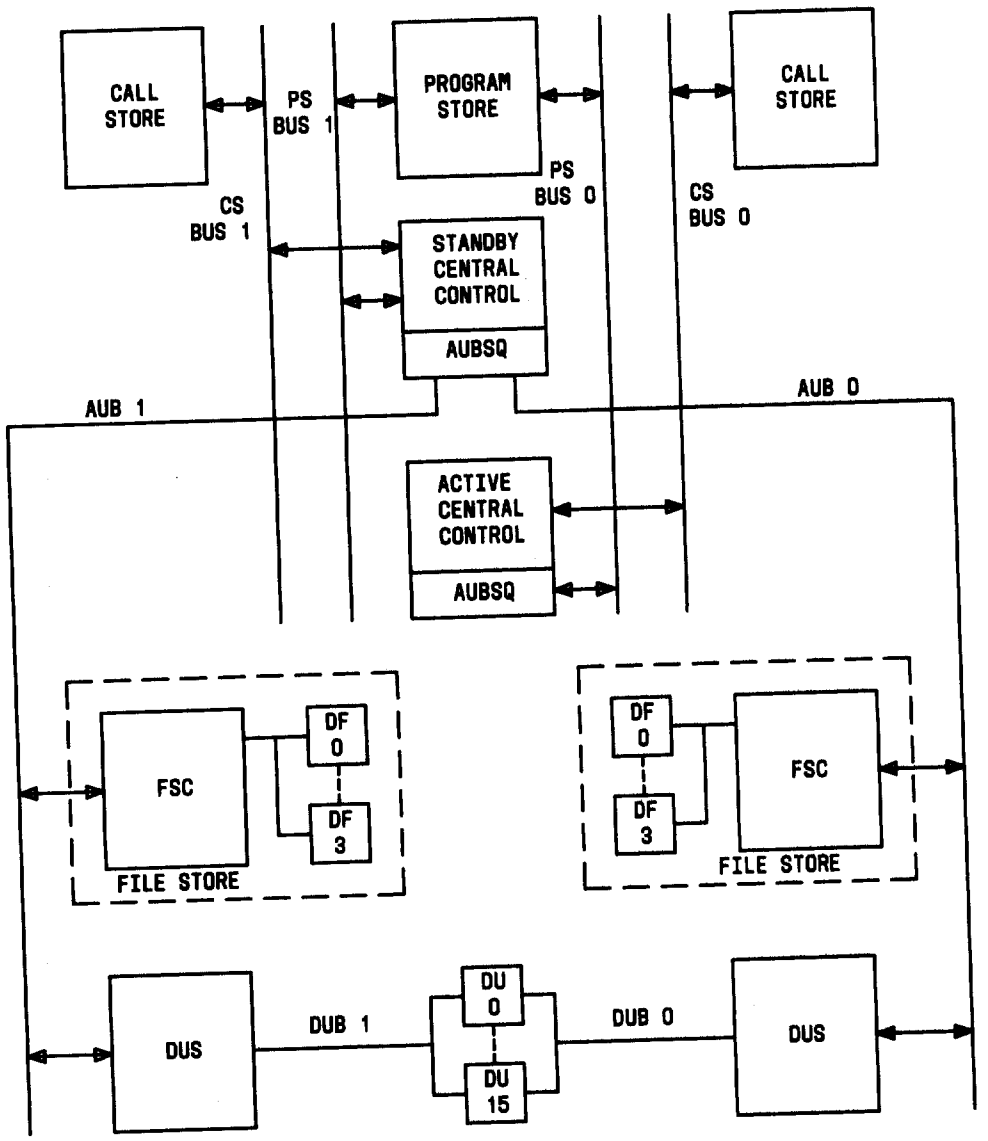
faces (APIs). The APS is a high capacity disk system for the 1A Processor. The API allows the sending and receiving of messages and blocks of data between the 1A Processor and up to eight 3B Processors.♦

6.06 The AUB system will also have at least two and a maximum of four data unit selectors (DUSs). A pair of DUSs may control from 2 to 16 data units. A DUS is similar to a file store controller except that instead of disk files, the DUS is designed to handle slower data devices such as tape units. The data unit system serves as a backup to the file store system for system reinitialization and as a primary facility for program updating, automatic message accounting data recording, and other functions.

AUFR—FUNCTIONS AND STRATEGY

A. General

6.07 The AUFR program is designed to function under several diverse conditions: interject, D-level interrupt, and other processor interrupt levels. Basically, AUFR uses the first-look approach to fault recovery. The first-look approach, as used by AUFR, involves the retrying of the failing operation using simple and fast testing techniques. If this approach fails to identify the source of the trouble, AUFR will then resort to more detailed testing to isolate the problem.



LEGEND:
FSC - FILE STORE CONTROLLER
DF - DISK FILE
DU - DATA UNIT (TAPE UNIT CONTROLLER)
DUS - DATA UNIT SELECTOR
DUB - DATA UNIT BUS
AUBSQ - AUXILIARY UNIT BUS SEQUENCER
AUB - AUXILIARY UNIT BUS

Fig. 14—1A Processor Auxiliary Unit Bus System—File Store Environment

6.08 The AUFR program can be entered under three conditions. The first condition involves maintenance action for the AUB system. This maintenance action will ordinarily be started through the interject request mechanism instead of the normal maintenance interrupt control hardware sequencer. This method is used because AU processing is independent of central control processing and can be momentarily deferred without degrading system performance.

6.09 In the second condition, the AUFR can also be entered on D level from the Call Store Fault Recovery Program (CSFR) when the central control

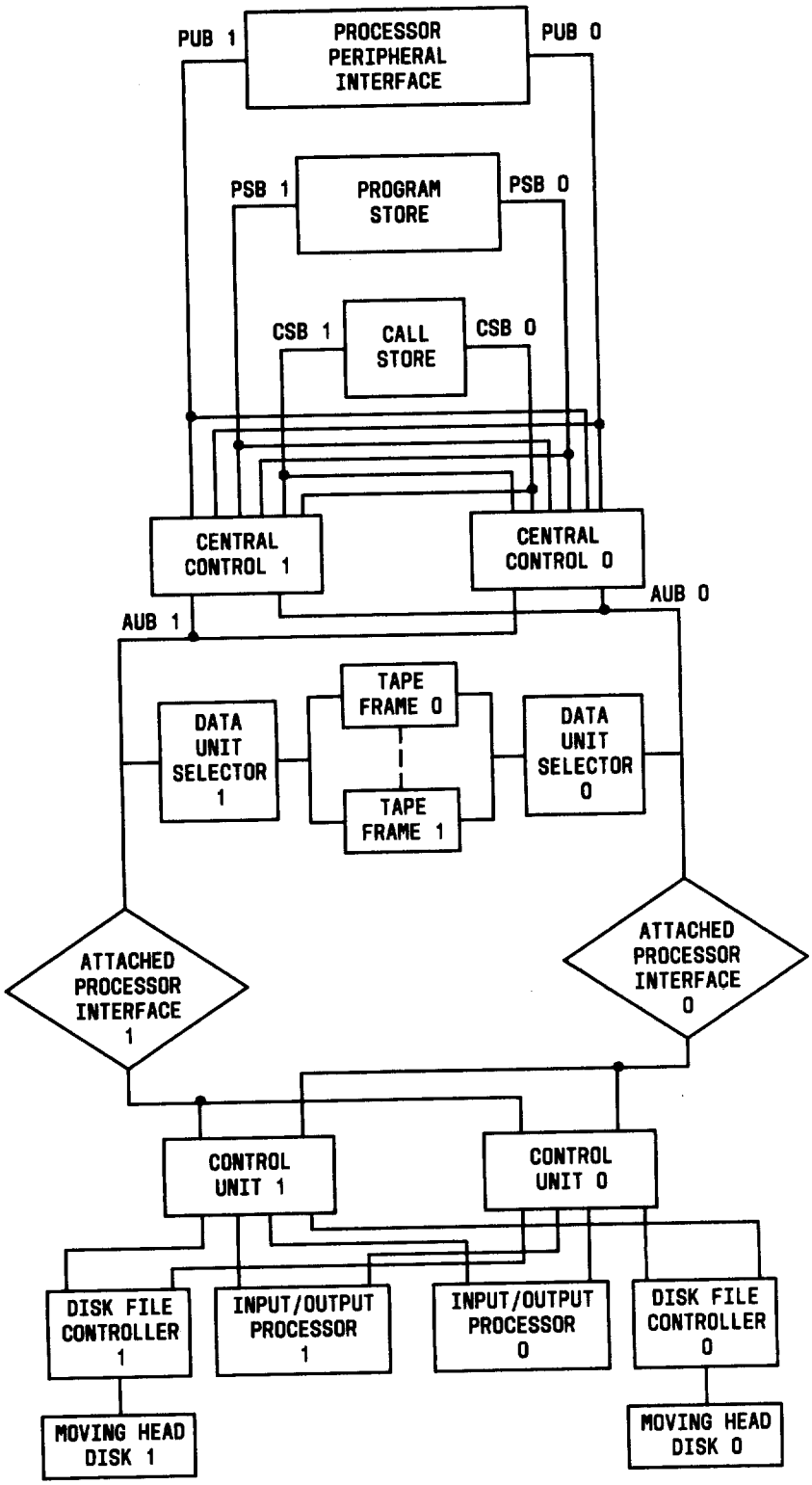


Fig. 15—IA Processor Auxiliary Unit Bus System—APS Environment

encounters an AU read/write failure. An AU read/write failure may occur when the central control addresses an AU and an accessing error is detected by either the central control or AU. The central control may also address an AU that is in a troubled state and has requested maintenance action through the interject mechanism. An AU which makes an interject request will not respond to central control addressing until it has been restored to service by the specific type of AUFR program. The AUFR program processes D-level entries basically as it processes interject entries.

6.10 Finally, AUFR may also be called by TTY request or by the processor configuration or another processor fault recovery program to test or reconstruct the AUB system interface with the central control system.

6.11 When an error indicator can be identified immediately as a faulty AU, the error source is classified as "unique." Every AU contains error detection circuits that check internal gating and other processing functions. When these circuits detect an error that involves an internal processing function, the source of the error has been isolated to the AU level. Always for unique AU faults, AUFR transfers to the specific AUFR program which then determines the proper course to follow for that particular AU.

6.12 The bulk of the AUFR programs involves the processing of central control AU "common" errors. Central control AU common errors involve situations where the error indicators in the AU do not necessarily indicate a malfunctioning AU. In some of these cases, a software error can cause an AU to appear faulty. In other cases, the AU will be communicating with other units (central control, call store, or program store) which can malfunction and cause an error to be detected by the AU error detection hardware. In all cases of central control AU common error, AUFR will analyze the error to determine the source of the trouble (AU, store, or central control). The analysis of interface errors will often require testing of the associated hardware.

B. Basic Program Strategy

6.13 When AUFR is entered on interject or on a processor interrupt level, all call processing and most maintenance interrupts are inhibited and no time breaks are taken. The AUB system is placed in a maintenance mode, and the error state of the AUBSQ and AU hardware is interpreted. As a result of the first-look approach used by AUFR, the fault recovery time for the AUB system will normally be minimal.

6.14 Most faults handled by AUFR are expected to have generated unique errors that will be handled in a straightforward way so that the AU can be immediately processed by the specific AUFR program. On the other hand, central control-AU common errors may require further testing before the faulty unit can be isolated. Since common errors usually involve the busing of data from one unit to another, the associated circuits are subject to transient errors, and the failing operation therefore will be retried. If the retry also fails, a fault will be assumed. If the retry passes, an error will be assumed, and the AUB system will be restored to service. This process quickly screens transient errors from hardware faults.

6.15 When it is determined that a unit is faulty, the unit will be removed from service. If the unit is a central control, call store, or program store, the unit will be removed (Fig. 16) by a routine in the Central Control Fault Recovery Program (CCFR), Call Store Fault Recovery Program (CSFR), or Program Store Fault Recovery Program (PSFR).

6.16 For a common AU fault indication (either file store, data unit, or API faulty), AUFR determines the faulty unit and uses the File Store Fault Recovery Program (FSFR), the Attached Processor Fault Recovery Program (APFR), or the Data Unit Fault Recovery Program (DUFR) remove routines to remove the unit from service and request a diagnosis. If there is a unique AU fault indication, AUFR goes directly to FSFR, APFR, or DUFR for testing and fault isolation (Fig. 16). If the AU cannot be removed from service because of the unavailability of the standby or mate AU (duplex failure), AUFR will perform error analysis. If too many duplex failure conditions occur, AUFR will call its bootstrap routine that will completely rebuild the AUB system.

AUFR—PROGRAM STRUCTURE

A. General

6.17 The AUFR program is divided into the following five pidents:

(a) ▶AUFRILEV—AUFR Interrupt Level (resident in program store)◀

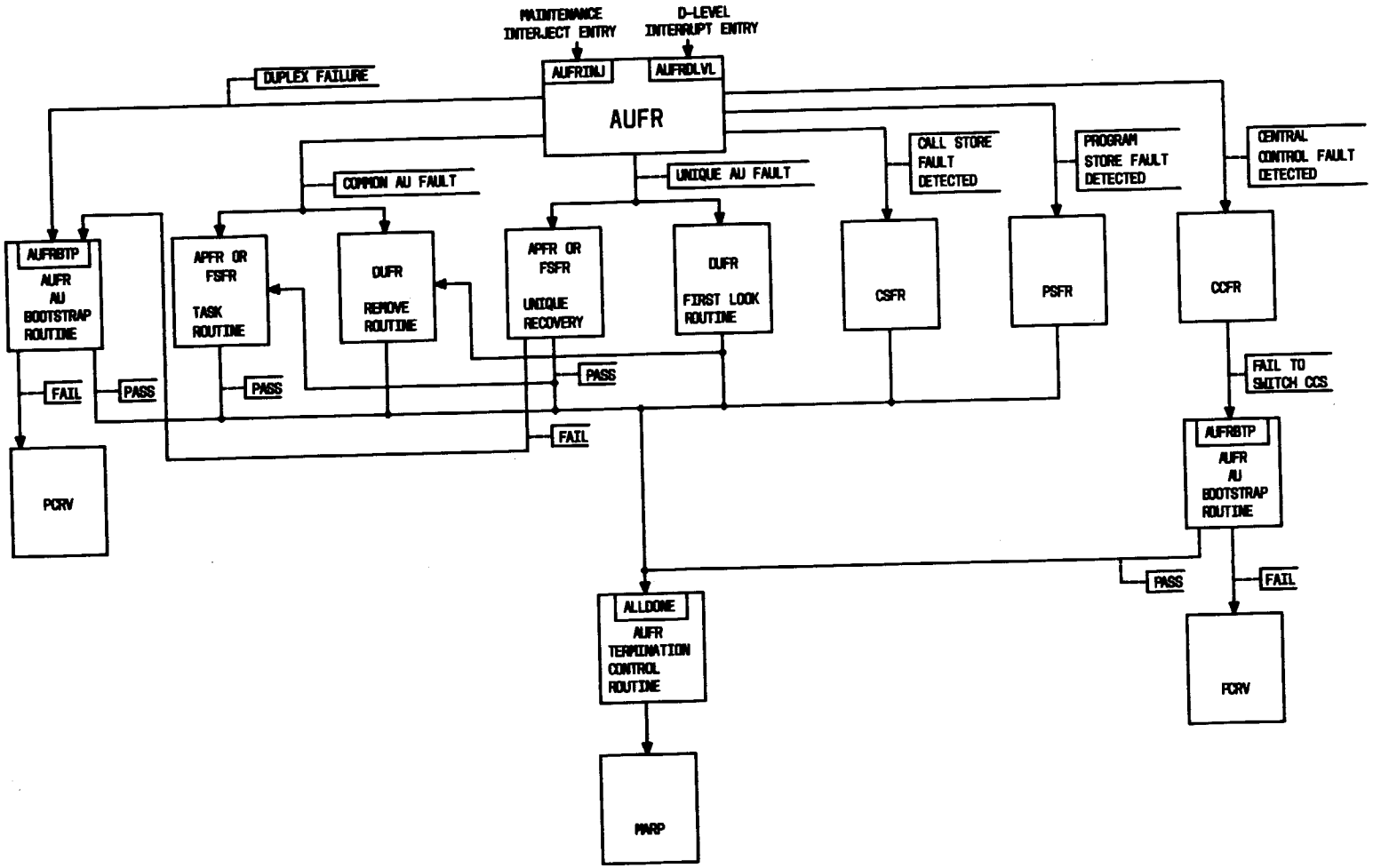


Fig. 16—Auxiliary Unit Fault Recovery (AUFR)—Overview

(b) AUFRCNTL—AUFR Control (resident in program store)

(c) AUFRCPGM—AUFR Call Store Program (resident in call store)

(d) AUFRTEST—AUFR Test (resident in program store)

(e) AUFRDFOR—AUFR Deferred Fault Recovery (resident in program store).

6.18 ▶The AUFRILEV pident performs needed fault detection tests on initial entry from interject or D-level (first look). Also AUFRILEV attempts to reproduce the fault (if in the central control or main memory access type circuitry) or passes control to an appropriate unique error handling program (FSFR, APFR, or DUFR) if the error is that type.◀

6.19 The AUFRCNTL pident contains subroutines that do tasks such as bootstrap, AU stop and start, error analysis, and translation retrieval.

6.20 The AUFRCPGM pident is made up of subroutines that do program store bus communication tests or alter the program store bus configuration.

6.21 The AUFRTEST pident contains test routines which test the AUBSQ, central control access to AUs, and AU access to call store and program store.

6.22 The AUFRDFOR pident contains all the diagnostic interfaces that concern the AUB. This pident also handles all Manual Input Request Administration Program (MIRA) interfaces and provides some service routines for manipulating the different diagnoses and MIRA inputs and forming them into a standard format.

6.23 The functional organization of AUFR consists of two main parts, fault recovery and service routines. The basic fault recovery steps and the AUFR pidents involved in their execution are as follows:

(1) ▶Establish the failing AU for testing and error analysis (AUFRILEV/AUFRCNTL).

(2) Retry the failing job using failing data (AUFRILEV/AUFRCPGM).

(3) Isolate the failing unit (if any) and remove it from service (AUFRCNTL/AUFRCPGM/AUFRTEST/AUFRILEV).

(4) Perform AU bootstrap recovery if paragraph 6.23(3) is not possible because of duplex failure (AUFRCNTL).

(5) Perform short-term error analysis if no fault found (AUFRCNTL).

(6) Return to call processing (AUFRILEV).◀

6.24 The major AUFR service routines and associated pidents include the following:

(a) Pre- and post-AUB diagnostic handlers (AUFRDFOR)

(b) TTY interfaces for RMV/RST/TEST messages (AUFRDFOR)

(c) Off-line AUB removal and restoration (AUFRDFOR)

(d) Administration of the AUB deferred fault recognition tests (AUFRDFOR)

(e) Stopping and starting of in-service AUs (AUFRCNTL)

(f) Unconditional restoral of all AUs (AUFRCNTL)

(g) Office data assembler translation (AUFRCNTL).

B. Fault Recovery

6.25 ▶Two pidents, AUFRILEV and AUFRCNTL, provide necessary routines for AU fault recovery. The pident AUFRCNTL which contains routines necessary for both PCRV and AUFR is resident in program store block 0 (K-code 20).◀

Interject Control

6.26 Pident ▶AUFRILEV◀ is entered on interject at AUFRINJ (Maintenance Interject Entry Program Unit) as the highest priority interject request. It is expected that this will be the mechanism by which most AU hardware problems will be resolved. The maintenance actions done by AUFR here

may be no different from those done by the D-level interrupt entry (paragraph 6.28). The main reason for using interject control instead of a D-level interrupt is to keep the central control program, which is probably doing work having nothing to do with AU functions, from being unnecessarily interrupted.

6.27 When any central control program uses the stack transfer with the interject option and an AU interject request is pending, the central control will take a hardware transfer to AUFRINJ. The basic reasons for an AU interject request include the following:

(a) The AUBSQ in central control detects a verify mismatch. This causes the verify mismatch source to be set which in turn sets the AU interject request flip-flop and freezes the AUBSQ hardware. The AU interject request flip-flop is cross-coupled between central controls to keep the central controls in step.

(b) An AU has detected trouble and has stopped. An interject request flip-flop will be set in the AU which in turn will pulse the central control maintenance interject lead on every 700 ns clock cycle. This will set the maintenance bus request flip-flop in both central controls which will in turn set the AU interject request flip-flop.

D-Level Control

6.28 The pident ⧫AUFRILEV⧫ is entered at AUFRDLVL (D-Level Interrupt Entry Program Unit) by CSFR when it is found that the D-level source is an AU read/write failure. There are four ways that this type of interrupt may occur:

(a) The central control program encountered access trouble with an AU and could not have continued data processing.

(b) An AU will not respond to a system program request when it has detected trouble and stopped. Therefore, any system program addressing the stopped AU with a specific K-code will be interrupted.

(c) An AU will not respond to a system program request even when the common K-code feature is used to address the AUs if both members of an AU community have detected trouble and stopped. The unsuccessful access attempt will result in an interrupt.

(d) Invalid use of maintenance access by a maintenance program could lead to an AU read/write failure. However, this is an unusual class of software programming errors that is difficult to handle and, therefore, is not processed directly by D-level control.

Software Checks

6.29 An AU read/write failure may be caused by several important software errors that must be screened out before hardware trouble can be assumed. To do this, the failing address (and sometimes data) must be known. Consequently, the interrupt sequencer saves the failing address and data on all AU read/write failures. There are no important software errors that would lead to an interject request and not also cause an AU read/write failure. Therefore, the following software checks are made only in D-level control:

(a) The AU selected for central control access is determined by the K-code (bits 11 through 14) of the failing address. If the K-code is unassigned or the associated AU is out of service, an AU read/write failure will occur. Therefore, the K-code is checked for both conditions and an invalid K-code is interpreted as a software error. Since no hardware trouble exists and the interrupted program must be audited, control is passed to MARP.

(b) The K-code may be valid, but the address (bits 0 through 10) for the AU may be invalid. The valid address for an AU depends on both the type of AU and how it is equipped. Consequently, a separate address range check routine has been written for each type of AU.

(c) Whenever a common K-code read is performed, a data parity failure may occur. For override prevention of the all-seems-well (ASW) checks, the user of a common K-code read must execute the maintenance load instructions with the read (R) and inhibit parity check (IPCK) options only. Therefore, a check for an ASW failure on a common K-code read will indicate hardware trouble. Otherwise, a software error is assumed.

Central Control Matching

6.30 When all software checks pass, a possible hardware problem is assumed. Also, when both central controls are in service, certain fault recovery problems exist.

6.31 Some forms of central control trouble with AU accesses may either generate a central control mismatch or an AU read/write failure. If both sources are generated, then either a C- or D-level interrupt will occur. The C-level interrupt has higher priority unless the D-level source precedes the C-level source by at least one 700-ns cycle. If the D-level source precedes the C-level source, a D-level interrupt will result because the interrupt sequence will stop the standby central control and will inhibit the effects of any subsequent C-level interrupt sources. Therefore, the standby central control will be stopped on all entries into the D-level control routine. However, a CCFR service routine will be called by the COMCON routine (paragraph 6.33) to initiate a special central control match mode where C levels are inhibited but the standby central control will stop on mismatch.

Common Control

6.32 The bulk of the ♦AUFRILEV♦ pident is concerned with the control function (common to both interject and D-level control). There is a primary entry and several secondary entries to common control. The primary entry (COMCON) is by interject control and D-level control to process most hardware troubles in the AUB system. Other entries have been provided for performing tasks such as terminating common control, causing a faulty unit to be removed from service, initiating more drastic recovery procedures. Some of these entries are used not only by AUFR, but are also called by CSFR, PSFR, and the Processor Configuration Recovery Program (PCRV). The following discussion of the common control function (of the AUFRILEV pident) includes the primary entry and some of the more important secondary entries.

COMCON Entry (Common Control Routine)

6.33 The common control for both interject and D-level entry program units (COMCON entry into the AUFRILEV pident) initiates an error analysis and filter routine for processing interject and D-level AUFR tasks. The routine interrogates three sources of information as follows:

(a) The AU miscellaneous group B register in central control is checked for a verify mismatch source.

(b) The maintenance interject source and all central control access dependent error sources are interrogated with the control pulse read facility.

(c) The AU error summary registers are maintenance read to determine all remaining sources of trouble.

6.34 The error analysis and filler processing begins with a check to determine if the AUB configuration flip-flops are in a valid state, since it is possible that an invalid bus configuration in central control will cause an AU interject request or AU D-level interrupt. If any check fails, a software error is assumed. If attempts to correct the error appear unsuccessful, the AUB system may be bootstrapped. (See paragraph 6.54.)

6.35 Next, all AUs are control pulse read. Examples of the functions performed by the control pulse read of an AU include the following:

(a) The AU is taken off-line by placing it in a maintenance mode.

(b) Except for the central control interface sequencer, all sequencers in the AU are either stopped or cleared.

(c) All inhibits on central control access are removed by clearing the bus trouble flip-flop in the AU. (For a DUS, the central control interface sequencer is enabled).

(d) All clock error sources are automatically cleared and their former state is returned on the read.

(e) All central control access error sources, the state of the maintenance flip-flop, and the contents of the writable K-code register (if any) are read.

(f) The interject pest flip-flop in the AU is set.

(g) The state of the interject request flip-flop is read.

6.36 All sources of trouble are determined according to a priority structure that is intended to minimize the chance of processing misleading error information. Clock errors, since they may directly cause almost all other sources of error, are processed first. Every AU contains clock error sources that monitor the working state of the central control synchronized clock circuits. If clock error sources are indicated in an AU, the sources will be control pulse

read again. If the control pulse read indicates that a clock error source is still set, then a fault is assumed. At this time, the necessary checks are made to determine if the fault is in the AU clock circuit or in the central control that is supplying the clock synchronization pulses for each AU.

6.37 The verify mismatch source is processed next. If the verify mismatch source is set in the AU miscellaneous group B register, a check is made to determine if the verify mismatch source is valid. If it is, control is then transferred to the Verify Mismatch Error Program Unit (VMMERR), which will resolve faults in the AUBSQ and the AU store access interface.

6.38 Since no other error sources may be determined if central control access trouble is present, the central control access error sources are next checked. If any central control access source is set, a transfer is made to the Central Control Access Functional Tests Program Unit (ACSERR), which resolves faults in the central control AU access interface circuitry.

6.39 The AUFR program next verifies that the AU can be accessed via the central control so that the remaining error sources can be checked. All errors up to this point were checked in the generate control pulse (GCP) response. If the AU fails the central control access, control is then transferred to ACSERR.

