FPS-164 IBM/CMS FRONT-END COMPUTER MANUAL

(RELEASE F)

860-7494-004A

by FPS Technical Publications Staff

FPS-164 IBM/CMS FRONT-END COMPUTER MANUAL

(RELEASE F)

860-7494-004A

Publication No. 860-7494-004A
February, 1985

NOTICE

The information in this publication is subject to change without notice.

Floating Point Systems, Inc. accepts no liability for any loss, expense, or damage resulting from the use of any information appearing in this publication.

Printed in USA

REVISION HISTORY

This manual is the FPS-164 IBM/CMS Front-end Computer Manual, 860-7494-004. The last four characters shown under the revision number column indicate the part number change for each revision. The last entry is the latest revision to this manual.

REV. NO.	DESCRIPTION	DATE
-004A	Original issue.	2/85

NOTE: For revised manuals, a vertical line "|" outside the left margin of the text signifies where changes have been made.

CONTENTS

CHAPTER 1	INTRODUCTION	Page
1.1	PURPOSE	1-1
1.2	SCOPE	1-1
1.3	CONVENTIONS	1-2
1.4	RELATED PUBLICATIONS	1-3
CHAPTER 2	PROGRAM DEVELOPMENT	
2.1	INTRODUCTION	2-1
2.2	TOOLS NECESSARY FOR PROGRAM DEVELOPMENT	2-1
2.2.1	PDS Command Line Syntax	2-1
2.2.2	Default CMS Naming Conventions	2-3
2.2.3	Interpretation of Error Messages	2-5
2.2.4	Accessing SC Software	2-6
2.2.5	Program Development Software Programs	2-7
2.3	SJE PROGRAM DEVELOPMENT AND EXECUTION	2-16
2.3.1	SJE Program Development	2-17
2.3.2	Execution Using SJE	2-18
2.3.3	Executing an SJE Job in CMS Batch Mode	2-26
2.3.4	SJE User Attention Command	2-27
2.4	APEX64 PROGRAMMING	2-28
2.4.1	UDC/ADC FEC Programs	2-28
2.4.2	Program Development for APEX64	2-36
CHAPTER 3	IBM/CMS SPECIFIC SOFTWARE OF THE FEC/SC SYSTEM	
3.1	INTRODUCTION	3-1
3.2	ALLOCATION OF AN SC	3-1
3.2.1	The Role of the APMGR	3-1
3.2.2	Number of Users	3-2
3.2.3	Selecting an SC	3-2
3.2.4	Releasing and Detaching an SC	3-3
3.2.5	SC Device Address 0	3-3
3.2.6	Force-Release Processing	3-3
3.3	ROLL-IN/ROLL-OUT (RIRO)	3-4
3.4	I/O TO FEC FILES	3-5
3.5	USING PRESERVE AND RESTORE	3-6
3.6	SJE DATA CONVERSION UTILITIES	3-8
3.6.1	FEC Data File to SC Data File Conversion Procedure	3-11
3.6.2	APFTN64 File to FEC File Conversion Procedure	3-15
3.7	LINKING TO FEC-SPECIFIC LIBRARY ROUTINES	3-19