6.40 The AU error summary register 0 is next checked for store errors. If a store access error is found, a transfer is made to the Store Access Functional Tests Program Unit (STRERR), which resolves faults in the AU to call store and program store interface circuitry.

6.41 At this point, the remaining error sources are all unique. Consequently, control is now passed to a specific AU fault recovery program (FSFR, APFR, or DUFR).

AUFRERR Entry (No Fault Found Routine)

6.42 Pident ⧫AUFRILEV⧫ is entered at AUFRERR (No Fault Found by AUFR Program Unit) whenever the AUFR program fails to find the existence of a fault condition. Usually, this indicates a transient error condition. However, a repeated error condition would indicate that the wrong hardware tests were being applied to the failing AU. Therefore, the number of times that a no-fault condition occurs is counted at the Perform AU Error Analysis Program Unit (AUFRERX) in AUFRCNTL. If the count becomes excessive, all AUFR tests will be applied to the failing AU by transferring to AUFRCPLT (Complete Test Using all the AUFR Logical Tests Program Unit). Otherwise, if the count is acceptable, the AUB system will be restored to service by transferring to ALLDONE (Termination Control Program Unit). (See paragraph 6.59.)

FAULT Entry (Fault Control Routine)

6.43 Pident ⧫AUFRILEV⧫ is entered at FAULT (Fault Control Program Unit) when a hardware fault is detected by the interject request or D-level interrupt mechanism. When any AUFR test routine fails, a flag is set in the G register to show the failing unit and that control is transferred to FAULT. The flags are AU1RCCAF for active central control failed, AU1RCCSF for standby central control failed, AU1RSTRF for store failed, AU1RAUBF for AUB failed, and AU1RAUF for AU failed.

6.44 When the AU1RCCAF flag is set, a CCFR subroutine will be called to switch central controls. If the switch is successful, the resultant B-level interrupt will cause AUFR to lose control. However, data on the B-level printout will show that AUFR caused the switch. If the central control switch is denied for some reason, AUFRILEV will call its bootstrap routine (AUFRBTP) in AUFRCNTL. (See paragraph 6.53.) If the bootstrap operation fails, control may be passed to PCRV to initiate a processor configuration recovery. If the bootstrap is successful, control is then passed to AUFR ALLDONE.

6.45 When the AU1RCCSF flag is set, a CCFR subroutine will be called to remove the standby central control from service, and control is then transferred to ALLDONE.

6.46 When the AU1RSTRF flag is set, the store type will be determined by the contents of the F register. The F register will contain the failing address that is required by the store fault recovery programs. Also, the resultant store error sources in the central control error summary register are saved in memory for later interrogation by the store fault recovery programs.

6.47 For a faulty call store, CSFR will be called to remove the call store from service. If the call

store is successfully removed, control will be returned to AUFR at AUFRCSR, which in turn goes to ALLDONE. If the removal of the call store is unsuccessful because of complex trouble, CSFR may initiate its own recovery procedures before returning to AUFR.

6.48 The processing for a faulty program store is similar to that for the faulty call store. Here, PSFR returns control to AUFR at AUFRPSR that then transfers to ALLDONE.

6.49 When the AU1RAUBF flag is set, a request is made to remove all AUs on the specified AUB. However, before any AU is removed from service, the configuration is first tested for availability of all essential units on the other AUB by calling a routine in FSFR, APFR, or DUFR. If the test passes, all AUs on the indicated AUB will be removed from service by a subroutine in FSFR, APFR, or DUFR. Control is then transferred to ALLDONE.

6.50 When the AU1RAUF flag is set, the specified AU is to be removed from service. However, before the AU is removed from service, the availability of the mate AU and all essential subunits will be determined by calling a unique AUFR program (FSFR, APFR, or DUFR) AU availability subroutine via the AUFR type table. (The AUFR program interfaces with the other fault recovery programs via the type table in the AUFRTYPE Program Unit.) If the mate AU is available, the AU will be removed from service by an AU removal subroutine in FSFR, APFR, or DUFR.

AUFRCPLT Entry (Complete Test Routine)

6.51 Pident ⧫AUFRILEV⧫ is entered at AUFRCPLT (Complete Test Using all the AUFR Logical Test Program Unit) from the No Fault Found Routine (AUFRERR) when the number of recorded transient errors for an AU becomes excessive. The three available AUFR test routines in the AUFRTEST pident (paragraph 6.60) are applied to the failing AU indicated in the X register. The first part of the AUBSQ logical test is executed to test the AUBSQ independent of the AU. The central control access logical test is next executed to test the ability of the central control to access the AU. The AU access logical test is then executed to test the ability of the AU to access the call store and program store. Finally, the second part of the AUBSQ logical test is executed to verify the operation of the various AUBSQ blockage algorithms.

6.52 If the AU fails one of these tests, control will be transferred to FAULT to remove the faulty unit from service. If all the tests pass, control will be transferred to ALLDONE. The running of AUFRCPLT on any AU is recorded at the Perform AU Error Analysis Program Unit (AUFRERX). Consequently, if AUFRCPLT is called repeatedly for any AU, it will be assumed that the AU is at fault and the AU will be removed from service.

AUFRTEST Entry (AU Test Routine)

6.53 Pident ⧫AUFRILEV⧫ is entered at AUFRTEST (Perform All AUFR Logical Tests on All In-Service AUs Program Unit) from COMCON when the trouble could not be identified and from the PCRV program to test all AU central control/call store/program store interface circuits in the current AUB system configuration. The AU Test Routine calls the same tests and in the same sequence as does the Complete Test Routine except that the tests are repeated for all AUs marked in-service. This routine ends either on pass when all AUs have been tested or on fail when the first AU fails. When the first AU fails, the following data is placed in the appropriate central control registers:

- Reason for failure
- Identity of failing AU
- Failing test address
- Expected data
- Actual data.

AUFRBTP Entry (AU Bootstrap Routine)

6.54 Pident AUFRILEV is entered at AUFRBTP (Bootstrap All Equipped AUs Program Unit) to bring the AUB system up from ground zero. This routine performs the same test functions as the AU Test Routine but differs in the initialization and termination controls. The AU Bootstrap Routine is called by ⧫AUFRILEV⧫ and the PCRV program when there appears to be double trouble in the AUB system. Here, the existing AUB system configuration cannot be salvaged, so the configuration must be recovered from scratch.

6.55 All equipped AUs are brought into service and tested. This is necessary because AUFR tests

on the AU are not complete in that AUFR checks only the central control call store program store AU communication circuitry. When these tests pass, the AU will be brought into service and its internal checking circuits will be expected to detect any other malfunction in the AU.

6.56 When an AU fails a test, it is marked in trouble in status only. If the bootstrap routine succeeds in finding a satisfactory configuration, the units that are marked in-trouble and are equipped are removed from service and will have diagnoses requested. If the bootstrap routine fails to find a workable configuration, the last significant AU failure information will be returned to the client.

6.57 The bootstrap will first attempt to bring up both AUBs. If that fails, it will try to bring up AUB 0 first and then AUB 1 if AUB 0 fails. The routine will give up and return to the client only after all three configurations have been tried.

6.58 If AUFR was the client of the bootstrap routine, a pass return will increment an emergency action state counter that is a "last resort" measure for faults that may be undetectable by testing. If too many bootstrap pass conditions occur in an hour's time, various actions may take place. The possible actions include switching central controls, removal of AUB 0 or 1, removal of all nonessential AUs, duplex failing of units, and a transfer to PCRV. Barring any of these actions, control is returned to ALLDONE. A fail return may result in the duplex failing of AUs or a first-level processor configuration recovery.

ALLDONE Entry (Termination Control Routine)

6.59 The ♦AUFRILEV♦ common control function terminates at the Terminate the Interrupt or Interject or Base Level Actions Program Unit (ALLDONE) where a decision is made based on the entry mechanism of AUFR. If the D-level interrupt mode flag is set, control is transferred to MARP at MARPDLVL. If the interject request mode flag is set, control is transferred to MARPIJAU. If the base level maintenance flag is set, control is transferred to MARPBASE. All three of these MARP entries from AUFR will call AUFRSTRT (Start All In-Service AUs Program Unit) to restore the AUB system.

Test Routines (AUFRTEST Pident)

General

6.60 The AUFRTEST pident consists of three test routines that are called as required by the AUFRCNTL and AUFRDFOR pidents. The test routines are:

(a) AUBSQ Logical Test

(b) Central Control to AU Access Logical Test

(c) AU to Call Store and Program Store Access Logical Test.

6.61 These tests differ from comparable tests in AUFRCNTL in that they are "logical" tests while those in AUFRCNTL are considered "functional" tests. A functional test is generally a very simple and fast one that exercises a basic function of a circuit (first look). In contrast, logical tests are time-consuming ones that test the operation of a circuit in a detailed and systematic way for the presence of a hardware fault. The AUFRTEST logical tests, therefore, serve to correct the shortcomings of the simple AUFRCNTL functional tests. To keep the recovery time to a minimum, the lengthy logical tests are generally called only as a last resort to solve a particular fault situation that cannot be solved by functional testing.

AUBSQ Logical Test

6.62 The AUBSQ logical test (AUFRTBSQ entry) is a self-contained test routine that tests all parts of the AUBSQ that may lead to a verify mismatch (central control) failure or central control maintenance interrupt. The logic functions tested include the AU priority selection, AU program store blockage, AU call store blockage, AU-AU blockage, AU enable verify match, and AU blockage verify match algorithms. The central control AU control functions tested include the AUB request, AU enable, AU blockage, AU enable verify, and AU blockage verify communications.

6.63 The AUBSQ logical test routine is divided into two parts. In the first part, all logic functions are tested without an AU by using the step-by-step maintenance control feature of the AUBSQ. The second part uses either a selected AU or all available AUs to check the control functions. Fewer tests are

needed for the second part since only simple bus communication is left to be tested.

Central Control to AU Access Logical Test

6.64 The central control access logical test (AUFRTCC, CCDPE entries) is a central control interface sequencer test routine that is designed to test all AU circuits that may lead to a central control access error either in the form of an interject request or an AU D-level interrupt. Consequently, nearly all tests are done with the AU in a normal mode except for a few supplementary tests that require that the AU be in a maintenance mode. Also, it is not necessary to test maintenance access, most of the address decoder, and most of the internal register gating circuits.

6.65 The circuits tested by this routine include the central control interface sequencer and K-code match circuit, the input address and data parity checker, the output data parity generator, and the AU address, reply, and write buses. Also, in the central control, the AU address portion of the address parity generator and AUB selection circuits are tested.

AU to Call Store and Program Store Access Logical Test

6.66 The AU to call store and program store access logical test routine (AUFRTSTR entry) is a store sequencer access test that is designed to test all AU circuits that may lead to a store sequencer access error. The circuits tested by this routine include the AUBSQ steering circuit, the AU input data parity checker, the AU output data parity generator, the AU output address parity generator, the AU store sequencer, and the AU send address, reply, and write bus communications. Basically, there are three broad conditions under which these tests are done: store independent, call store bus dependent, and program store bus dependent.

6.67 ***Store Independent:*** The first test done after initializing the AUB system for store sequencer operation is of necessity independent of all store communications. The purpose of these tests is to efficiently check the output address parity generator, output data parity generator, and input data parity checker.

6.68 ***Call Store Bus Dependent:*** These tests are simple call store bus communication tests. They do not attempt to test the address and data parity circuits except for those parts that are as yet untested because of the abnormal mode of previous tests. All AU store writes are deferred until it is established that there are no problems with AU store addressing circuits.

6.69 ***Program Store Bus Dependent:*** These tests are simple program store bus communication tests, that are similar to the call store bus dependent tests. The tests check for AU faults associated with the program store address, program store reply, and program store write buses.

C. Service Routines

Pre- and Post-AUB Diagnostic Handlers

6.70 Two service routines, AUFRDGN and AUBDGNRTN, are provided in the AUFRDFOR pident to perform tasks required before and after the AUB diagnoses are run.

AUFRDGN—AUB Prediagnostic Handler

6.71 The AUFRDGN routine does all the necessary checks before allowing the AUB diagnoses to run. Also, any bus removal routines that may be needed will be called from this routine. (However, the AUB diagnoses can run on either in-service or out-of-service buses.)

AUBDGNRTN—AUB Post-Diagnostic Handler

6.72 The AUBDGNRTN routine determines the final status of the AUB before returning it to the system. This routine may or may not restore the AUB to service, depending on the diagnostic results and other status indications.

TTY Interfaces for RMV/RST/TEST Messages

6.73 The AUFRDFOR pident service routines, AUFRMRQB and AUBDFOR, perform interface functions for remove, restore, and test messages.

AUFRMRQB—MIRA Interface Routines

6.74 The AUFRMRQB routine performs manual fault recovery functions that are requested either by TTY or alarm scan (MIRA program). The functions performed by this routine include the re-

moval and restoration of an AUB. The AU deferred fault recovery test routine (AUFRDEFR, see paragraph 6.82) also uses a segment of this routine to set up its abort address. In addition, other AUFRDFOR pident routines use segments of this routine to do services such as the printing of the restore message.

AUBDFOR—Deferred Fault Recovery Control Routine

6.75 The function of the AUBDFOR routine is to administer the pass, fail, and no tests run returns from the AU deferred fault recovery test routine (AUFRDEFR). The specific tasks to be done by this routine depend on several variables such as the type of input request and the test result. The AUBDFOR routine uses vector tables to determine the proper action to be taken. This routine also determines if the AUB is in service since the deferred tests are only run on an in-service AUB.

Off-Line AUB Removal and Restoration

6.76 The AUFRDFOR pident contains five service routines that perform tasks associated with off-line AUB removal and restoration. They are:

- AUFROSTP
- AUFRBAVL
- AUFRORMV
- AUFROPAB
- AUFRORST.

All these routines use specific fault recovery programs (FSFR, APFR, and DUFR) for unique operations. Interface with the specific fault recovery programs is via the AUFRTYPE Program Unit mentioned earlier. These routines are also used by AUFRMRQB (paragraph 6.74) to remove or restore a bus.

AUFROSTP—Terminate Off-Line Configuration (AUB)

6.77 The AUFROSTP routine terminates the off-line configuration and requests diagnoses on all units that were in service before the off-line configuration. (See AUFRORST, paragraph 6.81.)

AUFRBAVL—AUB Mate Availability Check

6.78 The AUFRBAVL routine checks for the availability of a mate bus for in-service operation. This operation ensures that the AUs on the mate bus are available for service before removing an AUB and placing it off-line.

AUFRORMV—Deferred Removal of AUB

6.79 The AUFRORMV routine removes the AUB and places it off-line if requested. The routine proceeds by first checking to ensure that all units are in service on the other AUB. Next, each unit on the designated bus is removed, and then the bus is removed. Finally, the AUB is marked off-line if the off-line flag is set. (A flag is set for any in-service AU that is taken out for off-line so that a diagnostic will be requested when the off-line restoration routine is called.)

AUFROPAB—Permit Access to Off-Line AUB

6.80 The AUFROPAB routine establishes bus routing from the central control to an off-line bus and disables the active bus. Basically, the use of this routine allows a client access to an off-line bus. The routine first resets any trouble flip-flops on the off-line AUB and then sets the trouble flip-flops on the active bus.

AUFRORST—Deferred Restoration of Off-Line AUB

6.81 The AUFRORST function is to restore the off-line AUB to service and to request diagnostics for all AUs that were in service prior to the off-line configuration.

Administration of the AUB Deferred Fault Recovery Tests

6.82 The administration of the AUB deferred fault recovery tests is performed by the AUFRDEFR routine in the AUFRDFOR pident.

AUFRDEFR—Administer AU Deferred Testing

6.83 The function of the AUFRDEFR routine is to serve as the interface between the AUFR program fault recovery tests (that are normally run on interrupt or interject levels) and the base level programs. An example of this interface function is the

segmentation (as required) of the fault recovery tests (AUFRTEST pident).

Stopping and Starting of In-Service AUs

6.84 The AUFRCNTL pident provides two complementary stop/start service routines that stop and start all AUs in the AUB system. The routines are AUFRSTOP and AUFRSTRT.

AUFRSTOP—Stop All In-Service AUs

6.85 The AUFRSTOP routine stops all in-service AUs immediately and unconditionally. This routine and the complementary AUFRSTRT are for use by interrupt/interject or by other maintenance programs where AU stoppage is mandatory. In addition to stopping the AUs, the AUBSQ is stopped and latched by this routine.

AUFRSTRT—Start All In-Service AUs

6.86 The AUFRSTRT routine starts up the AUBSQ and all currently in-service AUs. The registers in the AUBSQ are restored according to their former states or from last-look words. A special exception to the starting of all in-service AUs exists when the AUs have not been maintenance stopped. Here, only the AUBSQ is restarted.

6.87 In the case where an AU bootstrap has occurred, the Writable Store Audit Program (SAWS) will be requested to correct any out-of-date file store information that may exist as a result of the bootstrap bringing an out-of-service file store into service.

Unconditional Restoration of All AUs

6.88 The unconditional restoration of all essential AUs is performed by the AUFRREST routine in the AUFRCNTL pident.

AUFRREST—Unconditional Restoration of All Equipped AUs

6.89 The AUFRREST routine is a special purpose routine used primarily by PCRV. When entered, this routine immediately and unconditionally restores all essential AUs to service. Any AU that is powered down or for some reason fails to respond to a GCP, will be removed and a diagnosis requested. The routine provides various services such as stopping and clearing the AUBSQ, the removal from service of all nonessential AUs, and MCC lamp administration.

Office Data Assembler Translation

6.90 The AUFR program contains its own translation routines. The translation routines that are resident in the AUFRCNTL pident use translation table XL1AUKTRANS to perform their functions. The routines are:

- AUFRCKUM
- AUFRCUMK
- AUFRCPK
- AUFRCKP.

AUFRCKUM—K-Code to Unit Type Conversion

6.91 The AUFRCKUM routine converts an AU K-code to its unit type and member number.

AUFRCUMK—Unit Type to K-Code Conversion

6.92 The AUFRCUMK routine converts a unit type and member number to the AU K-code.

AUFRCPK—AUB Position to K-Code Conversion

6.93 The AUFRCPK routine converts the AU bus position to a K-code.

AUFRCKP—K-Code to AUB Position Conversion

6.94 The AUFRCKP routine converts a K-code to an AUB position.

7. FILE STORE FAULT RECOVERY PROGRAM—FSFR

INTRODUCTION

7.01 The FSFR program performs fault recovery tasks for faults occurring in a file store controller or a disk file. Program FSFR interfaces with the Auxiliary Unit Fault Recovery Program (AUFR) for processing of file store related errors that cause

a D-level interrupt or an interject. In addition, FSFR interfaces with the File Store Administration Program (DKAD) for processing status failure reports. The status failure reports are processed on base level because the error conditions are not believed to be severe enough to cause a D-level interrupt or an interject.

7.02 To perform its functions, FSFR interfaces with several other programs. The major FSFR interfaces are illustrated in Fig. 17.

FILE STORE ORGANIZATION

7.03 The 1A Processor uses a file store memory to provide backup storage for program and translation data. The file store is also used to store programs and data that are infrequently used and consequently are not normally kept in program or call store. The disk memory used by the file store possesses serial rather than random access characteristics. Because of its serial character, the time required to retrieve or store data from the file store is variable (a function of the position of the disk when the request is made). Because time required to retrieve or store data is on the order of milliseconds, it is not practical for the central control to directly access disk memory. Instead, a file store controller is provided to perform this function. The file store controller is a special purpose wired logic processor that buffers requests from the central control to read from or write into disk memory and transfers information from disk to main memory or from main memory to disk.

7.04 Each disk file has a storage capacity of 640,000 24-bit words. Each file store contains one to four disk files. File stores are arranged in pairs, and each pair is referred to as a community. The 1A Processor software is designed to accommodate a maximum of two communities. File store 0 (on bus 0) and file store 1 (on bus 1) make up one community; file store 2 (on bus 0) and file store 3 (on bus 1) make up the other community.

7.05 For more detailed information about the files store, refer to the following sections.

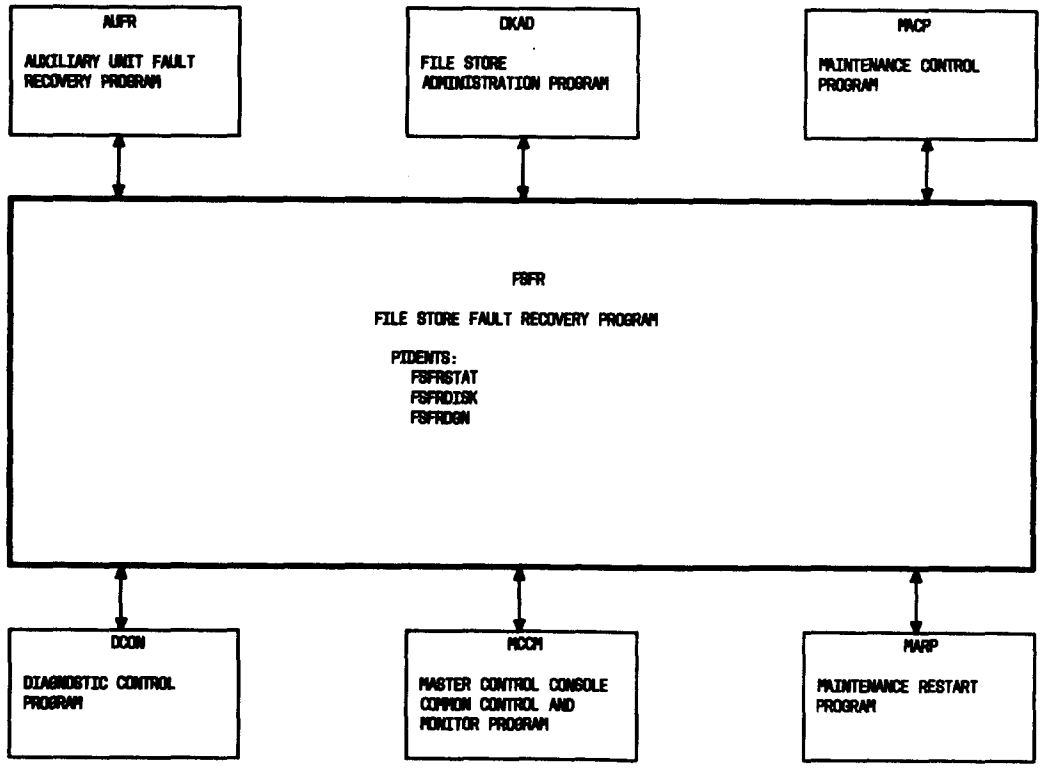


Fig. 17—File Store Fault Recovery Program (FSFR)—Interfaces

SECTION	TITLE
254-201-020	File Store Frame—Description
254-201-021	File Store Frame—Theory

FSFR—FUNCTIONS AND STRATEGY

A. General

7.06 Program FSFR resolves error conditions that occur in the file store that cause maintenance interject or D-level interrupt. Program AUFR determines that the fault belongs to a file store and transfers via a transfer table to FSFR. Now, FSFR must determine the error source (file store controller or disk file that caused the error) and the appropriate action to take. In addition, FSFR processes the disk status failure reports requested by the DKAD program. The status reports are job related and are processed on base level maintenance time.

7.07 The configuration routines of FSFR handle requests for removal and unconditional restoral of file stores and disk files. In addition, FSFR contains several service routines used in processing file store controller and disk file faults.

B. Basic Program Strategy

7.08 When FSFR is entered from AUFR or DKAD, FSFR determines the error type, increments the counter, and checks to see if the counter limit has been reached. If the counter had not reached its limit, the error is recorded, and a return to the calling program is executed. If the counter has reached its limit, the file controller or disk file is removed from service.

7.09 Whenever a file store controller or disk file is removed from service, a diagnosis is requested. Because the removal of a file store controller from service could mean that as many as four disk files would be inaccessible, every effort will be made to leave in service as many disk files as possible. Therefore, when the source of the trouble may be either the file store controller or disk file, only the disk file will be removed from service. If it is determined that the disk file only contains an error-prone record, the record will be rewritten and verified instead of immediately requesting a diagnosis.

7.10 Because the read or write of a disk file record is a relatively time-consuming process, all FSFR maintenance actions that require a disk file access operation are deferred. All disk file access operations are processed as a standard job request through the normal DKAD routines. Furthermore, those disk file related error sources that are expected to have a relatively high rate of occurrence are processed through the status failure report mechanism. Consequently, they will not require the more time-consuming and service-affecting actions of normal maintenance procedures.

FSFR—PROGRAM STRUCTURE

A. General

7.11 The FSFR program contains the following three pidents: FSFRSTAT, FSFRDISK, and FSFRDGN.

7.12 The FSFRSTAT pident processes file store controller internal error sources as requested by AUFR during an auxiliary unit (AU) maintenance interject or a D-level interrupt. The FSFRSTAT pident also processes the status failure reports that are received from DKAD. In addition, FSFRSTAT contains subroutines that do tasks such as short-term error analysis and an interface to the AU bootstrap routine.

7.13 The FSFRDISK pident contains subroutines that perform the file store stop and start functions, maintenance pump initializing and restoring, file store controller and disk file removal and unconditional restoral, and status updates.

7.14 The FSFRDGN pident contains the diagnostic interfaces associated with the file stores. The FSFRDGN pident also contains the IO interface and file store copy routine. In addition, it administers the file store deferred fault recovery test.

7.15 The functional organization of FSFR consists of three main parts: fault recovery, service routines, and configuration routines. The basic fault recovery steps and the FSFR pidents involved in their execution are as follows:

(a) Establish the type of error, using a priority structure analysis, based on certain errors causing other errors (FSFRSTAT).

(b) Remove the faulty unit or correct the error (whichever is appropriate) and/or perform

short-term error analysis (FSFRSTAT/ FSFRDISK).

(c) Call AU bootstrap if paragraph 7.15(b) is not possible because of a duplex failure (FSFRSTAT).

7.16 The FSFR service routines and associated pidents include the following:

(a) Pre-diagnostic and post-diagnostic handlers for file stores (FSFRDGN)

(b) File store copy function (FSFRDGN)

(c) TTY interface for RMV/RST/TEST messages (FSFRSTAT)

(d) Off-line file store removal and restoral (FSFRDGN)

(e) Administration of the file store deferred fault recovery test (FSFRDGN)

(f) File store stop and start routines (FSFRDISK)

(g) Initializing and restoring the file store for and from a maintenance pump (FSFRDISK)

(h) Status update routines (FSFRDISK).

7.17 The FSFR configuration routines and associated pident include the following:

(a) Removal of file stores and disk files (FSFRDISK)

(b) Unconditional restoral of file stores and disk files (FSFRDISK).

B. Fault Recovery

Disk Fault Recovery Control

7.18 The FSFRDISK pident is entered at FSFRJOB (job transfer table) to process a fault recovery request from AUFR. The FSFRJOB entry point uses an index to determine the task requested by AUFR and to pass control to the specified routine.

FSFRCCON Entry (Unique Fault Routine)

7.19 The FSFRSTAT pident is entered at FSFRCCON for processing a unique file store fault. The FSFRSTAT pident is used to determine the error type based on a priority scheme (eg, controller errors have a higher priority than disk file errors, and exerciser errors have a higher priority than time out errors). After determining the error type and the disk file or file store controller causing the error, FSFRSTAT increments the appropriate counter. The counter is checked to determine if an excessive number of errors have occurred. If the error count is not excessive, the fault is reported and control is returned to AUFR.

> ***Note:*** If the fault occurs for a disk file operating in the update mode or if a heads-out indication exists, then the disk file is removed and diagnosed.

7.20 If the error count has exceeded an acceptable level, then the file store controller or disk file is removed from service and a diagnostic is requested.

FSFRSTAT Entry (Status Failure Routine)

7.21 The FSFRSTAT pident is entered at the FSFRSTAT entry for processing status failure reports from DKAD. The DKAD program will rely on FSFR to determine the nature of the trouble and to do whatever maintenance actions are required. The DKAD program calls FSFR when disk jobs are being dispensed and either of the file store controller job aborted bits is set in word 3 of the job's disk request buffer. The DKAD program provides the following information for FSFR:

(a) Address of word 0 of the disk request buffer

(b) File store community

(c) Associated disk request register number.

7.22 The FSFRSTAT pident uses status information in word 3 of the disk request buffer to determine the type of trouble. The base K-code of the failing file store controller is available. The failing file store controller is determined from a flag passed by DKAD that shows the disk community and status information in the disk request buffer that indicates that member of the community failed. There are basically five sources of trouble reported as status failures.

7.23 ***TRAMM Error Source:*** The translated address mismatch source (TRAMM) is set be-

cause the number of words or the disk starting address for a write request was not a multiple of 32 or the disk information in the disk request register and word 1 of the disk reguest buffer are inconsistent. Therefore, the following checks are made and, depending on the outcome, the job may be either retried at DKADRTRY or aborted at DKADSFWR.

(a) When a write is specified, the number of words and the disk starting address must be a multiple of 32 in the disk request buffer.

(b) The read/write bit in the disk request register must match that specified in the disk request buffer.

(c) The disk, face, and sector information in the disk request register must match that specified in the disk request buffer.

If all the above checks pass, then a file store controller hardware malfunction is assumed and the file store controller is removed from service if an excessive number of errors has occurred.

7.24 ***Disk Access Error:*** The identification tag mismatch (IDMM) or untranslated record address mismatch (UTRAMM) source is set. The two conditions are recognized by the following:

(a) If IDMM=1 and UTRAMM=0, then an invalid disk data situation has occurred. The store identification that is returned in word 3 of the disk request buffer is verified using subroutine DKADVFID. A verification failure shows that the data stored at that location is invalid. Therefore, the record will be rewritten and the job may be retried on the other disk file by making a return at DKADRTRY. Otherwise, a verification success indicates an improper identification submitted by the client, and the job will be aborted at DKADSFWR.

(b) If UTRAMM=1, then an access error is assumed to have occurred. An access error may be the result of a data, addressing, or translator error. Therefore, the error producing record is rewritten. The FSFR program will use the copy routine to rewrite and verify the record. This action will cause any other lower priority copy operations such as a disk update to be preempted because the detection of this error is service-affecting. A rewrite is not attempted if a rewrite is already in progress. If errors on a disk file become excessive, or the rewritten record fails to verify, the disk file will be removed from service for a diagnosis. The job is retried on the other disk file at DKADRTRY.

7.25 ***BUFOFL Source:*** The buffer overflow source (BUFOFL) is set to indicate that the file store controller encountered bus or store blockage that causes the job in progress to be aborted. The file store controller will return the approximate file record that was the last completed record before ending the job. On long jobs, it is reasonable to pick up the job from the last completed record. Excessive blockage or an improper return of the aborted file record would indicate hardware trouble and result in the removal of the file store controller. Therefore, blockage due to a buffer register 1 overflow on a read, buffer register 1 underflow on a write, or a late job indicator is counted. The job will be retried from the beginning or where is left off at DKADBOFL. If the job was aborted for none of the above reasons, the error is counted and the job is retried at DKADRTRY. The unit will be removed from service and diagnosed if the error rate becomes excessive.

7.26 ***DFOFL Source:*** The disk file overflow source (DFOFL) is set indicating that a client attempted to read or write past the last record on face 1 of the disk file. Several checks are made to determine if the indication is legitimate. The partial record address returned in the disk request buffer is used to confirm that a disk boundary had been straddled. An attempt to access beyond the last record of the last available file will be treated as a failure, and the job will be aborted at DKADSFWR. Finally, a check is made on the contents of the disk request buffer to confirm that the request would straddle a disk boundary. The file store controller will be removed from service if trouble is detected. The job will be continued onto the next disk file at DKADBOFL. If the job was aborted for none of the above reasons, the error is counted and the job is retried at DKADRTRY. The unit will be removed from service and diagnosed if the error rate becomes excessive.

7.27 If the job was aborted for none of the above reasons, the error is counted and the job is retried at DKADRTRY. The unit will be removed from service and diagnosed if the error rate becomes excessive.

7.28 ***Time Out/No Status Entries:*** The FSFRSTAT pident is entered at FSFRTONS

to perform error analysis of time out conditions requested by the System Audit for File Store Administration Program (SADK) or no status conditions requested by the File Store Administration Program (DKAD). If error analysis determines that hardware is at fault, and if excessive errors have occurred, the file store controller will be removed from service and diagnosed. If it is determined that excessive software errors have occurred, FSFR requests AUFR to do a status update of the AU system.

7.29 ***AU Bootstrap Call:*** If a disk file or file store controller has an excessive number of errors and removal of the unit is not possible because of a duplex failure, FSFR will call the AU bootstrap routine at AUFRDTBL.

C. Service Routines

Pre- and Post-File Store Diagnostic Handlers

7.30 The FSFRDGN pident service routines FSFRDGN and FSDGNRTN perform tasks that are required before and after file store diagnostics are run.

7.31 ***FSFRDGN—File Store Prediagnostic Handler:*** The FSFRDGN routine performs checks to assure that the file store is available for a diagnostic and calls the file store removal routines to remove the unit from service.

7.32 ***FSDGNRTN—File Store Post-Diagnostic Handler:*** The FSDGNRTN routine performs an analysis of the diagnostic results and determines the disposition of the unit. If the file store is to be restored to service, it is updated to agree with its mate.

File Store Copy Routine

7.33 The FSFR program has three applications that require one or more continuous records on a destination file be rewritten with that stored on the source file. These applications arise when the disk exerciser finds a bad record causing an interject request, a job request encounters a bad record causing a status failure, or one or more disk files on a file store controller are to be updated and restored to service. The FSFRCOPY routine is a common service routine that is used for these specific applications.

7.34 The FSFRCOPY routine uses an interlocking mechanism between copy requests and all system write requests. Basically, all job requests are permitted, but if any write requests interfere with a copy segment already in progress, the job will be locked out as if no disk request registers were available. Alternatively, if a copy segment is to be initiated, but it is found that one or more write requests could invalidate the copy operation, FSFRCOPY will wait until these requests are completed. To ensure a reasonable wait period, FSFRCOPY will lock out any new write request that may interfere until the copy segment can be initiated and completed.

TTY Interface for RMV/RST/TEST Messages

7.35 The FSFRDGN pident service routines FSFRMRQS and FSFRDFOR perform interface functions for remove, restore, and test messages. These routines perform manual fault recovery functions that are requested by TTY or alarm scan. The FSFRMRQS routine provides an interface for removal and restoration of a file store controller or disk file. The FSFRDFOR routine provides an interface for deferred testing of file stores.

Off-line File Store Removal and Restoration

7.36 The FSFRSTAT pident provides service routines that perform tasks associated with off-line file store removal and restoration. The routines are FSFRORMV and FSFRORST. In addition, the FSFRDISK pident provides the FSFRDAVL routine for use with this service.

7.37 ***FSFRDAVL—File Store Mate Availability Check:*** The FSFRDAVL routine checks for the availability of a mate file store and its associated disk files. This operation ensures that the mate file store is available for normal service before removing a file store and placing it off-line.

7.38 ***FSFRORMV—Deferred Removal of File Store:*** The FSFRORMV routine removes a file store from service for AUFR when an off-line configuration is being set up.

7.39 ***FSFRORST—Deferred Restoration of Off-Line File Store:*** The FSFRORST routine restores the file store to the state that it was prior to the off-line configuration.

Administration of the File Store Deferred Fault Recovery Test

7.40 The FSDFOR routine of the FSFRDGN pident administers the deferred fault recovery test for file stores.

7.41 ***FSDFOR—Administer File Store Deferred Testing:*** The FSDFOR routine is a control routine for the file store deferred fault recovery test. It will call AUFR at AUFRDEFR to perform the same tests as the AUB deferred test, except the tests are done on the file store only.

Stopping and Starting of In-Service File Stores

7.42 The FSFRDISK pident provides the stop/start service routines that stop and start the specified file store. The routines are FSFRDSTP and FSFRDSRT.

7.43 ***FSFRDSTP—Stop In-Service File Store:*** The FSFRDSTP routine immediately stops the file store specified by the K-code. This action stops all sequencers in file store and sets the maintenance flip-flop.

7.44 ***FSFRDSRT—Start In-Service File Store:*** The FSFRDSRT routine starts the file store and initializes it for normal operation. The K-code of the file store to be started is specified.

Initializing and Restoring the File Store For and From a Maintenance Pump

7.45 The FSFRDISK pident provides two service routines for initializing and restoring the file store for a maintenance pump of main memory. The two routines are FSFRIMP and FSFRRMP.

7.46 ***FSFRIMP—Initialize File Store For Maintenance Pump:*** The FSFRIMP routine initializes the file store system for a maintenance pump of main memory according to the equipage and status of the file store system. Each file store controller is set up to report failing status for all disk file related failures and maintenance interject for all file store controller related failures.

7.47 ***FSFRRMP—Restore File Store After Maintenance Pump:*** The FSFRRMP routine restores the file store system to normal service after a maintenance pump of main memory.

Status Update of File Store

7.48 The FSFRDISK pident provides the FSCSUPD routine that is used by the Processor Configuration Recovery Program (PCRV). After processor configuration has pumped program store 0, it calls FSCSUPD to determine the status of the file store system. The file store controller status is updated according to equipage information and the state of the file store hardware on a community basis.

D. Configuration Routines

General

7.49 The FSFRDISK pident contains six configuration routines that perform the task of removal and unconditional restoration of file stores and disk files. The routines are:

(a) FSFRDRMV—File store controller removal routine

(b) FSFRFRMW—Disk file removal routine

(c) FSFRDRST—File store controller restoration routine

(d) FSFRFRST—Disk file restoration routine

(e) FSFRPERX—Initiate duplex file store fail error analysis

(f) FSFRPLEX—Duplex fail file store or disk file.

Removal of File Store Controller

7.50 The file store controller removal routine, FSFRDRMV, removes from service the file store controller indicated. The following actions are done:

(1) The file store controller is control pulse read. This action stops the file store controller, places it in a maintenance mode, and sets the interject prevent error source transmission (PEST) flip-flop.

(2) All disk files associated with the file store controller are taken out of service by setting their out-of-service flip-flops.

(3) Status for the file store controller and associated disk files is updated.

(4) The file store controller hardware is normalized by clearing all error sources and resetting the sequencers.

(5) The disk request buffer is audited to reflect the removal of the file store controller.

(6) The out-of-service lamp at the frame and master control console (MCC) is lighted and the file store diagnostic program is called.

Removal of Disk File

7.51 The disk file removal routine, FSFRFRMV, removes from service the indicated disk file. The following actions are done:

(1) The disk file is taken out of service and disk file status is updated.

(2) The disk request buffer is updated to reflect the removal of the disk file.

(3) The out-of-service lamp at the frame and MCC is lighted and the file store diagnostic program is called.

Unconditional Restoration of File Store Controller

7.52 The file store controller restoration routine, FSFRDRST, reinitializes the indicated file store controller. The following actions are done:

(1) All disk files are returned to service unless the heads fail to fly.

(2) Status for the file store controller and associated disk files is updated.

(3) The out-of-service lamp at the frame and MCC is extinguished.

(4) The file store controller is initialized.

Unconditional Restoration of Disk File

7.53 The disk file restoration routine, FSFRFRST, restores to service the indicated disk file. The following actions are done:

(1) The disk file is restored to service unless the heads fail to fly.

(2) Status for the disk file is updated.

(3) The out-of-service lamp at the frame and MCC is extinguished.

Initiate Duplex File Store Fail Error Analysis

7.54 The duplex file store fail error analysis routine, FSFRPERX, is entered by AUFR when an AU bootstrap has been requested because of failing file stores. Duplex file store fail analysis begins unless duplex disk file error analysis is already in progress.

Duplex Fail File Store or Disk File Routine

7.55 The duplex fail file store or disk file routine, FSFRPLEX, is entered by AUFR when a file store or disk file has caused an AU bootstrap and the bootstrap either failed or too many pass conditions have occurred. The file store or disk file and its mate will be removed from service and the duplex fail flag marked for DKAD.

8. ▶ATTACHED PROCESSOR SYSTEM SINGLE STRATEGY FAULT RECOVERY—SSFR

INTRODUCTION

8.01 Fault recovery of 1A Processor subsystems has traditionally been handled by a single program for each subsystem. However, with the addition of the Attached Processor System (APS), fault recovery is handled by two programs with a common recovery control. Fault recovery in the APS is divided into two major categories. The first category is fault recovery on interrupt or interject level and is handled by either the Auxiliary Unit Fault Recovery Program (AUFR) or the Attached Processor Fault Recovery Program (APFR). The second is fault recovery on base level that is handled by the APFR. The size and complexity of a fault recovery package for each major category resulted in a single recovery package serving both functions. The single recovery package is called the Single Strategy Fault Recovery (SSFR). It provides common recovery control for both interrupt or interject level faults and base level maintenance faults.

8.02 The SSFR does fault recovery tasks for faults occurring in either the active or the standby Attached Processor Interfaces (APIs). These faults or failures may be initiated by either the 1A or the 3B processor. The SSFR is divided into four major areas:

- Interrupt and interject control
- Base level maintenance control
- Common recovery control
- Timing administration.

The interrupt and interject control takes place in the AUFR, and the base level maintenance control is handled within the APFR.

APS ORGANIZATION

8.03 The APS replaces the disk file system on either the No. 1A or No. 4 ESS. The APS consists of one to eight 3B processors connected to the 1A Processor through an API system. The APS provides the 1A Processor disk access to a high-capacity 3B disk system. The API allows the sending and receiving of messages and blocks of data between the 1A and 3B Processor Systems. The API supports the attached processor communications link (APCL) protocol between the 1A and 3B processors. The APCL protocol has both efficient block transfer and message-handling capabilities. The APCL protocol also includes a high-priority maintenance message communication capability that is supported by the API. These messages are communicated in a closely coupled, synchronous, high-priority way by using the 3B IO interrupt and the 1A auxiliary unit bus (AUB) maintenance interject mechanisms.

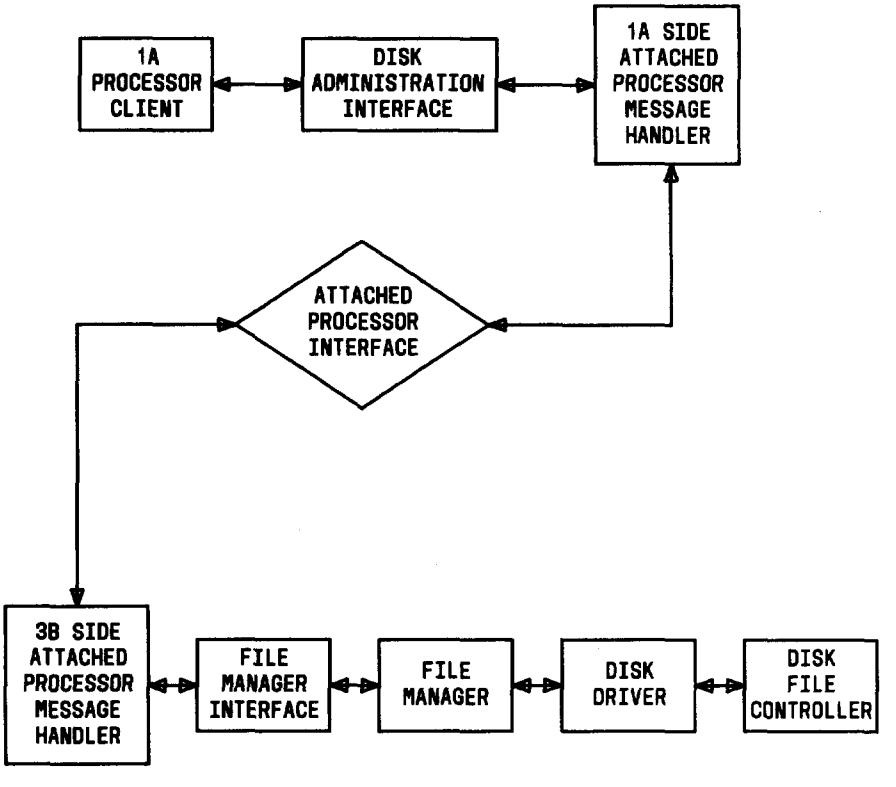
8.04 The APS includes attached processor message handlers on both the 1A and 3B sides of the API (Fig. 18). Also included are the file manager interface, the file manager, the disk driver, and the disk file controller, all on the 3B side.

SSFR—FUNCTIONS AND STRATEGY

A. Interrupt and Interject Control

8.05 The AUFR program is the interrupt and interject control program for the SSFR. All AU interjects and D-levels are first handled by AUFR; consequently, AUFR processes the interject or interrupt and tries to isolate the problem. The problem may be in the central control, the main memory, the AUB or in an individual AU. In the APS version of the 1A Processor, the AUs are the API and the data unit selector (DUS).

8.06 If the AUFR determines the fault was caused by an AU, AUFR communicates with the fault recovery programs for the faulty AU. The AUFR program communicates with the unique fault recovery



♦Fig. 18—Attached Processor Interface Layout♦

programs through a transfer vector table. For interrupt and interject processing, AUFR requests the unique fault recovery programs to do these tasks:

- Load unique bins
- Process the unique trouble
- Report data
- Update plant measurements.

If the API was the faulty AU, APFR is the unique fault recovery program and is called to do the above tasks.

B. Base Level Maintenance Control

8.07 The APFR program is the base level maintenance control program for the SSFR. During normal operations of the APS, there is continual checking for errors in these major areas:

- Link integrity monitor
- APS message handler
- System audit of disk.

8.08 The link integrity monitor does a check every second on the active and standby links to the 3B. Base level maintenance is called when there is no access to the 3B or the API through either the active or standby link.

8.09 The APS message handler calls a service routine within APFR to get the K-code of the active API. If an active API cannot be found, a failure is returned to the message handler. The message handler will then call base level maintenance control.

8.10 There is an audit done by the System Audit for File Store Administration Program (SADK) that monitors disk activity to ensure that jobs are being completed. If jobs are being accepted but not completed within a certain time, base level maintenance is called.

8.11 The base level maintenance control routine, APFRBLM, does three primary functions:

(a) Gets a base level maintenance report printed on the TTY

(b) Stops all AUs

(c) Sets all interrupt inhibits.

C. Common Recovery Control

8.12 The common recovery control routine, APFRTBL, may be called from either Interrupt and Interject Control or Base Level Maintenance Control. The purpose of this routine is to determine the error and to recover from the error condition. The error and recovery information is formatted into proper form for printing on the TTY. The recovery interaction between APFRTBL and AUFR is important since the AUFR routine, known as double trouble (AUFRDTBL), is the backup recovery used by APFR.

8.13 Common recovery control functions are:

- Error detection
- Error analysis
- Error recovery
- Error termination.

8.14 The error information word is built during error detection. This word is used to record as much information about the fault condition as possible. This information is used by the recovery module to determine which course of action to take for recovery.

8.15 One of the basic strategies of error recovery is to initialize the buffers only when absolutely necessary. If recovery can be made without initializing the buffers, jobs to and from the 3B will not be affected; but, if the buffers require initializing, all the jobs in the buffers are lost.

8.16 Another strategy of error recovery is to have a configuration of the API regardless of the state of the finite state machine (routine APFRTBL). The current active API may be reconfigured or its mate can be configured. The API that is configured will be the active API upon return to the system. There is one exception. When a fault occurs in the standby API and it is removed from service, there will not be any configuration performed on the active API. If an API is configured, the following actions take place:

(1) The peripheral interface controller (PIC) is reset.

(2) The API is informed about the location of the common buffer resources.

(3) The appropriate state for the API and update status is determined.

(4) The 3B is informed of the configuration of this API.

(5) The maintenance control console (MCC) lamps and power switch lamps are updated.

8.17 Following the recovery actions in any state, the active link to the 3B is tested to ensure communication between the 3B and 1A exists before the fault recovery ends.

8.18 Another strategy of error recovery is for all recovery routines to have a pass-fail indication. Error recovery does not assume that recovery actions are done successfully. The pass-fail indications from recovery modules allow for intelligent decisions to be made within error recovery.

8.19 The final strategy of error recovery is the recording of all recovery actions. These recovery actions are recorded in the recovery information word in memory. The recovery information word is included in the printout and is used for determining the exact recovery actions taken. If any recovery module fails, the reason for failure is saved in memory and is also included in the printout.

8.20 The error termination function used by all three states (paragraph 8.71) to end processing is the same for all states. It does three functions:

(1) Saves all information gathered during processing

(2) Formats information for printing

(3) Updates lamps.

D. Timing Administration

8.21 The common recovery control routine, APFRTBL, has three states: 0, 1, and 2. The processing of a fault may begin in any state and will end in the same state. To return the routine to state 0, a sequence timer allows the state counter to be reset to 0 after a certain time has elapsed.

SSFR—PROGRAM STRUCTURE

A. Interrupt and Interject Control

8.22 The AUFR program determines for the SSFR that the API is faulty and calls the APFR to do the following tasks (Fig. 19):

- Load unique bins
- Process the unique trouble
- Report data
- Update plant measurements.

Load Unique Bins Task Routine

8.23 The unique bins is a 100 decimal word area of program store 0 used to save information about the interrupt or interject. The load unique bins task routine in APFR is APILDBI. This routine has two functions. First, it initializes the unique bins area with an empty code; and then, it loads some data about the interrupt or interject into the bins. The APFR program loads three words in the unique bins area. These three words, read from the PIC are:

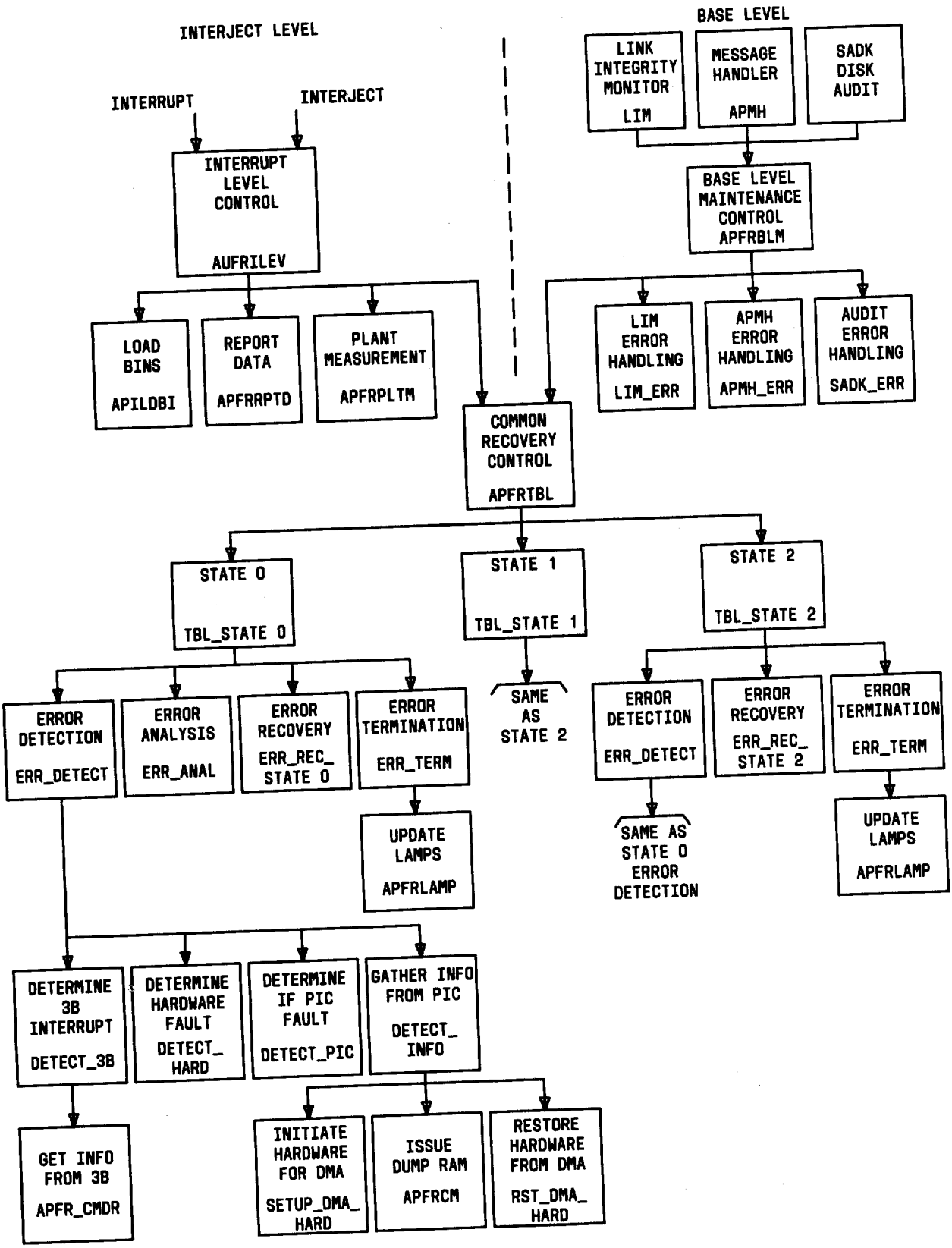
- PIC status word
- PIC error word
- PIC state word.