CONTENTS

CHAPTER 4	IBM/CMS FEC-SPECIFIC HARDWARE OF THE FEC/SC SYSTEM	Page
4.1	INTRODUCTION	4-1
4.2	HARDWARE FOR FEC/SC COMMUNICATION	4-1
4.2.1	The Role of the IBM CPU	4-2
4.2.2	The Role of the Channel	4-2
4.2.3	The Role of the HISP	4-3
4.2.4	The Role of the SC CPU	4-3
4.2.5	The Role of the Formatter	4-4
4.3	THE SC'S FEC INTERFACE INTERNAL STRUCTURE	4-4
4.3.1	Host Adapter	4-4
4.3.2	FEC Interface Support Processor	4-4
4.3.3	Formatter	4-4
4.4	IBM/CMS FEC-SPECIFIC HARDWARE OF THE FEC/SC SYSTEM	4-6
4.4.1	Address Recognition	4-6
4.4.2	Channel Interface Protocol	4-6
4.4.3	Channel Commands	4-7
4.4.4	Sense Bytes	4-8
4.4.5	Sense I/O Type Command	4-12
4.4.6	Unit Check Mask	4-12
4.4.7	Channel Interrupts	4-13
4.4.8	Interface Busy Conditions	4-15
4.4.9	Format Conversion	4-15
4.4.10	FEC/SC Interconnect Hardware	4-20
4.4.11	Configuring The Host Adapter Board	4-20
APPENDIX A	FEC COMMAND AND STATUS REGISTER	
A.1	FEC COMMAND AND STATUS REGISTER (HCSR)	A-1
A.1.1	HCSR(u): Interrupt Control and HISP Commands	A-1
A.1.2	HCSR(l): Error Flags	A-3
APPENDIX B	FEC-SPECIFIC ERROR MESSAGES	
APPENDIX C	RESERVED ROUTINE NAMES	
INDEX		

ILLUSTRATIONS

Figure No.	Title	Page
1-1	FPS-164 Manuals Group	1-5
3-1	FEC File to SC File Conversion Steps	3-10
3-2	FEC Data File to SC Data File Conversion Logic	3-13
3-3	APFTN64 File to FEC File Conversion Logic	3-17
4-1	Interface Hardware Block Diagram	4-5
4-2	Interface Sense Byte Format	4-9
4-3	Bytes Returned by Sense I/O Type Command	4-12
4-4	Unit Check Mask Bit Fields	4-12
4-5	IBM Integer*4 to and From SC Integer	4-16
4-6	IBM Integer*4 to and From SC Long Integer	4-16
4-7	IBM Integer*8 to and From SC Long Integer	4-16
4-8	IBM Integer*4 to and From SC Integer Halfword Packed Integer	4-17
4-9	IBM Unformatted*8 to and From SC Word Type	4-17
4-10	IBM Real*4 to and From SC Floating-point Number	4-18
4-11	IBM Real*8 to and From SC Floating-point Number	4-18
4-12	Dual IBM Real*4 to and From SC Halfword Floating-packed Number	4-19
4-13	IBM Logical*4 to and From SC Logical	4-19
4-14	SC I/O Panel	4-20
A-1	HCSR(u): Interrupt Control and HISP Commands Format	A-1
A-2	HCSR(1): Error Flags Format	A-3

TABLES

Table No.	Title	Page
1-1	Related FPS Publications	1-3
1-2	Related IBM Publications	1-4
1-3	Related ANSI Publications	1-4
2-1	Default CMS Filetypes, Record Formats, and Block	2-4
2-2	APAL64 Options	2-8
2-3	APFTN64 Options	2-9
2-4	APLIBR64 Options	2-10
2-5	APLINK64 Options	2-11
2-6	APDBUG64 Options	2-13
2-7	SJE Options	2-14
2-8	JDL Control Statements	2-20
2-9	JDL Service Request Statements	2-21
2-9	JDL Service Request Statements	2-22

TABLES (cont.)

Table No.	Title	Page
3-1	FEC File Prefixes	3-5
4-1	Channel Commands	4-7
4-2	Interface Sense Byte 0 Fields	4-9
4-3	Interface Sense Byte 1 Fields	4-10
4-4	Interface Sense Byte 2 Fields	4-11
4-5	Interface Sense Byte 3 Fields	4-11
4-6	Unit Status Field Bits	4-13
4-7	Configuration Mode Switch States (Switches 1 and	
4-8	Configuration Mode Switch States (Switches 3, 4, and 5)	4-22
4-9	Configuration Mode Switch States (Switches 6, 7, and 8)	4-23
A-1	HCSR(u): Interrupt Control and HISP Commands Fields	A-2
A-2	HCSR(l): Error Flags Fields	A-3

CHAPTER 1

INTRODUCTION

1.1 PURPOSE

This manual explains how to use an FPS-164 Scientific Computer (SC) that is connected to an IBM Front-end (FEC) computer. The FEC can be an IBM 370, 33XX, 43XX, or 308X mainframe running the VM/SP-CMS operating system (referred to as "CMS" in this manual). The manual provides program development and execution examples that illustrate use of the System Job Executive (SJE) and the SC Executive (APEX64). The SJE and APEX64 software allow users to run programs on the FPS-164. This manual also provides FEC-specific software, FEC-specific hardware, and FEC-specific error information.