The PIC status word contains important information about the status of the API. The error word may contain information about an error if one has occurred in the API. The state word contains the state (active or standby) of the API at the time of the interrupt or interject.

Process the Unique Trouble Task Routine

8.24 Routine APFRTBL is the common recovery control routine for the SSFR. Detecting the type of error, recovering from the error, formating data to be printed—are all functions of APFRTBL. All AUFR task routines, except the unique trouble task routine, return a success to the address in the J register. The unique trouble task routine has three different ways to return to AUFR.

(a) Return to AUFRALDN in pident AUFRILEV. This return is used if the unique trouble task routine found the problem and recovered the APIs.



♦Fig. 19—APFR—Single Strategy Fault Recovery Structure Chart♦

(b) Return to AUFRSERR in pident AUFRILEV. This return is used if the unique trouble task routine could not find any problems. The AUFR maintains a count of the number of times the unique trouble task routine could not find a problem. If the count exceeds a threshold, AUFR bootstraps the AU community.

(c) Return to AUFRDTBL in pident AUFRILEV. This return is used if the unique trouble task routine determines the problem is so severe that a bootstrap of the AU community is required. This return, known as double trouble, will guarantee the AU bootstrap.

Report Data Task Routine

8.25 After processing the interrupt or interject, control is passed back to AUFR. The AUFR program does some cleanup and then calls the unique fault recovery task routine, APFRRPTD, to report data. This routine can report two different kinds of data: unique bins and dump random access memory (RAM) data.

8.26 The unique bins contain data collected in the load bins routine and data from the unique trouble routine. The unique bins always get printed. If during the unique trouble routine the dump RAM data was collected, it will also be printed. The dump RAM data contains information about the communication buffers and the message that was being processed at the time of the interrupt or interject.

Plant Measurements Task Routine

8.27 Following the reporting of data, APFRPLTM, the task routine that administers the plant measurements, is called. Plant measurements are counts of the number of errors having occurred in the system. This routine increments counters for the following errors:

(a) Maintenance interject

(b) Software error

(c) Invalid address

(d) Out-of-range access (API to call store)

(e) Out-of-range access (API to program store)

(f) D-level interrupt

(g) In-range access (API tc central control)

(h) In-range access (API to call store)

(i) In-range access (API to program store)

(j) Unique API error.

8.28 Plant measurment counts are used by the applications programs. They do not affect AUFR or APFR, both of which maintain separate counts of errors for error threshold checks.

B. Base Level Maintenance Control

Error Handling Task Routine

8.29 The APFRBLM routine calls a separate routine for each client calling base level maintenance control (Fig. 18). For the link integrity monitor, it calls LIM_ERR; for the message handler, it calls APMH_ERR; and for the SADK disk audit, it calls SADK_ERR. Each of these routines is responsible for setting up the error information word. The error information word is important to common recovery control because it contains information about the error, which unit caused the error, and actions required for recovery.

8.30 The base level maintenance control routine APFRBLM is passed four inputs:

- The client
- The APS number
- The problem type
- The initialization request.

Printing a Base Level Maintenance Report

8.31 The four actions needed to get a base level maintenance report printed on the TTY are as follows:

- Issue the RPTLEVL macro
- Call the RPTACTN routine
- Call the RPTDATA routine
- Transfer to MARPBASE.

8.32 The RPTLEVL macro, issued in APFRBLM, sets a flag in a word used by the Maintenance

Restart Program (MARP). This flag will suggest to MARP that a base level maintenance report is to be printed.

8.33 The call to RPTACTN is made when a text phrase (indicating some action) is to be printed. The text phrase is passed in the call to RPTACTN. During the processing of the base level maintenance, the call to RPTACTN can be made several times. A call to RPTACTN is made in each of the separate client routines called by APFRBLM.

8.34 The call to RPTDATA is made when data about the error and the recovery is to be printed. The RPTDATA routine is called with the starting location of the data and the number of words to be printed. This routine may be called with up to six different areas to be printed. The call to RPTDATA for base level maintenance is made in common recovery control, because this call must be made before the transfer to MARPBASE if common recovery control had to call for a bootstrap.

8.35 The transfer to MARPBASE is the final action in getting the base level maintenance report printed. In addition to printing all text phrases and data, MARPBASE also does any necessary cleanup. When MARPBASE is finished, it will transfer to common recovery control that returns to APFRBLM. The APFRBLM routine then returns to the client that called for a base level maintenance.

Stopping AUs

8.36 The stopping of the AUs processing is done to prevent possible interference problems since these units may be doing some actions when recovery actions are needed. The action of stopping the AUs is done by calling AUFRSTOP. The AUFRSTOP routine calls every in-service AU and requests suspension of any direct memory access.

Setting All Interrupt Inhibits

8.37 The setting of interrupt inhibits is required since recovery actions may cause an interrupt as part of normal recovery. The recovery actions will also take longer than is allowed for a normal segment (10 ms); therefore, the excessive time interrupt (K-level) is also inhibited. Even though base level maintenance was called on base level, the recovery actions will not be done on this level. No segment breaks will be taken until the 1A Processor recovery is completed.

C. Common Recovery Control

Common Recovery Control Task Routine

8.38 The routine that does the common control is called APFRTBL and is located in pident APFRILEV. The APFRTBL routine can be called from two places, Interrupt and Interject Control in AUFRILEV or Base Level Maintenance Control in APFRILEV.

8.39 The routine APFRTBL may be considered as a finite state machine (Fig. 20). (A finite state machine may be either a hardware or software structure consisting of a finite number of states.) The routine APFRTBL is a software structure consisting of three finite states: 0, 1, or 2. The states are considered accept states because APFRTBL may accept faults in any state to begin processing. By definition, all processing of a fault must begin in one accept state and end in the same or a higher accept state. The processing cannot go from a higher to a lower accept state. Once state 2 is reached, all processing of a fault must begin in state 2 and end in state 2. The only way to recycle back to state 0 is for the sequence timer to time out.

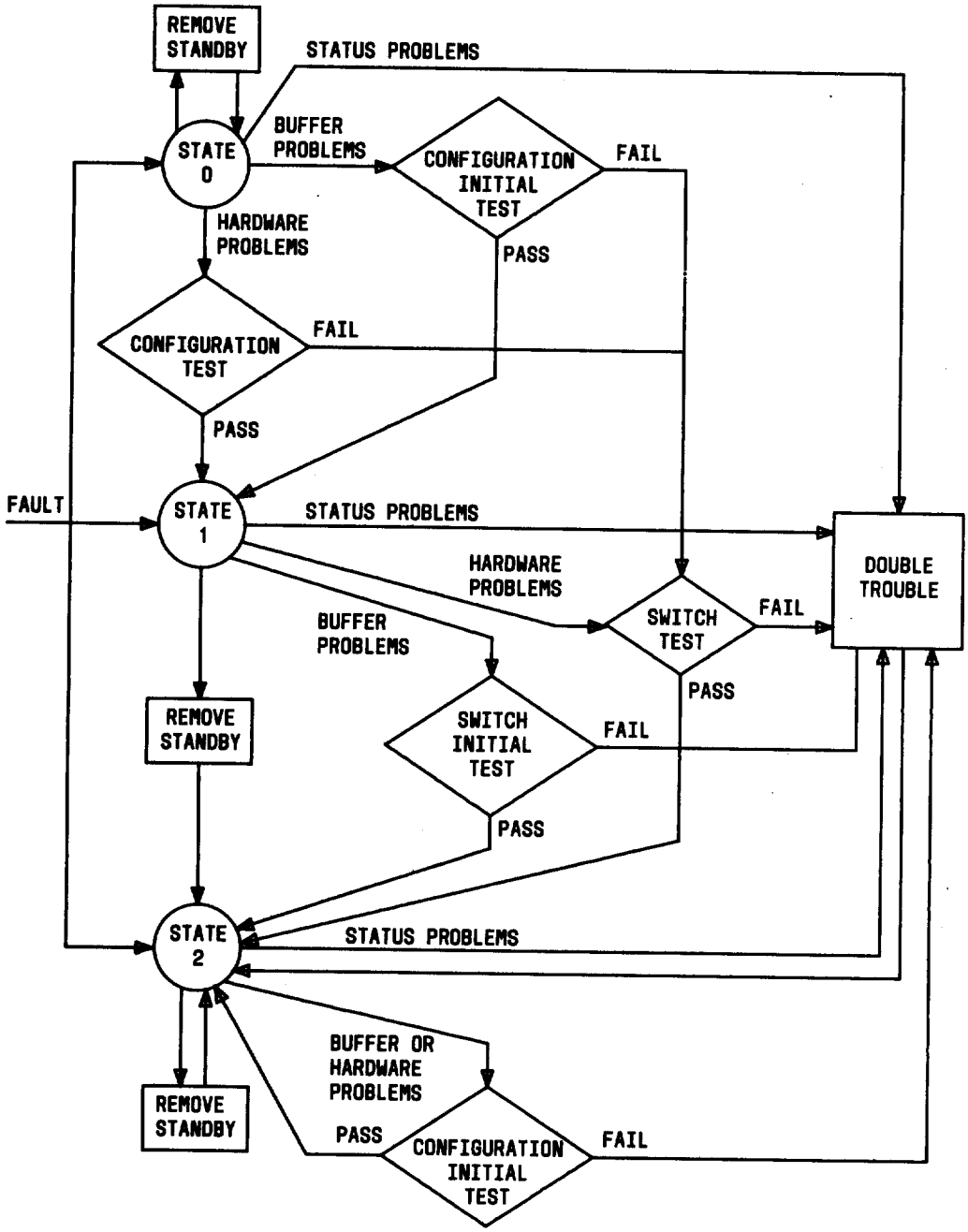
8.40 Although each state calls the same routine to determine the source of the error, only state 0 calls a routine that does any error analysis. Each state also calls a recovery module that does the actual recovery from an error. The recovery module is different for each state. Each state again calls the same routine to end the processing of the error.

Error Detection Task Routine

8.41 The error detection routine, ERR_DETECT, may be called from any of the three states of the finite state machine when APFRTBL is called from interrupt and interject control. But ERR_DETECT will not be called if APFRTBL is called from base level maintenance control. This is because ERR_DETECT looks for errors in the hardware, and firmware and base level maintenance control does not respond to errors in hardware and firmware. For base level maintenance, the error information word is built within the base level maintenance control.

Status Checks

8.42 The first action done within error detection is a status check. Every recovery state checks



♦Fig. 20—SSFR Common Fault Recovery Control State Diagram♦

the status of the API pair in the APS that is involved in the recovery. One and only one API must be active for an APS. If there is no active API or both are shown as active, a flag indicating no active API is set in the error word. Upon seeing this flag, the error recovery module calls "double trouble." This causes a bootstrap of the AU community, and the status of the APIs is corrected. If no errors were detected in the status, error detection must determine if the interrupt or interject was from the active or the standby API.

Interrupt From the Standby API

8.43 An interrupt from the standby API may mean the 3B is trying to interrupt and inform the 1A of an error condition in the 3B. The 3B will only interrupt the 1A over the standby API because errors occurring in the 1A-3B link will involve the active API. If the 3B tries to report errors to the 1A over the active API and the active is faulty, the attempt will probably fail. If the standby API is nonoperational, the 3B will be unable to inform the 1A of any 3B error conditions. The 1A will eventually discover the problem anyway because the initialization of the 3B buffers will be detected by the PIC, which in turn will interrupt the 1A.

8.44 The interrupting of the 1A by the 3B allows faster detection and resolution of error conditions. More information about the error condition is collected; and, with more information collected, the easier it is to determine exactly what happened. There are three types of failures the 3B will report to the 1A:

(a) The 3B buffers have been initialized by the attached processor driver.

(b) The APIs have been marked out of service by the attached processor driver.

(c) The attached processor driver has been initialized by the 3B Duplex Multienvironment Real Time Operating System (DMERT).

If the status word shows the 3B is interrupting the 1A, the command system response mode routine (APFR_CMDR) is called. This routine handles the communication protocol with the 3B and passes back the data sent from the 3B. Upon successful completion of APFR_CMDR, the data from the 3B is placed in the error information word. A code indicating response mode failure is placed in the error information word if this routine fails.

Interrupt From the Active API

8.45 If the 3B interrupt of the 1A is from the active API, a dump of the API memory is obtained. This dump is an aid to the craft in determining the problem. Before the dump, a routine (SETUP_DMA_HARD) will initialize the hardware for direct memory access (DMA). After the dump, a routine (RST_DMA_HARD) will restore the hardware. Although the interrupt and interject control called an APFR task routine (APILDBI) to load bins with interrupt data, the dump of API memory cannot be done at that time. This is because the load bins task routine is called too early in AUFR to allow DMA. Therefore, the dump of the API memory is done in the common recovery control.

Active and Standby API Error Detection

8.46 If the interrupt was from either the active or standby API (but not as a result of a 3B interrupt), two routines are called to determine the exact cause of the interrupt. The first, DETECT_HARD, tries to determine if there was a hardware error. The second, DETECT_PIC, tries to determine if there was a problem in the peripheral interface controller (PIC).

8.47 The hardware interrupt sources are located in the general control pulse response and the error summary register. Both the error summary register and the general control pulse response are in the AUB common interface hardware within the API. The AUFR program has access to the common interface hardware for all AUs and handles most of the hardware error sources. But, for the API, there are two hardware error sources that AUFR does not handle. One error source is for bad parity on a read; the other is for bad parity on a write. Both of these error sources appear in the error summary register so, if either source is detected, the source will be shown in the error information word.

8.48 After checking for hardware errors, ERR_DETECT looks for error sources from the PIC. This is done by reading the error summary register. If the PIC interject source is set, the PIC error word must be read. The errors the PIC can detect are:

(a) Circular buffer error

(b) Pointer check failure

(c) Illegal circular buffer command

(d) Command register inhibited

(e) AUBSQ inhibited

(f) AU address range error

(g) Parity failure on attached processor bus

(h) Parity failure on peripheral unit controller bus

(i) Command check error

(j) Maintenance buffer address not loaded

(k) Illegal peripheral unit controller RAM specified

(l) Peripheral unit controller error

(m) Illegal instruction.

8.49 After looking at all error sources to determine what caused the error, the error detection routine determines whether the communication buffers require initialization. This depends on the error sources. These error sources require initialization of the buffers:

(a) Initialization of the 3B attached processor driver's buffers

(b) Attached processor driver initialized by DMERT

(c) Circular buffer error

(d) Buffer pointer problem

(e) Command check error

(f) Illegal circular buffer command.

8.50 After the 3B error sources, the hardware error sources, and the PIC error sources have been checked, the error detection routine does one last check. The error information word is checked to determine if any error sources have been found. If none, the error detection routine issues a report action call with a text phrase stating that no error could be found. This action phrase appears on the interrupt or interject printout.

Error Analysis Task Routine

8.51 The error analysis routine, ERR_ANAL, is only called from state 0 of APFRTBL. The purpose of this routine is to allow for threshold levels to be evaluated for certain errors. There may be a time when recovery actions are not desirable for every occurrence of a certain fault. The error analysis routine allows a threshold count for those kinds of faults. The error analysis routine also enables the sequence timer. The sequence timer determines when the finite state machine (APFRTBL) can be recycled back to state 0.

Error Recovery Task Routine

General

8.52 The routine used to recover from faults in the SSFR program is ERR_REC. The ERR_REC routine is made up of three error recovery modules, one for each state: ERR_REC_STATE0, ERR_REC_STATE1, and ERR_REC_STATE2. The purpose of the error recovery modules is, of course, to recover from an error condition. It is not always known what recovery action must be done to recover from the error. Recovery from some errors only requires the API to be reconfigured. Some errors require the active API to be removed from service and the mate to be put in service. Some errors require the communication buffers to be initialized.

Actions Common to Every Recovery State

8.53 The finite state machine used for recovery of the APS has three states: 0, 1, and 2. Each state does some unique recovery; however, there are some recovery actions common to all states. These common recovery actions are:

- Recovery from status problems
- Recovery if interrupt from the standby API
- Reporting of recovery actions
- Collection of recovery information.

8.54 If the error detection routine determined there was bad status for the APS, the no-active-API flag is set in the error information word. Error recovery checks this flag before attempting any recovery. If this flag is set, error recovery calls

double trouble. There is no use in trying to perform recovery if the API to be recovered is unknown.

8.55 If the error information word shows the interrupt was from the standby API (but not initiated by the 3B), the standby API is removed. This provides the fastest recovery action since the standby API is not involved in the normal operation of the APS. The routine that removes the API also initiates diagnostics to be run on base level. If the diagnostics find no errors in the API, it is restored to standby.

8.56 Every successful recovery action done in any state results in a report action text phrase being issued. These text phrases aid in determining the kinds of recovery actions done. The text phrases are issued at the time of recovery action, resulting in them being printed in chronological order. This also aids in determining the sequence of recovery actions.

8.57 The collection of recovery information is important for two reasons. First, it is important to know if certain recovery routines have failed so that intelligent decisions can be made in fault recovery. Secondly, the recovery information is important to the craft and maintenance personnel trying to understand the fault.

Major Actions of Each Recovery State

8.58 Common recovery control may be entered from either base level maintenance or interrupt and interject control. The recovery actions done by error recovery depend on the state of the finite state machine. Processing will begin in state 0, 1, or 2 and end in one of these states. For a single entry into common recovery control, the maximum recovery is from one accept state to another accept state (ie 0 to 1 or 1 to 2). Processing through two accept states (0 to 2) cannot be done on a single entry.

8.59 The recovery actions done in each state are different from recovery actions of other states. The severity of the recovery increases with the number of the state. The primary recovery action for state 0 is to reconfigure the current active API, for state 1 to switch the APIs, and for state 2 to initialize the communication buffers. The most severe recovery actions are done when double trouble is called.

8.60 State 0 first checks the error information word to determine if the communication buffers must be initialized. If a buffer problem exists, it must first be cleared before configuring the active API or switching the APIs will not help solve the problem. If the buffers require initialization, it is done after reconfiguration of the current active API. If the buffers do not require initialization, only the current active API is reconfigured.

8.61 Following the reconfiguration of the active API, the APCL is tested. If the reconfiguration and link tests are successful, the finite state machine ends up in state 1. If either of the tests fail, the APIs are switched. The link is tested again following the switch. Now, if either the switch fails or the link test fails, double trouble is called. Successful completion of the switch and link test causes the finite state machine to end in state 2. State 1 is skipped because the switch that is the primary action of state 1 has already been done.

8.62 If the problem is in the standby API, it is removed from service and the finite state machine stays in state 0. It is not necessary to escalate to the next state since the standby is not involved in normal operation of the APS.

8.63 When the finite state machine begins processing in state 1, it will first check the error information word to see if the communication buffers need to be initialized. Should the buffers require initialization, the APIs are switched and the active link is tested before the buffer initialization. If the buffers do not require initialization, the APIs are switched and the active link is tested. If either the buffer initialization, the API switching, or the link testing should fail, double trouble is called. The finite state machine ends in state 2 when all the recovery actions succeed. If the problem is in the standby API, it is removed from service and the finite state machine advances to state 2. This is because the switch will fail if the finite state machine is left in state 1 with the standby API removed, and a fault should occur.

8.64 When the finite state machine begins processing in state 2, it first reconfigures the current active API and then tests the active APCL. Next, the communication buffers are initialized. If any of these three actions fail,"double trouble"is called. When all three actions succeed, the finite state machine again ends in state 2. Any later entries into common recovery control causes the above actions of state 2 to be done. To prevent state 2 from being repeated continu-

ally, there is a threshold count on the number of times the communication buffers can be initialized. When the threshold count is exceeded,"double trouble" is called.

Exceptions in Common Recovery Control

8.65 Although common recovery control does recovery actions for both interrupt level and base level, there are certain exceptions to it being completely common. The AUFR program has a word of memory that contains three flags. These flags show either an AU D-level, an interject level, or a base level fault. Common recovery control looks at the base flag to determine which control area called for recovery. The following is a list of the exceptions to the commonality of common recovery control:

(a) The RPTDATA routine is called from APFRTBL if the base flag is set. For interrupt and interject control, RPTDATA is called from AUFR.

(b) If invalid status is detected in base level maintenance control, the APFRTBL routine issues a report action to print a text phrase indicating invalid status.

(c) In the routines TBL_STATE0, TBL_STATE1, and TBL_STATE2, the error detection routine is not called if the base flag is set. The base level maintenance control does the actions of the error detection routine and sets up the error information word.

(d) A printout for a base level maintenance will not include a dump of RAM. The dump of RAM is not necessary for base level maintenance since the type of faults detected are normally not concerned with the job in progress at the time base level maintenance is called.

Double Trouble Task Routine

8.66 The AUFR double trouble recovery routine, AUFRDTBL, is the backup recovery for all AUs including the attached processor. This routine is called from common recovery control only when recovery cannot be done within the error recovery routine. The AUFRDTBL routine is called if any of the following conditions exist within the error recovery routine:

- Bad status has been detected.
- No working configuration can be found.
- The buffer initialization threshold count has been exceeded.

As shown in Fig. 19, AUFRDTBL can be called from almost anywhere in the finite state machine and the machine always ends up in state 2. The recovery actions in AUFRDTBL are more severe then in state 0 and 1; therefore, having the finite state machine end in state 0 or 1 is inappropriate. The recovery action of state 2 is to initialize the communication buffers and, since AUFRDTBL does not initialize the buffers, state 2 is the appropriate state to end.

8.67 The primary recovery action of AUFRDTBL is to bootstrap the entire AU community. The bootstrapping of the AU community involves:

(1) Initializing the status on all AUs.

(2) Performing an unconditional restoral of all AUs.

(3) Testing all AUs. (This is an exhaustive test of the AU hardware that interfaces with the AUB. The hardware used to do direct memory access into call store and program store is also tested.)

(4) Removing any AU that fails to pass all the above tests.

8.68 If the bootstrap of the AU community is successful, AUFRDTBL does additional recovery actions dependent on the value of the emergency action state count. The emergency action state count is used by AUFRDTBL to escalate recovery if AUFRDTBL is called repeatedly. If the emergency action state count has a value of 0 or 1, the count is incremented and no additional recovery actions are taken. Some recovery actions are done for values greater than 1 and, also, the count is incremented.

8.69 The relationship between the emergency action state count and the recovery actions is as follows:

(a) 0—Increments the emergency action state count (bootstrap with no additional recovery)

(b) 1—Increments the emergency action state count (bootstrap with no additional recovery)

(c) 2—Tries to switch the central controls

(d) 3—Tries to switch the central controls again

(e) 4—Removes AUB 0

(f) 5—Removes AUB 1

(g) 6—Removes all nonessential AUs

(h) 7—Duplex fails the essential AUs

(i) 10—Duplex fails the essential AUs

(j) 11—Generates a processor recovery.

8.70 When AUFRDTBL is called from common recovery control, there may or may not be a return. If common recovery control was entered by interrupt and interject control, AUFRDTBL will not return to common recovery control. Instead calls MARP which will return to a reference point or to the point of interrupt. If common recovery control was entered by base level maintenance control, AUFRDTBL will return to common recovery control. Next, AUFRDTBL will call MARP, and MARP will return to whoever is in the T register. The AUFRDTBL routine is called on the T register so MARP will return to common recovery control. Whenever AUFRDTBL is called, the finite state machine state counter is always set up to state 2.

Error Termination Task Routine

8.71 The error termination routine, ERR_TERM, is used by all three states to end processing. The purpose of the error termination routine is to save and format the information gathered during processing. It will also update the lamps on the MCC and the power switch.

8.72 Information gathered during processing is saved for printout in later fault entries. Information gathered during processing of a fault in state 0 is saved and included in the printout for state 1 and state 2, if these states are entered. There are three different tables used to save information.

- Error information table
- Recovery information table
- Test link results table.

8.73 Each word in the table represents a state in the finite state machine. When fault processing is ended, the error information word is put in the error information table. It is placed in the word corresponding to the value of the state counter on entry into the finite state machine. The recovery information word is put into the recovery information table similarly. The results of the last test of the active link are put in the test link results table.

8.74 There are two reasons for saving and printing the information for all states that were entered. First, if for any reason the printout for a state gets lost, the information is always available on the last printout. Second, looking at multiple printouts is unnecessary because the last printout contains the information from previous fault entries.

8.75 The formatting of the saved information is also done in the error termination routine. All of the information to be printed is placed in a Compool defined area known as the unique bins (AU1RIQBINS). This area is available to any unique AU. For the APFR, this area is only used as a buffer for data to be printed.

Update Lamps Task Routine

8.76 The error termination routine calls a routine that updates the lamps. The lamps in the power switch as well as the lamps on the MCC are updated to reflect the current status of the APIs.

D. Timing Administration

General

8.77 The finite state machine can only increment from one state to another for each fault entry but cannot decrement the state counter. Therefore, whenever state 2 is reached, the finite state machine stays there unless the counter gets reset. The importance of timing administered in AUFR is to reset the state counter to 0 after a certain time. If there is no timer, the finite state machine stays in state 1 or 2 indefinitely. This results in excessive recovery actions being done whenever a fault is detected. The sequence timer begins timing only after being enabled. It is enabled within common recovery control in the error analysis routine, which is called in state 0.

Timer Administration Task Routine

8.78 The administration of the timer is done in the routine APFREX. This routine is entered

every second from the Maintenance Control Program (MACP) that runs on base level. If the sequence timer has been enabled, a count will be incremented. If this count exceeds 60 seconds, the sequence timer is disabled, the state counter is reset to 0, and the following data structures are initialized:

- Error information word
- Recovery information word
- Error information table
- Recovery information table
- Test link results table
- Fault recovery miscellaneous information word.

The fault recovery miscellaneous information word contains the sequence timing count and the sequence timing enable flag.

E. Duplex File Store Failure

8.79 Although the APS does not use file stores in the same way file stores were used prior to the CPR7 generic, the same terminology is used here for consistency. In the original file store system, there were two levels of duplex file store failure but in the APS there is only one level. Duplex AUB outage, duplex API outage, duplex disk status check outage, duplex 3B disk file controller outage—all these factors could cause duplex failure.

Detection

8.80 Duplex failure can be detected in two ways by two different programs. The first program APFRLM, will detect such conditions as duplex AUB outage, duplex API outage, duplex disk status check bus problems on the 3B, and 3B attached processor driver problems. The second program SADKTI, located in the disk audit pident SADK, is able to detect problems in 3B software as well as problems in the 3B file system.

Operation

8.81 When the system is driven to duplex file store failure, the APIs are removed from service, MACP jobs are idled, and the system's duplex file store failure flags are set. A message is printed on the TTY at regular intervals until the duplex file store failure is cleared. During duplex file store failure, operation in the APS is identical to that in the file store system when both file store controllers are out of service.

Recovery

8.82 After the faulty hardware or software is corrected, recovery from duplex failure is accomplished depending on whether file mutilation has occurred. If mutilation occurred then the System Audit of Stores from Tape pident must be used to reload the disks. When no file mutilation has occurred, recovering from duplex failure is simply the restoring of an API unconditionally and then entering the new message to clear duplex failure.◀

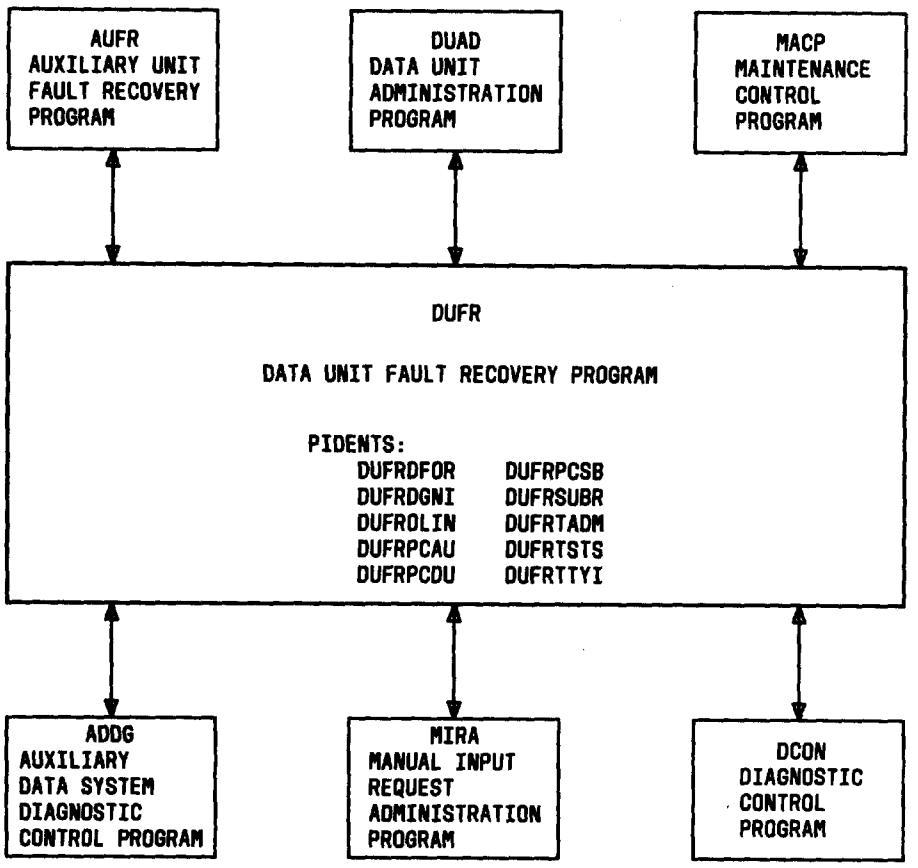
9. DATA UNIT FAULT RECOVERY PROGRAM—DUFR

INTRODUCTION

9.01 The DUFR program performs the fault recovery tasks for the auxiliary data system consisting of data units connected to the AUB system. The data unit fault recovery approach consists of finding a set of data units that is capable of carrying out the normal tasks associated with data stored on tape. This determination is made by a detailed set of tests that are run on interject or interrupt level priority or on demand via a TTY message. To perform its functions, DUFR interfaces with a number of other programs. The major DUFR program interfaces are illustrated in Fig. 21.

9.02 The DUFR program is entered from the Auxiliary Unit Fault Recovery Program (AUFR) of D-level interrupts and maintenance interjects. The AUFR program is responsible for all AU fault recovery tasks relating to the AUB system. The AUFR program selects the proper subsystem, FSFR/APFR or DUFR, based on indicators saved at the time of the interrupt. The AUFR program makes additional checks, resolves certain problems associated with the AUB system between central control or the stores (program or call store) and the data units, and routes data unit problems to the DUFR program.

9.03 The DUAD program administers the work to the data unit community and monitors its progress. When DUAD detects a problem during its base level maintenance cycle, control is passed to



♦Fig. 21—Data Unit Fault Recovery Program (DUFR)—Program Interfaces♦

DUFRTADM that administers tests to isolate the faulty unit and remove it from service.

9.04 The MACP program schedules the DUFR program periodically for routine diagnostic on all auxiliary data system units. The MACP program also schedules DUFR on a regular basis to normalize the auxiliary data system community.

9.05 The Auxiliary Data System Diagnostic Control Program (ADDG) performs diagnostics on the auxiliary data system data units as required by the DUFR programs. The ADDG diagnostics are requested through the Diagnostic Control Program (DCON) that schedules the required diagnostics through the MACP job tables. Diagnostic results are usually returned to DUFR for final analysis and disposition.

9.06 The MIRA program interfaces with DUFR to remove or to unconditionally restore a data unit selector (DUS) or a tape unit controller (TUC) to service. The TTY request for this service is detected in MIRA and control passes to MACP. The MACP program then schedules a job for DUFR.

AUXILIARY DATA SYSTEM ORGANIZATION

9.07 The auxiliary data system is comprised of a community of data units. The system is capable of being expanded to a maximum of two communities. Each community consists of two DUSs each connected to a single AUB (DUS 0, AUB 0; DUS 1, AUB 1), a minimum of 2 tape units and up to a maximum of 16 per community, and 2 data unit buses that interface the DUSs and tape units. Each tape unit has associated with it TUC that allows the tape unit to be configured to either of the two data unit buses and in turn to either of the DUSs. The auxiliary data system serves as a backup to the file store system for system reinitialization and as the prime facility for

program updating, automatic message accounting, data recording, and other functions (Fig. 14 and 15).

DUFR—FUNCTIONS AND STRATEGY

A. General

9.08 The DUFR program is designed to operate under D-level interrupts, maintenance interject, base level maintenance, and on demand via TTY request. The DUFR program performs the following functions:

(a) Removes and restores data units

(b) Tests the bus circuitry, internal DUS register, and DUS flip-flops

(c) Performs configuration of the data unit community

(d) Administers diagnostic requests

(e) Administers TTY message input and output requests

(f) Provides a common interface for other system programs.

All these functions are performed to maintain a viable auxiliary data system.

9.09 Sometimes DUFR uses the first-look approach to fault recovery that consists of retrying the failing operation with simple and fast testing techniques. If this approach fails to identify the source of trouble, DUFR resorts to more detailed testing to isolate the problem.

B. Basic Program Strategy

9.10 The DUFR program attempts to keep the tape units normalized as much as possible. This means configuring an equal number of TUCs to each DUS. System conditions may require that the data units be configured differently to maintain a viable auxiliary data system. The normalization scheme is exercised once a day at some nonbusy hour to equally divide the TUCs between the data unit buses. At this time, all TUCs are switched from the DUSs they have been configured with to the other DUS. This is done to ensure that all communication paths are exercised. The TUCs and DUSs are also exercised (diagnosed) once a day to ensure that they are capable of being used as the need arises. This exercise is also done at a nonbusy hour.

9.11 One source of entry into DUFR is from D-level interrupts representing central control communication and call store/program store communication problems, read/write out-of-range, etc. The D levels are detected and enter AUFR (the main auxiliary unit fault recovery program). The AUFR program can resolve some problems relating to the data unit community. If AUFR can define the specific unit at fault, DUFR will be entered to remove that unit. The DUFR program is responsible for configuring all the auxiliary data system units but will accept requests to remove or restore specific units. If AUFR is unable to define the specific unit at fault, DUFR is entered to resolve the problem and then to remove the faulty unit. The select registers keep records of which TUC is being accessed, and since only one TUC may be accessed at the time, the recovery is enhanced by having to test only over that particular path.

9.12 The Data Unit Administration (DUAD) program is the interface program that allocates work to the auxiliary data system. During normal processing, tape reads and writes are issued, and DUAD monitors the progress of these functions. When a fault occurs in the TUC, DUAD detects this as an operational interject (TUC maintenance interjects appear as operational interjects to the system), and determines that it is a maintenance type of indicator rather than the normal operational type indicator. Capstan motion abnormal is one such fault. The DUFR program then administers such tests as are necessary to isolate the problem and remove the faulty unit from service.

9.13 When it is necessary to change the configuration of the auxiliary data system units for nonemergency removals and restorals, the request will be honored when MACP allots a time segment. These jobs usually consume more time than is available in one segment, and segmenting is used. The request to switch TUCs is detected by the DUAD program. The DUAD program then enters DUFR with the switch request at a convenient time that will prevent interruption of normal tape operations.

9.14 Diagnostic requests can be initiated by MACP, MIRA, and DUFR. The MACP program initiates routinely scheduled diagnostics. The MIRA program handles MANUAL requests while

DUFR initiates all fault-related requests. All requests are scheduled through MACP. The MACP program then comes to DUFR to initiate the diagnostics. The DUFR program determines the unit to be diagnosed, removes one of the DUSs and its bus from service, and assigns the unit to be diagnosed to this DUS. The remaining DUS, its bus, and TUCs remain in service for normal operation. The DUFR program then issues the diagnostic request through DCON to ADDG. Programs ADDG and DUFR then communicate with each other to perform the diagnostic, analyze the results, and remove the unit as dictated by the diagnostic results. The DUFR program issues reports via TTY about the progress of all of its jobs to keep operating personnel informed as necessary. These reports are made via the Maintenance Restart Program (MARP) in interrupt cases or through the Input/Output Control Program (IOCP) normally.

DUFR—PROGRAM STRUCTURE

A. General

9.15 The DUFR program consists of ten pidents to accomplish the task of fault recovery for the auxiliary data system. These pidents communicate frequently with one another and with other systems. The pidents that constitute the DUFR program are as follows:

(a) DUFRDFOR—Deferred fault recovery test routines (PR-5A308)

(b) DUFRDGNI—Diagnostic interface routines (PR-5A309)

(c) DUFROFLN—Off-line configuration routines (PR-5A310)

(d) DUFRPCAU—AUFR interface routines (PR-5A311)

(e) DUFRPCDU—DUFR interface routines (PR-5A312)

(f) DUFRPCSB—DUFR subroutines (PR-5A313)

(g) DUFRSUBR—Auxiliary data system miscellaneous service routines (PR-5A314)

(h) DUFRTADM—Test administration routines (PR-5A315)

(i) DUFRTSTS—Auxiliary data system test routines (PR-5A316)

(j) DUFRTTYI—TTY interface routines (PR-5A317).

B. DUFR Pidents

DUFRDFOR Description

9.16 The DUFRDFOR pident contains the program necessary to administer the work needed to execute the deferred data unit selector (DUS) and tape unit controller (TUC) tests. This includes initializing the indicated selector, running the specified tests, and reconfiguring the community upon completion of the tests. Also included are routines that determine for the DUAD program whether a TUC function should be reassigned. The MACP program comes to DUFRDFOR to determine if the auxiliary unit bus (AUB) activity can be stopped over a segment.

9.17 The DUFRDFOR pident is entered at the DFORDFER entry to perform the deferred DUS tests. These tests are done for one of two conditions: (1) after the DUS has been diagnosed, for any reason, and its results are all tests pass (ATP) indicating that no problem has been found, (2) on receipt of a "TEST:DUS" TTY message. The tests initiated by this pident are:

- Common AU tests
- DUS register tests
- Select verify register tests
- All-seems-well tests
- Reply bus parity tests
- Control bus parity tests
- Post-poll tests.

The DUFR program transfers to the AUFR deferred fault recognition routine AUFRDEFR to perform the common AU tests. The DUS register tests are performed by the DUFR test administration pident DUFRTADM. All other tests are done in the DUFRTSTS pident.

9.18 Initializing the auxiliary data system for deferred fault recovery tests includes configuring the DUS and a TUC to be used as a helper

unit. The DUFRCNTL routine is then called to initiate the requested DUS test. Each test may have a number of phases. The tests are table-driven and these tables contain start and stop addresses of transfer tables for each of these tests. Each transfer table is made up of a series of transfers to the sequence of phases within the test. Each phase is entered in turn until all phases have been run or until a failure occurs. No additional phases are run after a failure has occurred.

9.19 These tests are long and only one test may be done in any segment. Thus upon the completion of a test, the auxiliary data system is reconfigured to the pretest conditions. This allows DUFR to take a segment break before the next test. The reconfiguration restores the DUS and TUC to their pre-diagnostic state that requires the start routine DUFRSTRT be run. After the segment break, DUFR is again entered to continue executing the various tests and continues this procedure until all tests have been run.

9.20 The DUAD program monitors the status of the auxiliary data system and when it finds an assigned TUC marked out of service, DUFR is entered. The DUFR program is entered at the DUFRWAIT entry to determine if the function assigned to the out-of-service TUC should be reassigned to another TUC. If the DC1RWAIT bit is set, the TUC is temporarily out of service for a diagnostic. If it is not set, the TUC is permanently out of service and its function is reassigned. If it is in a wait state, and checks on the type of job it performs pass, the diagnostic is aborted and DUAD is instructed to not reassign but to wait for its return to service.

9.21 When an MACP client gets a segment break, that client can request certain conditions be established for the next segment. One of these conditions is that AUB activity be stopped. When this is the case, MACP enters the DUFR entry DUFRHOLD and checks for any jobs that cannot be held up for one segment. If autonomous jobs are in progress, the request is denied. If fast data rate jobs are in progress, G-level timing is set up that allows the jobs to be completed. The DUAD program is instructed not to start any new jobs and the request is granted at the end of G-level timing. If no jobs are in progress, the request to stop AUB activity is allowed.

DUFRDGNI Description

9.22 The DUFRDGNI pident is the interface between DUFR and the diagnostic routines. Diagnostics are run any time a unit is restored to service. The diagnostics are also run routinely on each auxiliary data system unit once a day as scheduled by MACP. Units removed by the fault recovery programs and units returning from an off-line configuration are diagnosed before returning to service. Diagnostics are run on demand when requested via TTY as part of a maintenance service.

9.23 The DUFRDGNI pident is a collection of subroutines necessary to accomplish the required diagnostics. Included are initial and final handlers, routines that perform the routine exercise of all TUCs and DUSs daily, routines that normalize TUC configuration once per day, as well as several smaller routines that assist in these functions. The initial handler is entered at DUFRPRDG. This routine configures the auxiliary data system for diagnostics by placing the units to be diagnosed on one bus while keeping other units in service on the other AUB. The diagnostic tests consume more time than is allowed in one segment and the system is returned to normal at the end of each segment and reconfigured for diagnostics when entered again. Routine DUFRPRDG transfers control to DCON that schedules the diagnostic ADDG. The diagnostic buffer table contains the data for diagnostic phases requested, unit and member number, as well as any helper unit if specified.

9.24 The final handler routine is entered after the diagnostic is complete to analyze the results. Units that fail the diagnostics are removed from service. If no failures are recorded, the final handler enters routines that return the units to service and restart the system.

9.25 The DUFRDGNAEX routine is entered from MACP on a scheduled basis once a day to diagnose all equipped and in-service auxiliary data system units in both communities. The tests start with both the DUSs for the community and then pick up the TUCs in sequential order. Segment breaks are taken as needed. The MACP program control is retained via the post-diagnostic final handler until all equipped and in-service units have been diagnosed or until the allotted time has been exceeded. The unit number where this occurs is stored. Testing continues with this unit at the next scheduled time segment.

9.26 The DUFRCFRAEX routine is entered from MACP on a scheduled basis once a day to nor-

malize the auxiliary data system. This is done 12 hours after the ADS units have been diagnosed by the routine exerciser. The system is normalized by distributing the in-service TUCs evenly between the DUSs. After the normalization is complete, a request is made to switch all TUCs from the DUS to which it is presently assigned to the other DUS. This is done for both DUSs to ensure that all units are capable of being switched, to keep them evenly distributed for normal processing, and to ensure that all communication paths are exercised.

DUFROFLN Description

9.27 The DUFROFLN pident is used to configure an off-line auxiliary data system to perform off-line functions or testing and diagnostics. The DUFROFLN pident is entered from AUFR at the DUFROAVAL entry to determine if a specified DUS and sometimes a TUC can be removed from the on-line system. If the entry is to set up an off-line auxiliary data system consisting of an AUB, DUS, and TUC, DUFR checks to see if the nonrequested DUS is in service and that all the in-service TUCs can be assigned to the nonrequested DUS. If a TUC is requested, that TUC must be out of service and have no functions assigned to it. If the request from AUFR is for a removal and the requested DUS is out of service, an immediate pass return is given. If the requested DUS is in service, the auxiliary data system is reconfigured to show that DUS out-of-service and all in-service TUCs are configured to the other DUS. The status of affected units is updated to reflect those units in the off-line mode.

9.28 The off-line functions are automatically terminated upon completion of the job. The off-line configuration may be terminated and those units returned to service if they are needed for system operation. The termination procedure first removes the off-line units again to guarantee correct hardware and software agreement. Such units are then diagnosed and restored to service if they pass.

DUFRPCAU Description

9.29 The AUFR program enters the DUFRPCAU pident during the execution of fault recovery measures. Some entries perform no operations other than to return control to AUFR but exist for commonality. When entered because of a fault, DUFRPCAU fills the load bins with print data about the units at the time of the fault. The AUFR program enters the DUFRDSRM entry to remove a DUS that AUFR has defined as faulty during the AUFR fault recovery processing. The suspect unit is removed from service via the DUFRNEUT routine in DUFRPCDU and a diagnostic is requested. On completion of the diagnostic, the unit is restored to service by DUFRPCAU routines if all tests pass.

9.30 After the auxiliary data system has been stopped for either fault recovery or nonfault recovery functions, the auxiliary data system restart is done by entering DUFRPCAU at DUFRSTRT. The start routine then performs exhaustive checks before starting the auxiliary data system. There are two entry conditions. One is the result of a processor configuration change in that no units are known to be functional and a complete initialization is performed. The software status is developed from the hardware configuration. The second entry is for all other conditions and assumes that some units are functional and that the software status is correct. This entry performs an initialization on the DUS hardware only.

DUFRPCDU Description

9.31 The DUFRPCDU pident is entered after testing has been performed by fault recovery or diagnostics, to update the status words of the DUS or TUC. The remove and restore bits have been set according to test results. The failure may have been pinpointed to a DUS, to a TUC, or all tests may have passed. If all tests have passed, no units are removed but a record is kept of the number of times an all tests pass (ATP) occurs. The DUFRRATP entry is used when all tests pass on a fault recognition entry. The DUFRRMDC entry is used when fault recognition has determined that the failure is in a TUC. The DUFRRMDS entry is used when fault recognition has determined that the failure is in a DUS. The DUFRNEUT entry point is the main entry and is used by other DUFR routines that have isolated the problem to a specific unit. Entry point DUFRNEUT is from diagnostic final handler and from manual actions (remove and restore messages). Exit from this pident is to the calling program.

DUFRPCSB Description

9.32 The DUFRPCSB pident is a collection of 13 auxiliary data system subroutines. Each subroutine performs a single unique function and is used by one or more other DUFR pidents. The subroutines and their functions are listed below:

(a) DUFRCNFG—Calculates a normalized DUS/TUC configuration based on present status

(b) DUFRGNST—Updates the TUC software status to agree with the hardware

(c) DUFRHINT—Initializes DUS hardware to the in-service state

(d) DUFRIRUD—Initializes the DUS inihibit register to present TUC status

(e) DUFRDSLP—Updates DUS primary/secondary trouble lamp on the MCC

(f) DUFRTCLP—Updates TUC primary/secondary trouble lamp on the MCC

(g) DUFRDSEQ—Determines if the specified DUS/TUC is equipped

(h) DUFRCLDS—Clears AU D-level interject source flip-flops

(i) DUFRGCPC—Determines if the specified DUS can be generate control pulse (GCP)

(j) DUFRBST1—Chooses a TUC to be used as a DUS diagnostic helper unit

(k) DUFRCLRA—Clears specified bits in TUC status for all TUCs

(l) DUFRSETA—Sets specified bits in TUC status for all TUCs

(m) DUFRGTID—Converts TUC identity to binary format.

9.33 The configuration routine DUFRCNFG is entered from the start routine DUFRSTRT to normalize the TUC load between the available DUSs whenever a unit is removed or restored to service. This is done by distributing the configuration evenly between the DUSs, taking in consideration such factors as present assignments, requests for new configuration, and bus troubles associated with a TUC.

9.34 The generate status routine DUFRGNST is entered from the start routine DUFRSTRT to determine the hardware state of the DUS and TUCs in the in-service DUS community. This entry also initiates the TUC status words to reflect the hardware status so that a valid DUS-to-TUC configuration will be established. If no errors are encountered, control returns to the start routine. If errors are detected in the even DUS (DUS 0), control returns to the caller of the start routine without checking the TUCs. If errors are detected in the odd DUS (DUS 1), control returns to the caller of the start routine but only after the TUCs have been configured to the good DUS.

9.35 The inhibit register update routine DUFRIRUD is entered to translate the TUC assignment in status to the inhibit register format. The inhibit register denotes the TUCs connected to the DUS. There is one inhibit register for each DUS.

9.36 The equipage check routine DUFRDSEQ determines the equipage of any type unit in the auxiliary data system community. This routine is structured to accept either K-code data or unit type and member number.

9.37 The locate suitable TUC helper unit routine is entered to find a TUC that is now assigned to diagnostics or as a spare to be used to diagnose a DUS. It has two entries: DUFRBST1, the initial entry to find a helper unit, and DUFRBSTC, the second entry to continue the search. On initial entry the program attempts to find a helper that is assigned to diagnostics or is a spare and that has no trouble bits set. If no helper is found after checking twice, the retry bit restriction is lifted and two additional passes are made. If a helper is still not found, the suspect bit restriction is lifted and a final attempt is made. If during any pass a helper is found, its K-code is passed to the user. If this helper is unsuitable to the user, the program is reentered at DUFRBSTC to continue the search for another helper and begins at the K-code of the last helper found.

DUFRSUBR Description

9.38 The DUFRSUBR pident is a collection of miscellaneous service routines for the auxiliary data system. Pident DUFRSUBR consists of six program units, each of which performs a single unique function for a non-DUFR program client. These routines are entered from MACP for the user program, an MACP client. The functions performed by DUFRSUBR program units include:

(a) DUFRDSST—Assembles DUS status for the output message "output DU status"

(b) DUFRDCST—Assembles TUC status for the output message "output DU status"

(c) DUFROSDS—Updates the DUS out-of-service frame lamp for all DUSs

(d) DUFROSTC—Updates the TUC out-of-service frame lamp for all TUCs

(e) DUFRSOS1—Determines the out-of-service DUSs for the output message "OP:OOS units "

(f) DUFRTOS1—Determines the out-of-service TUCs for the output message "OP:OOS units."

9.39 The DUFRDSST routine is entered to determine if a specified DUS is in service or out of service for the auxiliary data system output message "output DU status." In addition other pertinent status information for the specified DUS is assembled to be printed in the output message as raw status data. This routine is entered only if the specified DUS is operationally equipped. The routine is reentered for each operationally equipped DUS until all such DUSs have been processed.

9.40 The DUFRDCST routine is entered to determine if a specified TUC is in service or out of service for the auxiliary data system status output message "output DU status." It also determines the DUS to which it is configured and includes this data in the message. In addition, other pertinent data about the TUC is collected and assembled to be printed in the output message as raw data. The routine is entered only if the TUC is operationally equipped, and subsequent reentries are made for each operationally equipped TUC until all such TUCs have been processed.

9.41 The DUFROSDS and DUFROSTC routines are entered to update, respectively, the DUS or TUC out-of-service frame lamp for all DUSs and TUCs in the office. If the DUS/TUC is operationally equipped, but with an active in-progress frame request; or, if it is not operationally equipped, the out-of-service frame lamp is not updated.

9.42 The DUFRSOS1 and DUFRTOS1 routines are entered respectively to determine which of the operationally equipped DUSs and TUCs are out of service for the output message "OP:OSS units." These routines check all DUSs and TUCs and, if an equipped unit is found out of service, an exit is made to the user to include that unit in the output message. The user then reenters the program and processing continues until all DUSs/TUCs have been checked for out-of-service conditions. This data is used for the output message "OP:OOS units" that is issued routinely every half-hour, in response to the input message "OP:OOS units," and the MCC request of depressing either the DATA UNIT SELECTORS or the TAPE UNIT CONTROLLERS processor equipment status key.

DUFRTADM Description

9.43 The DUFRTADM pident contains the test administration routines for the various tests performed by DUFR. For some D-level interrupts and maintenance interjects, AUFR and DUFR are unable to determine which units are at fault. For these situations, AUFR or DUFR enter DUFRTADM to make the determination. The DUAD program enters DUFRTADM when it detects errors during its normal course of monitoring the auxiliary data system.

9.44 The DUFRTBL routine is entered by AUFR and other DUFR routines to determine what type of DUS-to-TUC communications failure has occurred. Failure indicators are obtained from the error summary register and the generate control pulse (GCP) register save bins. Routine DUFRTBL verifies that these registers are operating properly and that the error indication signifies a valid error. If no failure is detected, an error count is incremented that keeps a record of the number of times DUFRTBL is entered for transient errors. After the failure type has been defined, program control is transferred to the appropriate test routine shown below:

(a) DUFRSVRF—Select verify register failure

(b) DUFRASWF—All-seems-well failure

(c) DUFRPKRF—Reply bus parity failure

(d) DUFRPKCF—Control bus parity failure.

9.45 The DUFRDUC routine is entered from the test analysis routines in the DUFRTBL routine. The test routine determines the suspect TUC. The DUFRDUC routine determines if the TUC needs testing, in which case there is a transfer to the select verify register tests in DUFRTSTS, or DUFRDUC assumes that the TUC is at fault and transfers to the DUFRRMDC routine in DUFRPDCU to remove the TUC from service.

9.46 An entry to the DUFRDS1 entry signifies that a DUS diagnostic phase 1 (no specific helper

unit needed) is requested. The DUFRDS1 routine updates the status to indicate DUS removal and the diagnostic requested by DUFR, then transfers to the DUS remove routine DUFRRMDS in DUFRPCDU. An entry to the DUFRDS2 routine indicates a request for a DUS diagnostic phase 1 and 2 (a helper unit other than the TUC presently configured to the DUS is needed). The status is updated and control transferred to the DUS remove routine.

9.47 The DUFRRTRY routine is entered when a test has failed but the faulty unit (DUS or TUC) cannot be identified. This routine repeats the failing test using another DUS or obtains another available TUC and tests with the same DUS. Depending on the results of these tests, control exits from this routine to remove either a DUS or a TUC. If all the tests pass, control goes to the Retry All Tests Passed routine (DUFRRATP in the DUFRPCDU pident).

9.48 The DUFRTATP routine is entered when a DUS-to-TUC communications fault has been detected and the test run in DUFRTBL was an ATP. If a previous retry failed with this TUC and the other DUS, then this TUC is removed. If a previous retry failed with this DUS and another TUC, then this DUS is removed. If tests performed in this routine are also an ATP, then a transient error is recorded.

9.49 The DUFRSVTP routine is entered when the DUS-TUC select/select verify test has been run and passed. These tests consist of sending specific TUC K-code data over the bus and ascertaining that only the correct TUC responds. The first time this test passes, a transient fault is assumed and no action other than to record the error is taken. The second time it passes, a fault in the DUS is assumed and that DUS is removed from service.

9.50 The DUFRIVOI routine is entered from the DUAD program when an operational interject is observed but no DUS is found reporting on an operational interject. On completion of an operation dispensed by DUAD, operational interjects are set in the TUC and reported to the DUS that in turn reports the operational interjects to central control. In this case DUAD observed the operational interject but, in verifying the work completion, no DUS had its operational interject bit set. The DUFRIVOI routine then attempts to isolate the faulty unit by calling the AU bootstrap program. If the AU bootstrap fails to rectify the problems, an immediate transfer is made to the Processor Configuration Recovery Program (PCRV) and the processor configuration circuit is activated which initiates a change in the processor configuration. This is the most severe action that is taken by the DUFR program and normal call processing may be affected.

9.51 The DUFROBRF routine is entered from DUAD when a write into a TUC fails the output buffer register verification check twice without causing an interrupt. This routine checks the original DUS-TUC combination with a 0 through 1's test, a 1 through 0's test, an alternating 0's test, and an alternating 1's test. If an interrupt is caused by the test, interrupt processing is allowed to process the fault. If the test fails and still no interrupt is generated, the DUS is removed for a diagnostic using a different TUC helper unit. If all tests pass, the DUS is assumed to be good and the original TUC is removed from service with a diagnostic request. In any case, the test results are stored for a later report data printout.

9.52 The DUFROIDS routine is entered from DUAD to remove a DUS from service following a maintenance type of operational interject. Some operational interjects within the DUS suggest a maintenance-type problem within the DUS. The DIFROIDS routine does some setups, checks that the DUS can be removed, saves additional report data, and transfers to the DUFRNEUT routine to have the DUS removed from service and the status registers updated.

9.53 The DUFRBLDC routine is entered from DUAD to remove a TUC from service following base level checks made by DUAD. After some setups are made and report data is saved, control passes to the DUFROIDC routine.

9.54 The DUFROIDC routine is entered from DUAD when DUAD recognizes a TUC maintenance-type interject in the TUC or from an entry from DUFRBLOC. The TUCs are not permitted to set system maintenance interjects, so a TUC maintenance interject bit is set in the TUC but the operational interject bit is set in the system. The DUAD program recognizes the TUC maintenance interject and transfers to DUFROIDC. Some setups are performed and report data is saved. Control is then passed to the DUFRNEUT routine in PCDU to remove the TUC from service and update the software status.

9.55 The DUFRTADM pident also contains several short routines, such as

(a) The reporting of TUCs not equipped—no action taken

(b) The printing of data about TUCs and DUSs—data is saved in print bin

(c) The returning of a DUS to the normal mode—after testing.

DUFRTSTS Description

General

9.56 The DUFRTSTS pident contains all the auxiliary data system tests that are needed by and used by the Data Unit Fault Recovery Program (DUFR). This includes the internal logic of the DUS, the DUS-TUC communications circuitry in the DUS and TUCs and the data unit bus between them, and the internal logic of the TUC. Extensive tests are provided for testing and verifying communication paths with little being provided for checking internal unit logic. This is done because the communication paths or buses are more susceptible to noise than internal logic, and therefore more likely to exhibit transient type errors. It is desirable to filter these errors as quickly as possible, thereby reducing the recovery time. Errors in internal logic are more likely to be of the hard variety and are detectable by error detection circuitry, usually with enough resolution to pinpoint the faulty unit.

9.57 There are no explicit tests provided to test the TUCs. To determine if a TUC is faulty, the following methods are used:

(a) Evaluation of the TUC internal error registers

(b) Error-free unsuccessful job completions detected by DUAD

(c) Error analysis techniques

(d) Alternate route testing using the DUS-TUC communication tests and different DUS-TUC configurations.

9.58 The tests in DUFRTSTS may be called by any one of three pidents (DUFRTADM, DUFRDGNI, or DUFRTTYI) that contain the control program for executing the tests.

9.59 The DUFRTADM pident calls the tests in DUFRTSTS after an auxiliary data system D-level interrupt or maintenance interject. However, not all tests are called. If the error is determined to be in the DUS, only the DUS internal register test is called. If the error is in DUS-TUC bus communications, then both the DUS internal register test and one of the four DUS-TUC bus communications tests are called. The DUS internal register is called only once but the DUS-TUC bus communication test may be called as many as three times.

9.60 The DUFRDGNI pident calls all the tests in DUFRTSTS after a DUS diagnostic that is completed with an ATP. The DUS diagnostic is scheduled to be executed on a routine basis; therefore, the DUFRTSTS routines are executed routinely.

9.61 The DUFRTTYI pident calls the tests in DUFRTSTS on a demand basis initiated by the TTY input message "TEST:DUS A:RPT B:TUC C, TSTNO D!" where:

A = DUS member number to be tested

B = Number of times to repeat the test

C = Member number of TUC helper unit

D = Tests to be executed.

This TTY input message allows any combination of tests to be executed on any DUS-TUC combination any number of times. For more information, refer to Input Message Manual IM-4A001.

Test Routines

9.62 The DUS internal register test routine is entered at entry DUFRDUSS to test the DUS internal register complex. The following registers are tested.

- Error source register
- Input buffer register
- Inhibit register
- Output buffer register
- Select verify register
- Maintenance and control register
- Output address register

- Operational interject register.

The following set of tests is performed on each of the above registers and the data is saved at the time of the D-level interrupt or maintenance interject.

(1) Write original data

(2) Write the complement of the original data

(3) Read and compare

(4) Write original data

(5) Read and compare.

9.63 The DUFRDUSS entry is called after an AU D-level interrupt or when an AU maintenance interject has occurred because of a failure in the auxiliary data system community. It is also called as part of the DUFR deferred tests that are executed after a nonpartial ATP DUS diagnostic and on request by the "TEST:DUS" input message. Control returns to the calling program upon completion.

9.64 The data unit select/select verify register (SVR) test routine is entered at entry DUFRSVRF because of a DUS-TUC select/select verify failure within the auxiliary data system community. Only the DUS-TUC communications route that existed at the time of failure is tested. The following circuitry and/or functions are included in the tests.

(a) DUS select verify register

(b) Select verify register 1/N checker

(c) Select verify register inhibit input gating function

(d) Inhibit TUC select pulses function

(e) Select verify register inhibit reset function

(f) TUC for multiple enabling (selecting) TUC

(g) DUS for null enabling (selecting) TUCs

(h) DUS for transporting TUC enables (select pulses).

9.65 A second entry to the data unit select/select verify register test routine is DUFRSVRS. This entry is used to repeat a particular data unit select/select verify test using a different DUS-TUC configuration in an attempt to isolate the faulty unit. This entry is also used for the deferred DUFR tests and on request by the "TEST:DUS" input message.

9.66 The data unit address and write bus all-seems-well (ASW) test routine is entered at entry DUFRASWF. Pident DUFRTADM enters this routine because of a DUS-TUC ASW failure within the auxiliary data system community, and returns to DUFRTADM upon completion. This routine tests the following circuitry and/or functions:

(a) Maintenance and control register ASW flip-flop and the maintenance and control register to error source register ASW function

(b) Maintenance and control register ASW flip-flop inhibit reset function

(c) Maintenance and control register ASW flip-flop inhibit input gating function

(d) Normal/maintenance read/write mode of the data unit address bus

(e) Address bits part of the data unit address bus

(f) Address parity generator in the DUS

(g) Address parity checker in the TUC

(h) Address decoder in the TUC

(i) Data unit data write bus

(j) Data parity generator in the DUS

(k) Data parity checker in the DUS

(l) Ability of the DUS to poll the TUC

(m) ASW generator in the TUC

(n) ASW reply from TUC to data units (part of DU reply bus).

9.67 The data unit address and write bus ASW test routine has a second entry DUFRASWS. This entry is used by DUFRTADM to repeat a particular data unit address and write bus test using a different DUS-TUC configuration in an attempt to isolate the

faulty unit. This entry is also used for the DUFR deferred test and on request by the "TEST:DUS" input message.

9.68 The data unit reply bus test routine is entered at DUFRPKRF. The DUFRTADM pident enters this routine because of a DUS-TUC reply bus parity failure within the auxiliary data system community and returns there upon completion. This routine tests the following circuitry and/or functions:

(a) Output address register and its parity checker

(b) Output buffer register and its parity checker

(c) Output address register and output buffer register inhibit reset functions

(d) Output address register and output buffer register inhibit input gating function

(e) Output address register and output buffer register selection circuitry

(f) Data unit reply bus

(g) TUC output (+gating) from the TUC data register and the address register

(h) DUS input (+gating) to the output buffer register and the output address register

(i) Parity reply to the output buffer register and the output address register

(j) Not gating into the output address register when gating into the output buffer register

(k) Not gating into the output buffer register when gating into the output address register.

9.69 The data unit reply bus test routine has a second entry DUFRPRPS. This entry is used by data unit FRTADM to repeat a particular DU reply bus test, using a different DUS-TUC configuration in an attempt to isolate the faulty unit. This entry is also used by the DUFR deferred test and on request by the "TEST:DUS" input message.

9.70 The data unit control bus test routine is entered at entry DUFRPKCF. The DUFRTADM pident enters this routine because of a DUS-TUC control bus parity failure within the auxiliary data system community, and returns to DUFRTADM upon completion. This routine tests the following circuitry and/or functions:

(a) TUC maintenance interject bit

(b) TUC operational interject bit

(c) Read/write bits

(d) Data unit control bus parity bit

(e) Data unit control bus parity checker in the DUS

(f) Data unit control bus and parity reply

(g) TUC output (+gating) from the TUC control bus register

(h) TUC output (+gating) to the bit positions in paragraph 9.70(a) through (d)

(i) Data unit control bus parity generator in the DUS

(j) Operational function to the bit position in paragraph 9.70(a) through (d)

(k) Inhibit reset function to the bit positions in paragraph 9.70(a) through (d)

(l) Inhibit gating functions to the bit positions in paragraph 9.70(a) through (d)

(m) Control bus register in the DUS.

9.71 The data unit control bus test routine has a second entry, DUFRPCTS. This entry is used by DUFRTADM to repeat a particular data unit control bus test using a different DUS-TUC configuration in an attempt to isolate the faulty unit. This entry is also used by the DUFR deferred test on request by the "TEST:DUS" input message.

9.72 The post-test initialization service routine is entered at entry DUFRPSTT. This routine is entered after a test routine has been run to initialize the DUS to pretest conditions. This properly initializes the DUS for execution of the next test subroutine, and leaves it in a stable and safe state if no more tests are to be executed.

9.73 The DUFRTSTS pident also contains a subroutine to be used by the test routine when a

test within a test subroutine has failed. This routine, which is entered to do post-test initialization and cleanup, does the following:

(a) Saves the failing test data

(b) Initializes the DUS hardware to equal the DUS initialization before test subroutine execution

(c) Returns back to the failing test subroutines or returns directly to the user.

DUFRTTYI Description

9.74 The DUFRTTYI pident is designed to process all DUS/TUC configuration changes that do not require a diagnostic. Situations that lead to entering this pident are:

(a) Removing a TUC before TTY-requested diagnostics

(b) Removing a TUC before restoral

(c) Removing a TUC for routine exercise diagnostic

(d) Switching a TUC to the other DUS during routine system normalization

(e) Switching TUCs to the other DUS before removing a DUS

(f) Switching TUC to other DUS prior to diagnosing an in-service DUS

(g) Switching TUC after restoral of a DUS

(h) Switching TUC to other DUS prior to setting up an off-line configuration

(i) Switching TUC to the other DUS prior to nonemergency removal of an AUB

(j) Removal or unconditional restoral of a DUS

(k) Removal or unconditional restoral of a TUC.

9.75 The nonemergency switch/remove routine DUFRNESR is entered to effect a soft reconfiguration ("soft" meaning that no jobs are affected) of the auxiliary data system community. It is initiated by DUAD after DUAD has recognized a reconfiguration request flag set for the affected TUC. The request may be to either remove a TUC from service or to switch a TUC to the other DUS. If the AU community is frozen because of some long job in progress, the request to reconfigure is denied until that job is complete.

9.76 The reconstruct auxiliary data system routine is entered to reconstruct and restart the auxiliary data system community after a reconfiguration. This routine is entered at entry RMV_RST_DS_DC. The auxiliary data system hardware and software are reconstructed, and the status indicators are updated.

9.77 The request to remove or unconditionally restore a DUS routine is entered from the Manual Input Request Administration (MIRA) program. Its entry is DUFRMRQS. For any restore request the AUB must be in service. For any remove request the other DUS must be in service, and no in-service TUC can have bus trouble marked toward that DUS. The remove is done by first reconfiguring all in-service TUCs to the other DUS and then removing the requested DUS in hardware and software. A restoral is done by first restoring the DUS in hardware and software. Then the reconstruct auxiliary data system routine is used to normalize the entire auxiliary data system community.

9.78 The request to remove or unconditionally restore a TUC routine is also entered from MIRA. Its entry point is DUFRMRQT and at least one DUS must be in service to honor the request. For a remove request the TUC must be unassigned, and the TUC is removed in both hardware and software. The reconstruction routine is then entered to normalize the remaining units. A TUC restoral request is honored by first connecting the TUC to an available DUS in hardware and software. The reconstruct routine is then called to normalize the new auxiliary data system community.

9.79 The Run Deferred DUS Tests After Test Message routine is designed to process the deferred DUS tests. The "TEST:DUS" message first enters MIRA where a general buffer table is acquired and filled with relative data for use by this routine. The DUFRTTYI entry DUFRDFOR is then entered. It checks the validity of test numbers specified and, if they pass, transfers control to the prediagnostic initializer in the DUFRDGNI. Pident DUFRDGNI

sets up the proper configuration, makes additional validity checks, and initiates the individual tests. The DUFRDGNI pident then reenters DUFRTTYI at DUFRSUCC which marks the DUS status before testing so that the DUS can be returned to its original tests, and takes a segment break before going off to run the tests. If the DUFRDGNI validity checks fail, DUFRTTYI is reentered at DUFRDENY and the requested tests are denied. The DUFRDFST entry is also used by DUFRDGNI if any of the tests fail. The test data is saved in case the tests are to be repeated, and test results are not printed until later tests are complete. The DUFRDGNI pident reenters DUFRTTYI at DUFRDFAT for the case where all tests passed. Routine DUFRDFAT also has the option of repeating the test, even though they are ATP, when so requested in the test message.

9.80 The deferred test may be stopped by the office personnel or it may be aborted by the MACP program. In either case the DUFRABT routine is entered. The results are reformatted and a transfer is made to the DUFRDGNI print routine that outputs a TTY message indicating the results.

10. 1A PROCESSOR F-LEVEL FAULT RECOVERY PROGRAM—PFLR

INTRODUCTION

10.01 The PFLR program performs a filter function for all peripheral unit bus-related faults as well as the fault recovery tasks associated with the master control console (MCC), input/output units (IOUs), and ▶input/output processor (IOP) frames.◀

10.02 Functions performed by PFLR are:

(a) Filters all F-level interrupts for switching systems using the 1A Processor. The filtering process routes program control to the appropriate fault recovery program according to the type of F-level interrupt.

(b) Recovers from F-level interrupts and faults detected on base level if they are associated with the MCC, IOU, or IOP. The recovery process finds a working peripheral configuration and isolates the faulty equipment.

(c) Performs short-term error analysis of high-frequency transient faults. (The Error Analysis Program [ERAP] provides an additional facility for manual analysis of system trouble data that may be used to aid in resolving transient faults.)

(d) Provides test, diagnostic, and service routines which may be used by PFLR and also by other maintenance programs to interface with input/output (IO) and MCC.

(e) Administers the use of a diagnostic when PFLR removes an MCC, IOU, or IOP from service.

(f) Does IOU and IOP fault recognition exercises periodically and diagnostic exercises on a noninterrupt basis, ie, base level processing.

▶MCC/IO◀ AND PERIPHERAL UNIT BUS CHARACTERISTICS

A. General

10.03 The MCC and IOs are common system peripheral units. The MCC and IOs have simplex controllers (ie, nonduplicated controllers). The MCC and IO channels are addressed by the coded enable method (K-code enabled) and communicate with the central controls over the duplicated peripheral unit bus.

Note: Equipment characteristics of the MCC, IOU, IOP, and peripheral unit bus are presented briefly in this section to aid in the understanding of the PFLR description.

10.04 For more detailed information on the MCC, IO, IOP, and peripheral unit bus, refer to the following sections:

SECTION	TITLE
234-110-000	Processor Peripheral Interface Frame and Master Control Console—Description
234-110-001	Processor Peripheral Interface Frame and Master Control Console—Theory
254-201-040	Input/Output Frame—Description
254-201-041	Input/Output Frame—Theory
254-201-042	Input/Output Processor Frame—Description

254-201-043 Input/Output Processor Frame—Theory.

B. MCC Description

10.05 The MCC is a dual matrix system containing a power control switch matrix (alarm matrix) and a control and display matrix (lamp and key matrix). The individual rows of each matrix may be accessed via program control over the peripheral unit bus.

10.06 The MCC contains maintenance check circuitry. Failures detected by this circuitry will cause the all-seems-well (ASW) signal to be inhibited and later causes an F-level interrupt. This fault detection circuitry consists of:

(a) K-code match checks—A check failure occurs when the MCC does not detect a K-code match.

(b) Received data checks—A check failure occurs when the MCC detects a parity error on data and address.

(c) Access of one row of a matrix (not both matrices)—A check failure occurs when MCC detects matrix access error; ie, operation (OP) code and row selection code failed to select one row of one matrix.

10.07 In addition, F-level failures occur when the central control detects an answer parity failure from the MCC or invalid reply data, ie, reply mismatch.

10.08 The MCC peripheral busing unit, all logic circuitry, and the central control interface circuitry to the MCC are housed in the processor peripheral interface frame.

C. IOU/IOP Description

10.09 The old type ⏵IO (IOU)⏴ is comprised of an input/output unit selector (IOUS) and eight input/output unit controllers (IOUCs). That is, the IOU is a multilevel access unit. Each IOUC works with the IOUS to form an input/output channel (IOC) which can accommodate up to three ports, ie, IO terminals such as a TTY, cathode ray tube, data set, etc. (The ports of a channel are slaved to each other—each port receives the same information.)

10.10 The new type ⏵IO (IOP)⏴ is comprised of an IOUS and either one or two input/output microprocessors (IOMP). Each IOMP may support up to 8 line units (or IOUCs)—a total of 16 line units for a fully equipped ⏵IOP.⏴ Each line unit (or IOUC) works with an IOMP and an IOUS to form an IOC that can accommodate up to three ports, ie, IO terminals such as a TTY, cathode ray tube, data set, etc. (The ports of a channel are slaved to each other—each port receives the same information.) Only asynchronous communications is supported by the line unit.

10.11 A variation of the IOP replaces the second IOMP with a fan-out board driving up to eight peripheral controllers. The peripheral controller supports only port and synchronous protocol communication.

10.12 The busing unit (circuitry containing peripheral unit bus drivers and receivers) are shared by two IOUSs; ie, there are two IOUSs in an IOU/IOP frame.

10.13 For the IOU, the IOUS status register of 4 bits records the access mode bus configuration flip-flops. An IOUC status register (one per IOC) records the channel state, data about maintenance requests, and other data. Any status read yields the 4 bits of the IOUS status register plus the 17 bits of the IOUC selected. In addition, there is a readable error summary register.

10.14 For the IOP, there are three types of status registers: a 24-bit status and error source register combined for the IOUS, a 16-bit status and error source register combined for each IOMP, and a 24-bit status and error source register combined for each IOUC. There is also a 24-bit channel maintenance register for each IOUC. Each of these registers may be accessed by a status read.

10.15 The IOUSs (both types) contain most of the IO system's fault detection circuitry. The failure of any one of the following checks inhibits an ASW signal and results in an F-level interrupt:

(a) K-code match checks—IO detects K-code match.

(b) Bus parity checks—IO detects a parity error on data and address.

(c) Validity checks of instructions and mode—IO detects incorrect character usage or incorrect setting of mode flip-flops.

(d) Correct performance of reply bus parity generating circuits—IO parity generating circuits send invalid parity data.

10.16 In addition, F-level interrupts may be caused by central control detecting answer parity and reply mismatch errors.

D. Peripheral Unit Bus Configurations

10.17 The central control (Table E) routing control flip-flops (PUBO, PUBA, PUBT and PUBR) with the peripheral unit controller (Table F) routing control flip-flops (RO, S0, S1) may be changed to control the peripheral unit bus configuration. The maintenance flip-flops in the controllers are set to change the access mode from the normal mode to the maintenance mode.

10.18 Normally, the active central control is sending and receiving on one bus while the standby central control is sending and receiving on the other bus. The peripheral unit (those having simplex controllers) normally is receiving on one bus and sending back on both buses. Thus, both central controls receive the same peripheral unit reply, and the parity over that reply is matched at the central controls to see if both central controls obtain the same information.

10.19 The processor peripheral interface frame, a common interface point between the 1A Processor and the peripheral community, contains a bus

TABLE E

PERIPHERAL UNIT BUS CONTROLS LOCATED IN CENTRAL CONTROL

CONTROL FLIP FLOPS (NOTE 1				PERIPHERAL UNIT ENABLE ADDRESS (NOTE 2)	PERIPHERAL UNIT WRITE (NOTE 2)	PERIPHERAL UNIT REPLY	
				SENDS ON BUS	SEND ON BUS	RECEIVE ON BUS	
0	0	0	0	0	0	0	
0	0	0	1	1	1	1	
0	0	1	0	0	0	0	Active
0	0	1	1	1	1	1	
0	1	0	0	0 & 1	0 & 1	0	Central
0	1	0	1	0 & 1	0 & 1	1	
0	1	1	0	0 & 1	0 & 1	0	Control
0	1	1	1	0 & 1	0 & 1	1	
1	D	D	D	0 & 1	0 & 1	0 & 1	
0	0	0	0	1	1	1	
0	0	0	1	0	0	0	
0	0	1	0	X	X	0	Standby
0	0	1	1	X	X	1	
0	1	0	0	X	X	1	Central
0	1	0	1	X	X	0	
0	1	1	0	X	X	0	Control
0	1	1	1	X	X	1	
1	D	D	D	X	X	0 & 1	

Note 1: 0 = Reset, 1 = Set, D = Don't Care

Note 2: X = No Bus Transmission

TABLE F

PERIPHERAL UNIT BUS CONTROLS LOCATED IN PERIPHERAL UNIT CONTROLLERS

CONTROL FLIP-FLOPS (NOTE)			PERIPHERAL UNIT WRITE ADDRESS	PERIPHERAL UNIT WRITE	PERIPHERAL UNIT REPLY
RO	SO	S1	RECEIVE ON BUS	RECEIVE ON BUS	SEND ON BUS
0	0	0	0	0	Don't send
0	0	1	0	0	1
0	1	0	0	0	0
0	1	1	0	0	0 & 1
1	0	0	1	1	Don't send
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	1	1	0 & 1

Note: 0 = Reset, 1 = Set

loop-around circuitry that is used in recovery from bus-sensitive faults.

FAULT TYPES/CLASSIFICATIONS

10.20 The PFLR program is capable of distinguishing three fault types. According to the retrial of a failed order, fault types are as follows:

(a) ***Hard Faults:*** Repeatable, consistent, permanent-type faults. If a failed order is retried several times alternately using the original bus, controller, and central control configuration and a new configuration but the order fails repeatedly, the fault is a hard fault.

(b) ***Marginal Faults:*** Initially reproducible faults, but they disappear during recovery procedures. If a failed order is retried on a new configuration and passes, then tried again on the original configuration and passes, the fault is a marginal fault.

(c) ***Transient Faults:*** Nonreproducible faults, intermittent-type faults. If a failed order is repeated using the original configuration and the order passes, the fault is a transient fault.

10.21 When PFLR can recover from a fault by successfully repeating a failed order over a new configuration, the fault is classified as a recoverable fault. However, if PFLR recovery action fails and determines that the MCC or IOUS is to be removed from service, the fault is classified as an unrecoverable fault. Any hard fault that PFLR cannot localize to a failed subsystem is classified as an unresolved fault. Sometimes a fault may be recoverable but also be unresolved.

10.22 The major portion of PFLR's fault recovery and isolation routines are concerned with ➧ MCC/IO➧ recovery from hard faults. However, for transient or marginal faults, PFLR attempts to resolve MCC or IO faults by using short-term error analysis. PFLR's short-term error analysis procedures are used when the transient or marginal fault has a high frequency of occurrence; this analysis results in an action recommendation. Data about transient/marginal faults having a low frequency of occurrence is stored in a common system ERAP and may be retrieved in resolving actions to be fulfilled by other maintenance programs.

PFLR INTERFACES

A. General

10.23 The PFLR program interfaces with other programs are shown in Fig. 22. A brief description of each program interface follows.

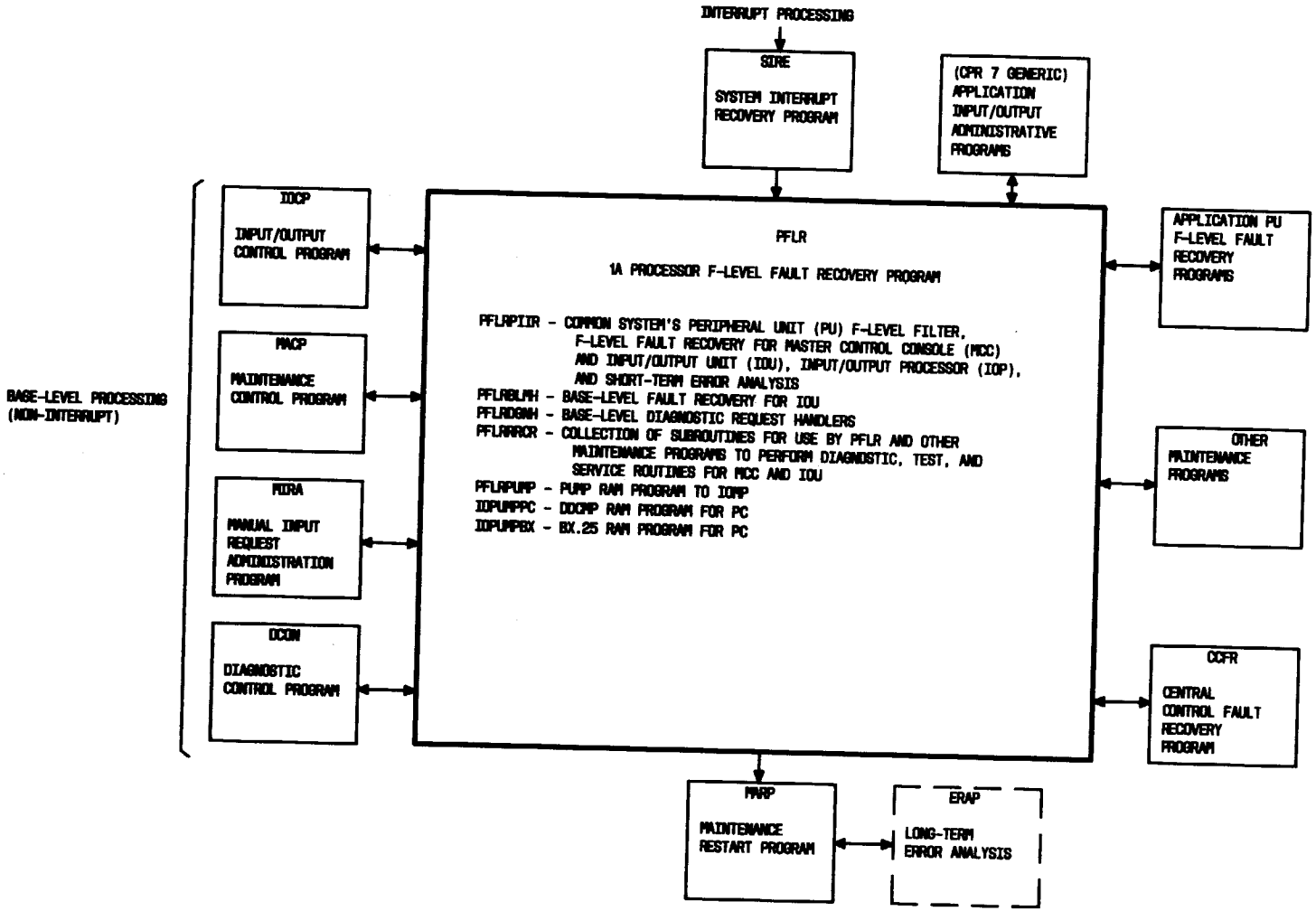


Fig. 22—IA Processor F-Level Fault Recovery Program (PFLR)

B. System Interrupt Recovery Program—SIRE

10.24 The initial entry into PFLR is made by SIRE. The SIRE program, on detection of any F-level interrupt, stores data pertinent to the peripheral unit interrupt source in an F-level interrupt bit (memory) and then transfers to PFLR. It is PFLR's responsibility to filter all F-level interrupts and to determine the fault recovery program that is to recover from the fault.

C. Application Fault Recovery Programs

10.25 When PFLR determines through the common system's F-level filter routine that the interrupt source is an autonomous unit or a central pulse distribution (CPD) enabled unit, but not MCC/IO, the PFLR transfers to the appropriate application peripheral unit fault recovery program. The PFLR program also transfers to an application fault recovery program for peripheral error analysis after classifying the MCC/IO problem and determining a recovery action. If the action is to remove a peripheral unit bus and the application fault recovery program concurs, the control is not returned to PFLR. Otherwise, PFLR regains control to achieve the recovery action.

D. Central Control Fault Recovery Program—CCFR

10.26 When PFLR determines that mismatching of information is occurring in the central controls, PFLR (by the final disposition routine) transfers the recovery function to CCFR.

E. Maintenance Restart Program—MARP

10.27 After PFLR executes final actions, PFLR transfers to MARP. With the transfer, PFLR indicates the return option to be used by MARP to return to normal base level processing. (Return options may unwind and reexecute the interrupted instruction, roll back to a safe point in the interrupted program, or return to a reference point in the interrupted program.)

> ***Note:*** The above PFLR interfaces occur during F-level (interrupt) processing. The remaining interfaces below occur during normal base level (noninterrupt) processing.

F. Input/Output Control Program—IOCP

10.28 When scheduled by the executive control program, IOCP, upon detection of maintenance requests and invalid polling responses, enters the IO fault recovery pident in PFLR to resolve the fault source.

10.29 The PFLR program interfaces with IOCP to report IOC removed and restored from service. Thus, IOCP can reroute messages to backup channels for these out-of-service channels. The PFLR program also uses an IOCP routine to verify that a backup channel is in service before removal of a channel.

G. Maintenance Control Program—MACP

10.30 The executive control program several times daily passes control to MACP to enter PFLR for the execution of an IO exercise routine for ⧫both the IOU and IOP.⧫ The PFLR program checks the error source register. If any errors have occurred since the last exercise, PFLR removes the applicable IOUS from service and calls the PFLR diagnostic request handler (routine in diagnostic pident) to set up an MACP job for a demand diagnostic.

10.31 Once a day, MACP enters the diagnostic pident of PFLR to do routine diagnostic exercises on both the MCC and the IO.

10.32 Also, PFLR interfaces with MACP for reporting units removed from service, ie, out-of-service reports. Finally, MACP interfaces with PFLR to remove MCC/IOs from service and to unconditionally restore (MCC/IOs) to service.

H. Diagnostic Control Program—DCON

10.33 The DCON program enters the diagnostic pident of PFLR for the execution of the diagnostic handling routines before the execution of the Master Control Console Diagnostic Program (MCDG), the Input/Output Unit Diagnostic Program (IODG), or the Input/Output Processor Diagnostic Program (I2DG) diagnosis.

I. ⧫Application IO Administrative Programs

10.34 For the CPR7 generic, the addition of this interface to the PFLR program allows fault recovery on application-administered IO channels.

This interface also allows PFLR to perform fault recovery on as many as 32 IOP frames (as compared to 4 frames allowed for generics before CPR7).

10.35 The application interface allows the application programs to administer IO channels instead of having the common IOCPs administering the IO channels. The application programs are notified when application-administered IOs are removed from and restored to service, when diagnostics are requested on the IOs, and when faults are detected on the IOs.

PFLR PIDENTS

10.36 The PFLR pidents (Fig. 23) and their associated functions are:

(a) PFLRPIIR—Filters F-level interrupts for entire switching systems using 1A Processor and recovers a working peripheral configuration from faults related to the MCC or an IO. Pident PFLRPIIR uses a short-term error analysis routine to resolve transient-type faults exceeding an error threshold. Permanent-type faults are generally resolved by the MCDG, the IODG, or the I2DG.

(b) PFLRBLMH—Contains routines used in resolving faults detected during base level (noninterrupt) processing. The IOCP program may enter PFLRBLMH either to take remedial action on a maintenance request or to resolve the cause of a false polling service request. The MACP program enters PFLRBLMH periodically to do an IO routine exercise. Also, PFLRBLMH contains other maintenance subroutines.

(c) PFLRDGNH—Performs hardware/software initialization and final handling requirements for running the IO and MCC diagnostics. The MACP program enters PFLRDGNH periodically to do routine diagnostic exercises.

(d) PFLRRRCR—Contains a collection of subroutines used by the diagnostics and any manual request (via TTY) to remove/restore a unit, a subunit, or bus from/to service, respectively.

(e) PFLRPUMP—Contains a routine for pumping the random access memory (RAM) of the IOMP. This pident also contains the data used to pump the RAM.

(f) IOPUMPPC—Contains data for pumping RAM of peripheral controller with code to handle digital data communication message protocol link level protocol.

(g) IOPUMPBX—Contains data for pumping RAM of peripheral controller with code to handle BX.25 link level protocol.

A more detailed description of each pident follows, except for PFLRPUMP, IOPUMPBX, and IOPUMPPC.

PFLRPIIR DESCRIPTION

A. General

10.37 The PFLRPIIR pident is responsible for filtering all F-level interrupts for the entire system and recovering from those faults related to the MCC or an IO. Specifically, PFLRPIIR performs the following functions:

(a) Filters F-level interrupts in all switching systems using the 1A Processor and passes control to the appropriate fault recovery program

(b) Retains responsibility for fault recovery only for interrupt causes related to the MCC or an IO

(c) Attempts isolation of the faulty equipment (bus, unit, or subunit) for hard faults

(d) Performs short-term error analysis for transient or marginal faults

(e) Attempts isolation of babbling on peripheral unit bus in the processor area.

B. Peripheral Unit Filter Routine

10.38 After loading the F-level interrupt bin (memory) with pertinent fault data, the System Interrupt Fault Recovery Program (SIRE) enters the PFLRPIIR filter routine at entry point PFLRFLEV. The filter routine determines which fault recovery program is to recover from the F-level interrupt for the following decisions:

(a) The interrupt was generated from an autonomous or nonautonomous source.

Note: An autonomous peripheral unit failure is attributed to an internal peripheral unit trouble in an autonomous unit whereas the

nonautonomous peripheral unit failure is attributed to a communication problem between central control and a peripheral unit, or an internal trouble in a nonautonomous unit.

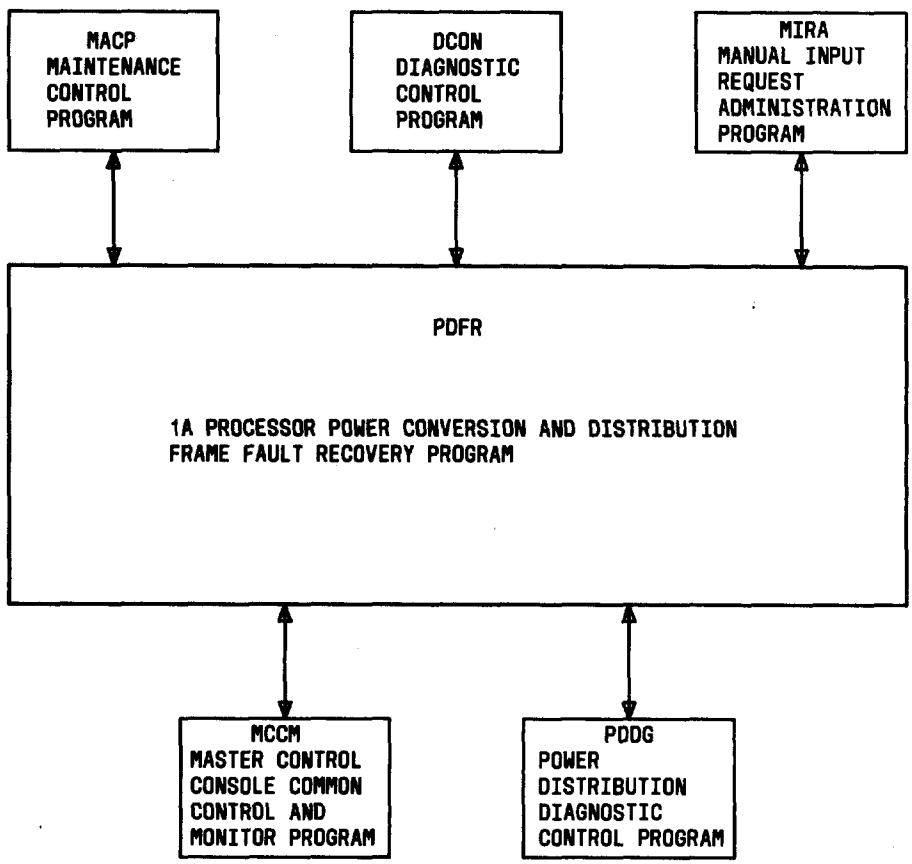
(b) The failure occurred during the execution of a central pulse distributor (CPD) enabled order or a coded enabled order (ie, K-code enabled).

(c) The MCC or an IOC (IOUS and one IOUC/IOMP) was addressed or was not addressed on the failing order.

For failures caused by autonomous peripheral units, by nonautonomous units other than MCC/IOs, or by a CPD enabled order, PFLR passes control to an appropriate application fault recovery program. Only failures related to the MCC or IO are kept under the control of the PFLRPIIR pident.

C. ⧫MCC/IO⧫ F-Level Recovery Routines

10.39 The PFLRPIIR recovery routine uses the data stored by SIRE in the F-level interrupt bin to perform common (applicable to the MCC and IO) recovery initializations that include the storing of pertinent data about equipment in service, initializing tables to be used by fault analysis, and deciding whether the fault source is the MCC or IO. Then unique initialization routine and test routines are entered since the MCC, the IOU, ⧫and the IOP⧫ have their own peculiarities. An MCC first-look initialization routine determines if the software is blamed for the ASW, answer parity, or reply mis-



⧫Fig. 23—Primary PFLR—Pident Interfaces⧫

match failure. An IOC first-look initialization routine (there are three of these routines, one for each type of IO frame) tries to establish a failing order or a test sequence to be used in the main recovery routines.

10.40 The main recovery routines (unique to the MCC or IO) determine the following:

(a) The failure was caused by the absence of an ASW signal, by the detection of an answer parity error, or by the detection of a reply mismatch error (L-register reply mismatch in central controls).

(b) The fault is transient/marginal (nonrepeatable or temporarily repeatable) or the fault is hard (repeatable and permanent).

10.41 If the failure is determined to be repeatable (hard), PFLR performs a series of configuration and test routines, ie, a deductive search to isolate the fault source to the peripheral unit bus community, the MCC/IO unit itself, the busing unit, or the central control.

10.42 If repeated trials of controller/bus configurations indicate a major bus problem (ie, trouble in the active or standby peripheral unit bus), results are set up in data words and passed to the appropriate application fault recovery program. This application fault recovery program determines by bus loop-around tests if the failure exists in a portion of the peripheral unit bus or just in the MCC/IO busing unit. If a peripheral unit bus problem is determined, the application program processes and passes the results on to MARP. However, if a busing unit trouble is determined, the application fault recovery program passes control back to PFLR at entry point PFLRACPT for the recovery action disposition routine (paragraph 10.45). All other decisions of recovery action are also passed to the applications recovery program, but control is always returned to PFLRACPT.

D. Short-Term Error Analysis

10.43 ▶The short-term error analysis routine is entered at entry point PFLRANYL. For the CPR5 and CPR6 generics, the PFLR program maintains a short-term error history in a 4-entry table. For the CPR7 generic, the PFLR program maintains a short-term error history in an 8-entry table.◀ A record is made each time an interrupt is classified as a transient or a unit/subunit is removed and a diagnostic requested. Each record contains the identity of the unit/subunit, type of action (removal for diagnostic or transient), and a counter indicating the number of times the action has been taken. When an entry is made in the table, a search is made of the other table entries looking for a match of unit/subunit identity and action type. If a match is made, the counter associated with the existing entry is incremented. If the counter exceeds a fixed threshold, the entry is made inactive and the calling program unit is informed that the threshold has been exceeded. If no match is found, all entries are pushed down and the new entry is placed at the top of the table. Thus, the oldest entry at the bottom of the table is eliminated. In this way, only the four most recent actively occurring unit/subunit problems are watched at once.

> ***Note:*** The Error Analysis Program (ERAP) program provides an additional facility for manual analysis of system trouble data that may be used to aid in resolving transient/marginal faults whose frequency falls below the predefined threshold. For a description of ERAP, refer to Section 254-280-320.

E. Babbling Isolation Routine

10.44 The babbling isolation routine is entered from the Babbling Bus Fault Recovery Program at entry point PFLRBABL. This routine then removes the processor peripheral interface and IOs from the bus in the processor area and initiates a check for babbling (incoherent data on the peripheral bus). If babbling is detected on the bus with no peripheral units configured, the condition is classified as unresolvable by automatic means and manual intervention is required. Otherwise, the units are reconfigured to the bus one at a time to see which unit introduces the babbling. Thus, either the faulty unit is detected or the babbling stopped. Program control is passed to the Peripheral Unit Bus Fault Recovery Program if the entry is from the Babbling Bus Fault Recovery Program or to the client program if the entry resulted from a phase of software initialization.

F. Final Disposition Routine

10.45 ***CPR5 and CPR6 Generics:*** At entry point PFLRACPT, the final disposition routine is responsible for achieving the fault recovery

decision made by PFLR's recovery routines, error analysis routines, or by an application fault recovery program (when PFLR passes an MCC/IO busing unit trouble report). The action disposition routine checks the trouble reported. If the central control trouble has been reported, the disposition routine transfers to the Central Control Fault Recovery Program (CCFR) which is responsible for all remaining fault recovery actions. Other troubles reported usually result in the removal of a faulty unit or subsystem. If a unit is taken out of service and this is not a repeated action, the final disposition routine requests a diagnosis to be run on a deferrable-time basis. For MCC/IO busing unit reported troubles, only a reconfiguration of hardware access is needed. After fulfilling recovery actions, the PFLR disposition routine transfers to MARP with an order about the appropriate return point to normal call processing.

10.46 ▶*CPR7* ***Generic:*** At entry point PFLRACPT, the fault is identified as F-level. The analyze and report routine (PFLRANRR) is called to analyze and then to implement the fault recovery decision. On return from the analyze and report routine, the final disposition routine transfers to MARP with an order concerning the appropriate return point to normal call processing.◀

G. Status Information and Recovery Reports

10.47 Throughout PFLR F-level fault recovery procedures, status words and state words are updated. At any specific time, this status information identifies the current state in recovery process and shows the hardware and software status of the MCC/IO and the peripheral unit bus.

10.48 Also throughout the recovery procedures, sources blamed for the F-level failures are buffered and, at the end of recovery actions, this data is transferred to the application system's report table PU1RESULT. (Application error analysis uses this data.)

10.49 Note that communication exists continuously between PFLR and IOCP about functioning IOCs so that messages may be rerouted from an out-of-service unit to a predefined backup.

H. Processor Peripheral Interface Loop-Around Test Subroutine

10.50 The processor peripheral interface loop-around test subroutine, entered at PFLRLOOP, executes test sequences from information in the client program's input. (A client program's request may involve a subset of a test sequence set.) Peripheral unit access testing may be for any combination of central control/peripheral unit bus loop-around configurations or a mapping selection for looping data back to the central controls. Note that the processor peripheral interface loop-around routine tests an isolated portion of the peripheral unit bus; a continuity test for the entire peripheral unit bus is not done by any client. A test failure return is accompanied by a failed test response that aids in determining a specific location of central control/peripheral unit bus circuitry faults.

10.51 The processor peripheral interface loop-around test can verify the functional operation of the peripheral unit busing unit receivers and drivers and the portion of the peripheral unit bus lying between the central control and the processor peripheral interfaces frame. The loop-around test cannot check for faults that are functionally associated with internal central control peripheral system circuitry such as the interleaved parity generation circuit or the answer parity check circuits. However, the capability of looping and applying desired test sequences allows PFLRLOOP to resolve troubles that are associated with a specific bit combination.

I. F-Level Fault Recovery Summary

10.52 In general, PFLRPIIR is mainly concerned with resolving hard faults caused by ASW failures or answer parity failures detected while accessing the MCC or an IOC. The search for the failure cause consists of sequences of bus reconfigurations followed by test routines.

10.53 For the MCC, following a bus reconfiguration, the failed order is retried to determine if it is possible to recover access to the control and display logic. No extensive testing to the control and display logic is made if the MCC is able to execute a failed order and pass a short test sequence.

10.54 However, attempting to retry a failed order to an IOC is generally unsatisfactory. Instead, a substitute test order or test sequence is used to obtain accurate test results. When PFLRPIIR determines an IOC is faulty, it has to localize the fault to the IOUS or the IOUC/IOMP. If the IOUC/IOMP is blamed for an ASW failure, no further testing is necessary. Status and/or error source register data

is used to aid in distinguishing IOUS errors from bus-related errors. (The IOP has a 24-bit combined status and error source register.)

10.55 However, if the failure is not an access failure, the IOUS to IOUC/IOMP interface circuitry is tested to determine if the IOUS is to be removed from the service. If the IOUS is not blamed, only the IOUC/IOMP that was addressed is removed from service.

10.56 When a unit is removed from service, PFLR usually requests that the appropriate diagnosis (IODG, I2DG, or MCDG) be scheduled by MACP. If the unit has a history of removal and all tests pass diagnostics, then the unit is not diagnosed.

PFLRBLMH DESCRIPTION

A. General

10.57 The PFLRBLMH pident contains all base level IO fault recovery routines, performed either as a routine or demand exercise. Two programs use PFLRBLMH routines to perform this noninterrupt fault recovery function as follows:

(a) The IOCP program detects failures in the IOUS or IOUC resulting in maintenance requests and invalid polling responses.

(b) The MACP program periodically runs a fault recovery exercise routine to check all maintenance circuitry of every IOC (old type IOU only).

Note: The IOP executes its own exercise routine from RAM.

10.58 Other subroutines in PFLRBLMH are used, as needed, for recovering from base level fault detection.

B. IOCP Entry

Maintenance Requests

10.59 Periodically IOCP polls the ▶IOs◀ to detect failures, one of which may be a maintenance request. Maintenance requests are generated when the following failures occur:

- Input buffer overflow
- Character parity failure
- Carrier failure alarm from a data set
- Low paper alarm from a terminal
- Character overrun
- Character time-out
- Automatic call unit error
- Protocol error
- Peripheral controller software error
- Peripheral controller invalid command.

10.60 When IOCP detects a maintenance request, IOCP enters PFLRBLMH at entry point PFLRIOMR. This routine determines the source of the maintenance request and takes appropriate action. If the maintenance request source is an input buffer overflow, a character parity failure, a protocol error, character overrun, or character time-out that exceeds a predefined error threshold (CPR5 and CPR6 generics) the faulty IOC is removed from service; for all other maintenance request sources (alarms), the IOC is taken out-of-service immediately. ▶For the CPR7 generic, the faulty IOC is marked for suggested removal in the request table (PU1RESULT); for all other maintenance request sources (alarms), the IOC is marked for suggested removal from service in the request table.◀

Note: A buffer overflow condition exists when an attempt is made to load characters in an already full buffer (in IOUC/IOMP).

10.61 ▶For the CPR5 and CPR6 generics, when the first protocol error of a 5-minute interval occurs, the error is ignored. If another protocol error occurs within 5 minutes, the error is reported and a hangup (restart) is issued to the channel to reinitialize the link. On the third protocol error within the 5-minute period, the error threshold is exceeded. The channel is then removed from service.

10.62 For the CPR7 generic, when the first protocol error of a 5-minute interval occurs, the error is ignored. If another protocol error occurs within the 5-minute interval, the error is reported and a hangup (restart) is requested (via the request table) to be sent to the channel to reinitialize the link. On the third protocol error within the 5-minute period, the

error threshold is exceeded. The channel is then marked for removal from service.

10.63 At 5-minute intervals, IOCP enters PFLR at entry point PFLRHANG to clear the error counts for parity errors and protocol errors.◀

Invalid Polling Responses

10.64 When an ▶IOU/IOP◀ is polled and the IOUS responds with an indication that it has work to do, but the IOCP reading of the IOUS's poll request register is all zeros (a false service request), IOCP enters PFLRBLMH at entry point PFLRPOLL. This routine resolves the false service request. The program action to resolve the false service request depends on the type of IOU that has failed. For the old type IOU usually, the source of an invalid service request is a clock failure, ie, an IOUS timing sequence failure. The PFLRPOLL routine determines the trouble to be in the IOUS, the bus, or the central control. However, for the new type IOU, the source of an invalid service request is usually an internally detected error in the IOP. For this condition, the PFLRPOLL routine determines the trouble to be in either the IOUS, the IOMP, or the IOUC. After determining the cause of the trouble (for either type IOU), the PFLRPOLL routine performs the appropriate reconfigurations and test routines, prints out the appropriate messages to the TTYs, takes the appropriate action to remove the affected equipment, and requests, as applicable, the IO diagnostic to be scheduled.

C. MACP Entry

10.65 Several times daily MACP enters PFLRBLMH at entry point PFLRIORX, the IO routine exerciser, to check the maintenance circuits of all IOCs (for old type IOU only). Routine PFLRIORX ensures that the IOUS/IOU circuits are capable of handling polling requests.

> ***Note:*** In addition to the above described IO routine exerciser (not a diagnostic) being executed on base level processing several times daily, MACP also controls the daily execution of the MCC/IOU diagnostics. Refer to paragraphs 10.71 through 10.78 for a discussion about the diagnostic request routine for these diagnostics (IODG, I2DG, and MCDG).

D. Other PFLRBLMH Subroutines

10.66 Besides the PFLRBLMH routines used periodically, PFLRBLMH contains other subroutines that are used on base level processing by PFLR or other programs. Examples of these routines and their functions are:

(a) PFLRUPMA AND PFLRIOMA (used on old type IOU only)—Sets the maintenance flip-flop(s) in one IOUC or all IOUCs, respectively, associated with a particular IOUS (set for diagnostic usage).

(b) PFLRFRLP—Audits the out-of-service lamps for all IOU frames.

(c) PFLRIOLP—Maintains primary and secondary trouble lamps on the MCC for the IOUSs and IOCs (indicates communication by normal bus configuration or by standby bus configuration).

E. ▶Analyze and Report Routine

10.67 The analyze and report routine (entry PFLRANRR) calls the short-term error analysis routine PFLRANYL in pident PFLRPIIR to enter the unit in the tables. If a removal from service was indicated in the request table (PU1RESULT) and the unit did not exceed the error threshold, the action is changed to a remove and diagnose. For the CPR5 and CPR6 generics, the PFLR fault recovery action is then reported to MARP and a return is made to the appropriate fault recovery routine. For the CPR7 generic, after entering the unit in the error analysis table, the PFLR requested action is reported to MARP. A call is then made to the application administration routines (via PATTANRR) to inform them of the failure on the unit. The application may now change the PFLR requested action. If a change is made, the alternative action is indicated in the request table.

10.68 For the CPR7 generic, the analyze and report routine checks the action reported. If the central control trouble has been reported, the analyze and report routine transfers to the Central Control Fault Recovery Program (CCFR). The CCFR is responsible for all remaining fault recovery actions. If the central control was not found at fault, the analyze and report routine will take the appropriate action as indicated in the request table. The routine then reports to MARP its final recovery action.◀

F. Base Level Recovery Summary

10.69 As conveyed by previous discussions, the MCC/IO fault recovery functions are shared

by base level (noninterrupt) processing and F-level (interrupt) processing. The MCC base level processing consists only of running a routine diagnostic daily. By sharing the MCC/IO fault recovery functions between interrupt and noninterrupt processing, delays caused by interrupts in normal base level timing are minimized.

10.70 Summarizing, IO base level fault recovery functions are done by PFLRIOMR and PFLRPOLL (routines of PFLRBLMH) which are run when fault flags are detected by IOCP through periodic polling action. Also, several times daily, PFLRIORX (IOC exerciser routine), which is MACP controlled, is run to check the workability of all IO maintenance circuitry on a channel basis (old type IOU only).

PFLRDGNH DESCRIPTION

A. General

10.71 Routines in the PFLRDGNH pident perform functions relating to the execution of the MCC/IO diagnostics, MCDG, IODG, and I2DG. The two main functions, involving two routines for the IODG/I2DG and two routines for MCDG, handle requests resulting from fault recovery actions. These two functions are:

(a) To initialize a diagnostic (prediagnostic routine)

(b) To be the diagnostic final handler (post-diagnostic routine).

10.72 As mentioned previously, PFLRDGNH also contains a diagnostic routine used periodically (daily) to request MACP to run specified portions of the MCC or IO diagnostics. Other diagnostic control routines in this pident perform miscellaneous services, such as setting up unit type and member numbers, etc.

B. Prediagnostic Initialization Routines

10.73 The prediagnostic initialization routine is entered from the Diagnostic Control Program (DCON) either at entry point PFLRDIMC or PFLRDIIO before executing diagnostic MCDG, IODG, or I2DG, respectively. These prediagnostic routines perform relevant hardware and software initialization, configuration, or service operations for units pending diagnosis. If units are unavailable to do the diagnostic tasks specified by the fault recovery actions, the requested diagnostic is canceled.

10.74 The MCC units pending diagnosis may involve the entire processor peripheral interface/MCC logic, or only the processor peripheral interface bus circuitry. If the processor peripheral interface/MCC bus diagnostic task was requested, the MCC bus must be capable of communicating over the remaining in-service bus. For any MCC diagnostic task, the MCC is removed from service. This routine also specifies the address of the post-diagnostic routine (explained below).

10.75 The IO prediagnostic routine sets up an address for the post-diagnostic routine, checks the availability of units specified for diagnosis, checks the status of backup channels (in service, or unequipped), and removes units of IO from service, as required. The IOs prepared for diagnosis may involve the entire selector (IOUs) or a single controller (IOUC) or an entire microprocessor (IOMP) or IOUS bus interface.

C. Post-Diagnostic Final Handling Routines

10.76 The post-diagnostic final handling routine is entered from DCON at entry point PFLRDFMC or PFLRDFIO at the completion of diagnostic MCDG, IODG, or I2DG, respectively. (After the PFLRDGNH final handling routine completes its work, it passes control back to DCON which passes control to MACP.) The post-diagnostic routines analyze the diagnostic results, interpret this output from the type of diagnostic request, and decide whether the diagnosed unit(s) should be restored to service. Specifically, if the diagnostic outcome is ATP, and the request was not a diagnose only request, the unit is restored to service.

D. Diagnostic Routine Exerciser

10.77 The diagnostic routine exerciser located in the PFLRDGNH pident is entered periodically (daily) by MACP at entry point PFLRMREX or PFLRIREX to run MCC or IO diagnostics, respectively.

E. Diagnostic Request Routine

10.78 A subroutine with entry point PFLRDIAG is called to set up an MACP job to run a diag-

nostic on either the MCC or IO or some portion of either one. Routine PFLRDIAG determines the MCC/IO unit to be diagnosed, and requests MACP to initialize the applicable software tables.

PFLRRRCR DESCRIPTION

A. General

10.79 The PFLRRRCR pident contains a collection of subroutines used to administer the servicing needs of the MCC/IO units and related bus circuitry during PFLR fault recovery processing; these subroutines are also available for use by other maintenance programs. The subroutines, in general, can be classified in two main groups:

(a) Diagnostic routines, primarily configuration routines

(b) Service routines, primarily unit removals and restorations.

10.80 The following subroutine descriptions cover only the most important ones in PFLRRRCR. For explanations and comments of those subroutines not explained in this section, refer to the PFLRRRCR program listing (PR-5A342).

B. MCC/IO Configuration Subroutines

10.81 Three basic configuration subroutines are provided by PFLRRRCR for the purpose of establishing any desired peripheral unit configuration with the MCC or IOC. The entry points into these subroutines followed by a brief description are as follows:

(a) PFLRBCON—According to the client program's input, a specified MCC controller or a specified IOUS is configured to communicate with the central control over a specified bus configuration (client inputs controller's K-code and desired state of routing control flip-flops RO, S0, S1). See Table F.

(b) PFLRUPDT—The bus configuration from a specified MCC controller or a specified IOUS (one channel) to the central controls is updated to the standard peripheral unit bus configuration used by simplex controllers. The standard peripheral unit bus configuration is obtained from a configuration control table in PFLR. Once the desired configuration is established, the unit's software status (status word per MCC or per IOUS) is updated, if requested by the client program.

(c) PFLRIOCU—The bus configurations of all equipped IOUSs in a switching office are updated. The update places the peripheral unit bus-to-central controls in the standard configuration used by simplex controllers. A simplex controller configuration table specifies these standard settings of the routing control flip-flops (in central controls and IOUSs). See Tables E and F. Software status words (a status word per channel) are also updated.

C. Remove/Restore Subroutines

MCC Removal/Restoration

10.82 The MCC restore subroutine PFLRRSTM configures the MCC on the peripheral unit bus, puts the unit in normal mode, and requests MCCM to update the matrix.

10.83 The MCC remove subroutine places the MCC in a bypass mode, updates the MCC status word to indicate the MCC is out-of-service, sets the maintenance flip-flop, lights an out-of-service lamp on the MCC/processor peripheral interface frame.

10.84 Subroutine PFLRMRQP (in PFLRRRCR) is the request handler for all manual and TTY requests for MCC removal and unconditional restoration.

IO Unit Removal/Restoration

10.85 Subroutine PFLRRMIS, PFLRRMIP, or PFLRRMIC is used to remove from service an IOUS and all equipped IOUCs/IOMPs or an IOMP and all equipped IOUCs or an IOUC associated with an IOUS, respectively. When the unit(s) is removed from service, pertinent software status words are updated, appropriate lamps are lighted on the MCC, and IOCP is notified that the applicable channel(s) is out-of-service.

10.86 Subroutine PFLRRSTS, PFLRRSTP or PFLRRSTC is used to restore to in-service status an IOUS and all associated IOUCs/IOMPs or an IOMP and all associated IOUCs or an IOUC/IOMP, respectively. An IOUC/IOMP is restored only if its IOUS is equipped and in service. On an IOUS

restoral, an IOUC/IOMP is restored only if the IOUC/IOMP is equipped and not marked in-trouble. When the units are returned to service, pertinent software status words are updated.

10.87 Subroutine PFLRMRQI (in PFLRRRCR) is the request handler for all manual and TTY requests for IO removal and unconditional restoration.

D. ◗ Routines Provided for Application Interface

10.88 For the CPR7 generic, subroutines PFLRXID, PFLRSTAT, PFLRTRBL, and PFLRRJOB provide status information and perform functions for the application. The PFLRXID subroutine will write a given vector into the first transmit buffer of a high speed IO protocol channel TN82. The PFLRSTAT subroutine will return to the application the status of any given IO. The PFLRTRBL subroutine will set or clear trouble bits on any given IO. Finally, the PFLRRJOB subroutine will request a run job for restoral on any given IO.◖

11. 1A PROCESSOR POWER CONVERSION AND DISTRIBUTION FRAME FAULT RECOVERY PROGRAM—PDFR

INTRODUCTION

11.01 The power conversion and distribution frame (PCDF) of the 1A Processor is made up of duplicated distribution circuits and C converters. The distribution circuits supply -48 and +24 volt power to the various processor frames. Power problems, such as blown fuses, are detected by power monitors and reported by standard 1A power control switches located within each circuit. The C converters also contain power monitors for detecting such problems as out-of-tolerance voltages and currents. The monitors also take the necessary action to remove any faulty unit (converter or distribution circuit) from service and report the problem to the operating personnel. The monitors are important in protecting the processor from power problems and must also have safeguards against malfunctions. The 1A Processor Power Conversion and Distribution Frame Fault Recovery Program (PDFR) is used for this purpose. Figure 24 shows the programs that interface with PDFR.

11.02 The functions of the monitors are tested by a wired test package, FB153, connected to each monitor. The test may be initiated by either software or direct manual actions. By using this test on the distribution circuits power monitors, their ability to efficiently detect and report distribution problems can be tested. The primary function of PDFR is to test these power monitors on request. The requests come from three sources. First, the monitors are routinely exercised once per day during a non-busy hour. Second, TTY messages test specific functions or units. Finally, power control switch scan point changes are detected by the scan point monitor program MCCM. The PDFR program identifies the unit or units involved, does whatever work was requested, and reports the results to the operating personnel. If removal is necessary PDFR can only grant permission; the unit will not stop supplying power until powered off by the operating personnel.

PDFR—FUNCTIONS AND STRATEGY

11.03 The PDFR program has a variety of jobs to do from routine daily testing to granting and denying a request for out-of-service. These jobs are initiated in one of the following ways.

A. Routine Exercise

11.04 Once each day all monitors in the PCDF are sequentially tested. The Maintenance Control Program (MACP) schedules this task at a nonbusy hour and brings in PDFR. The PDFR program requests a PCDF diagnostic via the Diagnostic Control Program (DCON) and acts as initial and final handler for the diagnostic. The PDFR program selects the monitors to be tested, analyzes the diagnostic results, and outputs the data via TTY. The routine exercise is canceled if any of the converter scan points are in an abnormal state.

B. TTY Requests

11.05 The PDFR program accommodates four TTY maintenance input messages initialized by the Manual Input Request Administration Program (MIRA) as follows:

(a) Diagnose (DGN)—The TTY request for a DGN on a specific unit enters PDFR from MIRA. The PDFR program provides a diagnostic routine and acts as the initial handler routine. The PDFR program checks the converter scan points for abnormal states and, if satisfactory, sets the proper phase bits for the request. Results of the diagnos-

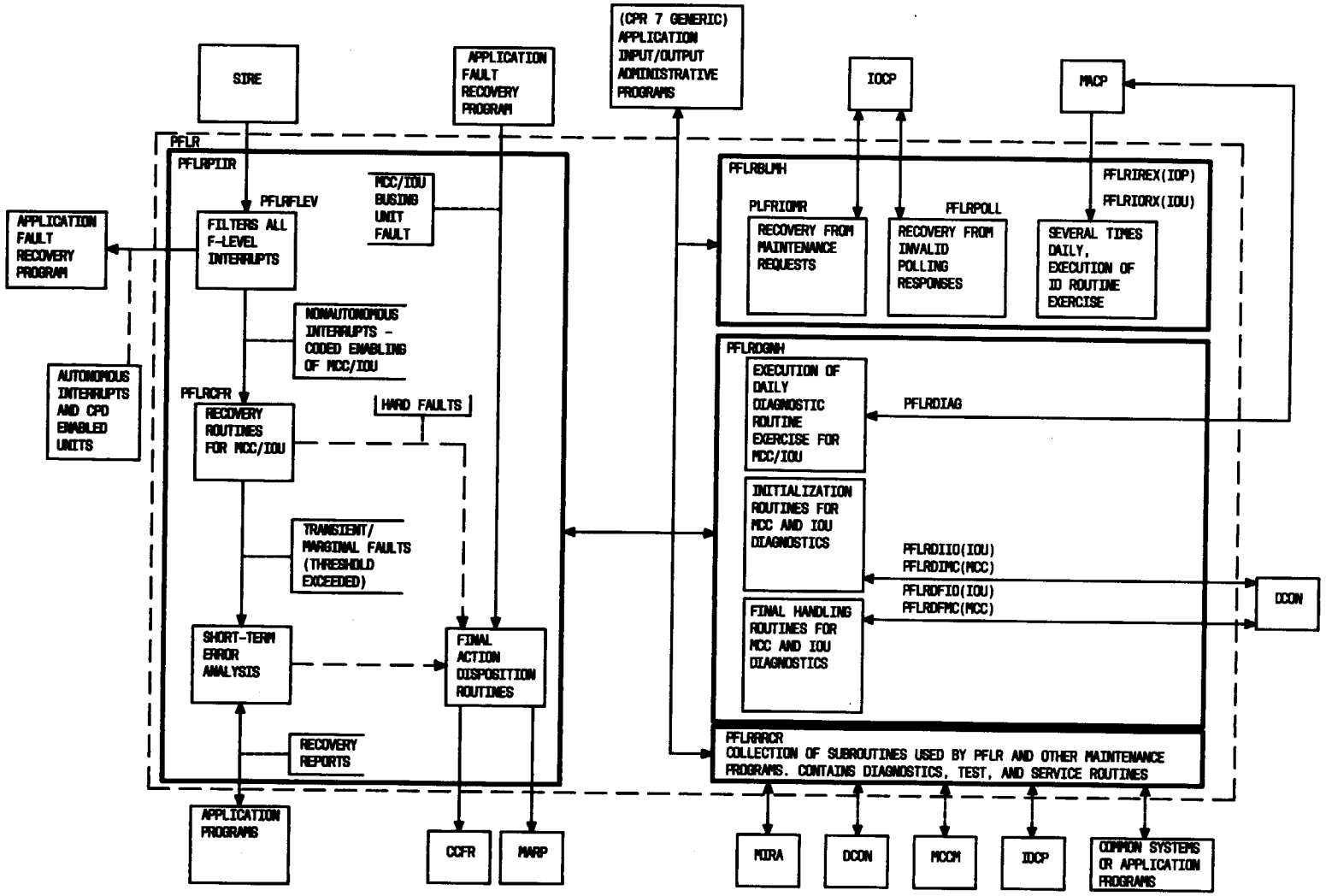


Fig. 24—1A Processor Power Conversion and Distribution Frame Fault Recovery Program (PDFR)--Program Interfaces

tic are reported via TTY from DCON. Units are not removed from service as the result of diagnostic failures since this indicates monitor troubles as opposed to convert problems; however, the failed unit will appear on the out-of-service list. The operating personnel may then make further tests and/or repairs for failures.

(b) Remove (RMV)—The RMV message is used primarily for test purposes and if actual removal is required the manual request-out-of-service on the frame itself is used since further manual action is needed to remove the frame from service. The request enters PDFR from MIRA and the unit requesting out-of-service is identified. If no unit is specified, the request is denied as invalid. If the unit is a power distribution circuit, the request is granted without further test since the power switch of the distribution circuit is not required for proper operation of the PCDF. If the request is for a converter, PDFR must verify that no other converter is out of service or is functioning improperly. The PDFR program makes two checks for this condition. First, each converter is tested for in-service indications via the MCCM scan points. If all units pass, PDFR then runs diagnostics against the monitors of the supporting converters of the PCFD member requesting out-of-service. If either check fails, the request is denied an RMV. A denied message is printed and a spurt minor alarm will sound. If the checks pass, the appropriate lamps are turned on and software tables updated. If one converter in each half of the PCDF is off-line, the primary lamp is lighted; otherwise, the secondary lamp is used.

(c) Exercise (EX)—The TTY request to exercise a particular unit enters PDFR from MIRA. The PDFR program actions then follow the same pattern used for the DGN request. The converter scan points are checked for proper states, and diagnostic routines in DCON are called that provide, in addition to diagnostic functions, provide control features such as looping within the diagnostic.

(d) Restore (RST)—The TTY request to restore enters PDFR from MIRA and must specify a particular subunit or else the request is denied as invalid. This request may be conditional or unconditional restore. If it is conditional MIRA requests that PDFR run diagnostics against the particular unit providing that all other converters are in the normal state. If the diagnostics pass, then the unit is restored and the appropriate message is sent via TTY. If the request is for unconditional restore, then MIRA comes directly to PDFR which grants the restore without any testing.

C. Scan Point Changes

11.06 Changes on the supervisory scan points of the power conversion and distribution frame (PCDF) are detected by the scan point monitor program MCCM, and PDFR is entered through MIRA. The PDFR program analyzes the transition codes and determines which of the several functions are required. the PDFR then enters one of the following:

(a) Request-Out-of-Service—When a change from normal to request-out-of-service is detected, PDFR treats it in the same way as the conditional remove request, and it is granted if companion units are functional. In addition the ACK lamp, which is turned on as an acknowledgment of the request, is extinguished.

(b) Power Off—The scan point monitor program MCCM reports power-off problems and the unit involved via TTY. The PDFR program then updates the status table and the primary and secondary trouble lamps.

(c) Power Alarm and Fuse Alarm—The scan point monitor program MCCM reports these problems via TTY and initiates a software major alarm. The PDFR program updates the status table, trouble lamps, and then grants an immediate out of service if other converters on that PCDF are in a normal state. For a fuse alarm, a hardware major alarm is also initiated from the frame power switch.

(d) Return to Normal—On detecting a change back to normal, MIRA obtains a general buffer table and requests a diagnostic. The PDFR program acts as initial and final handler, sets the phase bits, and returns to DCON. The DCON program performs the diagnostic and analyzes the results of the test. If the test passes, it will indicate ATP but the lamps will not be extinguished. An RST message at the TTY is necessary to complete the restoral and extinguish the lamps.

TROUBLE REPORTING

11.07 The PCDF frame troubles are reported by four means. The hardware generates two of these through office alarms, bells, and light emitting diodes (LEDs). The LEDs are mounted on the frames and point directly to trouble areas. Output messages

identifying problem areas via TTY come from the scan point monitor program MCCM if detected there or from PDFR if detected during a diagnostic. The last means of trouble reporting is the equipment status lamps on the MCC. The PDFR program has control of these lamps. The primary lamp shows that more than one C converter is down in the PCDF ▶or a distribution and monitor circuit is powered down while the secondary lamp shows a single C converter failure or a distribution and monitor circuit is out of service.◀

PDFR—PROGRAM STRUCTURE

A. General

11.08 The PDFR program is a single pident program containing 20 program units. The program units are routines that perform the various maintenance and reporting functions required for the PCDF. Figure 24 shows the various programs that interface with PDFR. Four major entry points are provided: three are for use by MACP and one for DCON. A discussion of these entries and their functions follow.

B. PDFR Routines

PDFRRDST Description

11.09 The PDFRRDST routine is entered from one of two sources, scheduled from MACP, or a manual entry from the MCC. The MACP program periodically reports a list of out-of-service units, and enters PDFR to obtain the status of all PCDF units for this report. The PCDF status table, PD1RST, is the source of out-of-service units. The PDFR program maintains and updates the table when scan point changes occur. On entry, PDFR checks the status of each unit, converter, and distribution circuit in the PCDF and reports this status sequentially to MACP. As each unit is checked and reported, MACP returns to PDFR and continues looping until all units are checked. Final exit from the PDFRRDST routine is to MACP indicating that all units have been checked.

PDFRRTNX Description

11.10 The monitor circuits are routinely exercised once each day to verify that they are functioning properly. The MACP program schedules this task at some nonbusy hour and enters PDFR at PDFRRTNX. Routine PDFRRTNX then obtains a general buffer table from MACP, initializes it, requests an MACP job via the job request table, and exits to DCON for routine exercise. The DCON program reenters PDFR at PDFRDFNH to act as initial and final handler for the diagnostic. The PDFR program selects the proper routine for the exercise, sets the diagnostic phase bits, and goes to the Power Distribution Diagnostic Control Program (PDDG) for the power monitor test on units specified by PDFR. The PDDG program calls set the test results of the power monitor test and return to PDFR which checks to see if all tests are complete. If they are, PDFR sets up for the next member, if required, or issues an end job request if all tests are complete.

PDFRDFNH Description

11.11 The PDFRDFNH routine is entered from DCON to act as initial handler for the PDDG diagnostics for several functions. One of these functions is the routine exercise periodically scheduled by MACP and was discussed in conjunction with the PDFRRTNX entry. In addition, this routine is also entered to process the following functions:

- Scan point changes to normal
- Diagnose TTY message
- Conditional restore message
- Exercise TTY message.

For these functions, MIRA is entered to set up the general buffer table and request diagnostics from DCON. The DCON program then enters the PDFRDFNH routine that selects an initial handler routine for the diagnostics. The PDFR program sets up the data in K register and reads scan points of the unit involved. If the unit's scan points indicate normal states, control passes to DCON to run the diagnostics. If the unit scan points were abnormal, restoral requests are denied and an endjob is issued. On completion of diagnostics, DCON returns control to PDFR to analyze the results. The PDFR program final handling routines determine if the test passed and, if not, a failure message is sent via TTY and the out-of-service lamp is left on. If the test passed, then PDFR runs the out-of-service/off routine that turns the out-of-service lamp off. In addition, the equipment status light on the MCC is turned off.

PDFRREMV Description

11.12 The PDFRREMV routine is an MACP entry to process the following requests:

- Remove TTY message
- Normal to request-out-of-service scan point change
- Normal to power alarm scan point change
- Normal to power-off scan point change
- Unconditional restore TTY message.

On receipt of any one of these requirements, MIRA requests that MACP schedule the PDFR program and then enters the PDFRREMV routine. The PDFRREMV routine sets up and reformats the input data for later use, identifies the job requested, and transfers to that routine. For an unconditional restore request, control passes to the restore routine that does the restore with no testing and extinguishes the out-of-service lamp.

11.13 If the request is for a remove or request-out-of-service, PDFR uses the PDREMV removal routine. This routine first identifies the requesting unit and, if it is a distribution circuit, the out of service is granted automatically. If the request is for a C converter, the power distribution frame status table is checked for any complementary converters out of service. If any complementary converters are out of service, the request is denied and a denial message is sent via TTY. If no converters are found out of service, PDFR obtains and loads a general buffer table to run diagnostics on each complementary C converter. The PDFR program transfers to PDDG which runs the diagnostic and, if any failures are found, the request is denied and the operating personnel notified via TTY. If no failures are found, the out of service is granted and the proper lamps are turned on. The PDFR program issues an endjob on completion, and control returns to MACP.

PDFR INTERFACES

11.14 MACP—The MACP program enters PDFR at PDFRRTNX once each day to run the routine exercise on the PCDF. On completion, the exercise issues an ENDJOB macro to return to MACP.

11.15 DCON—All entries from DCON for the PCDF initial handler routine come to PDFRDFNH. This entry is used for diagnosing TTY messages, conditional restore messages, exercise messages, routine exercises, and for all scan point changes to the normal state. The PDFR program defines the task and branches to the proper routine. An ENDJOB macro is issued at the end of processing for any of these routines.

11.16 MIRA—All entries from MIRA come to PDFRREMV. This includes the remove TTY message, unconditional restore message, and all scan point changes from the normal state. The PDFR program defines the task, selects the proper routine for execution and, on completion, issues the ENDJOB macro.

11.17 MCCM—The PDFR program interfaces with MCCM for all requests that involve lamp status. The final disposition of the OS and ACK lamps is sent to MCCM via the OSLAMP-ON and OSLAMP-OFF subroutines. The unit's identity is sent as requested in the K register. After any status change, the equipment status lamps on the MCC are updated via the MCCON and MCCOFF macros. On a periodic basis, the OS lamps status is audited on an entry from MCCM at PDFAUD.

11.18 PDDG—The PDFR program calls the PCDF program's diagnostic (PDDG) through DCON. The DCON diagnostic buffer table is filled as required for each case. The PDDG program then tests either one full power conversion and distribution circuit or some subset consisting of one or more of the subunits. This data is passed to PDDG in the phase request bits. The PDDG program runs the power monitor test against the specified units. The results are entered in the diagnostic buffer table and printed by DCON before passing control back to PDFR.

12. ABBREVIATIONS AND ACRONYMS

12.01 The following is a list of the commonly used abbreviations and acronyms used within this section.

ADDG	Auxiliary Data System Diagnostic Control Program
APCL	Attached Processor Communication Link

APFR	Attached Processor Fault Recovery
API	Attached Processor Interface
APS	Attached Processor System
ASW	All-Seems-Well
ASWF	All-Seems-Well Failure
ATP	All Tests Pass
AU	Auxiliary Unit
AUB	Auxiliary Unit Bus
AUBSQ	Auxiliary Unit Bus Sequencer
AUFR	Auxiliary Unit Fault Recovery Program
BUFOFL	Buffer Overflow
CCDG	Central Control Diagnostic Program
CCFR	Central Control Fault Recovery Program
CPD	Central Pulse Distributor
CPSR	Control Pulse Source Response
CSFR	Call Store Fault Recovery Program
DCON	Diagnostic Control Program
DFOFL	Disk File Overflow Source
DKAD	File Store Administration Program
DMA	Direct Memory Access
DMERT	Duplex Multienvironment Real Time Operating System
DPWEF	Data Parity or Write Enable Failure
DUAD	Data Unit Administration
DUFR	Data Unit Fault Recovery Program
DUS	Data Unit Selector
ERAP	Error Analysis Program
FSFR	File Store Fault Recovery Program
GCP	Generate Control Pulse
IDMM	Identification Tag Mismatch
IO	Input-Output
IOC	Input/Output Channel
IOCP	Input/Output Control Program
IODG	Input/Output Unit Diagnostic Program
IOMP	Input/Output Microprocessor
IOP	Input/Output Processor
IOU	Input/Output Unit
IOUC	Input/Output Unit Controller
IOUS	Input/Output Unit Selector
I2DG	Input/Output Processor Diagnostic Program
LED	Light Emitting Diode
MACP	Maintenance Control Program
MARP	Maintenance Restart Program
MCC	Master Control Console
MCCM	Master Control Console Common Control and Monitor Program
MCDG	Master Control Console Diagnostic Program
MIRA	Manual Input Request Administration Program
PCDF	Power Conversion and Distribution Frame
PCRV	Processor Configuration Recovery Program

PDDG	Power Distribution Diagnostic Control Program
PDFR	1A Processor Power Conversion and Distribution Frame Fault Recovery Program
PEST	Prevent Error Source Transmission
PFLR	1A Processor F-Level Fault Recovery Program
PIC	Peripheral Interface Controller
pident	Program Identification
PR	Program Listing
PSFR	Program Store Fault Recovery Program
RAM	Random Access Memory
SADK	System Audit for File Store Administration Program
SAWS	Writable Store Audit Program
SIRE	System Interrupt Recovery Program
SSFR	Single Strategy Fault Recovery for Attached Processor System
SYUP	Systems Update Program
TRAMM	Translated Address Mismatch Source
TUC	Tape Unit Controller
UTRAMM	Untranslated Record Address Mismatch