1.2 SCOPE

The following list shows the organization of this manual.

Chapter 1	This chapter explains the purpose, scope, conventions, and related publications for this manual.
Chapter 2	This chapter describes the necessary tools for SJE and APEX64 program development. Chapter 2 also provides examples of SJE and APEX64 program development and execution.
Chapter 3	This chapter describes IBM/CMS-specific software features.
Chapter 4	This chapter describes IBM/CMS-specific hardware features. Chapter 4 also provides the FEC/SC conversion formats supported.
Appendix A	This appendix describes the FEC command and status register (HCSR).
Appendix B	This appendix lists the AP Manager (APMGR) trace messages, their causes, and suggested corrective action.
Appendix C	This appendix lists DAPEX (Dependent Advances Processes Executive) error messages, causes, and the corrective action the user is advised to take.
Appendix D	This appendix discusses the convention for naming subroutines and common blocks.

Where appropriate, this manual refers the reader to other publications for more detailed information on how to operate individual components of the FPS-164 hardware and software.

1.3 CONVENTIONS

The following list presents conventions used in this manual.

- The term SC means the FPS-164 Scientific Computer.
- The term "FEC" (front-end computer) or "host" means the IBM computer to which the SC is attached. Throughout this manual, the terms "FEC" and "host" are used interchangeably.
- Uppercase parts of keywords (in syntax examples) are the shortest abbreviation of a command allowed.
- The notation "|" used in syntax examples means "or".
- Brackets [] used in a syntax example enclose optional items of the command.
- Ellipses (...) used in a syntax example indicate repetitions in a command.
- User-supplied parameters for commands are underlined when shown in a syntax example.
- IBM terminology such as filename, filetype, filespec, filemode, and fileid conforms with its use in the IBM manual set.
- Asterisks (*) denote default options and modifiers in the Program Development Software (PDS) command tables listed in Chapter 2.
- Parentheses used in PDS commands are required in sets around each user-supplied modifier.
- The optional closing parenthesis is omitted in PDS command syntax examples and PDS commands.

1.4 RELATED PUBLICATIONS

Table 1-1 lists FPS publications, Table 1-2 lists IBM publications and Table 1-3 lists ANSI publications that the reader can consult for details of individual components of the FEC/SC system.

Table 1-1 Related FPS Publications

PUBLICATION	PUBLICATION NO.
FPS-164 & FPS-164/MAX User's Handbook and Master Index (Release F)	860-7481-003
APMATH64 Manual	860-7482-000
APFTN64 User's Guide (Release F)	860-7479-002
APAL64 Programmer's Guide (F Release)	860-7506-008
APAL64 Programmer's Reference Manual (F Release)	860-7506-009
APLINK64 Manual	860-7486-000
APDEBUG64 Manual (Release F)	860-7489-002
APLIBR64 Manual	860-7488-001
FPS-164 Operating System Manual Set Volumes 1, 2, and 3	861-7491-002
Volume 1 SJE (Release F)	860-7491-007
Volume 2 APEX64 (Release F)	860-7491-008
Volume 3 File and Memory Management (Release F)	860-7491-009
FPS-164 System Manager's/Operator's Manual (Release F)	860-7478-002

Table 1-2 Related IBM Publications

PUBLICATION	PUBLICATION NO.
VM/SP CMS Command and Macro Reference	SC19-6209
VM/SP CMS User's Guide	SC19-6210
VM/SP CP Command Reference for General Users	SC19-6211
VM/SP System Programmer's Guide	SC19-6203
IBM System 370 Principles of Operation	GA22-7000

Table 1-3 Related ANSI Publications

PUBLICATION	PUBLICATION NO.
American National Standard Programming Language FORTRAN	ANSI X3.9-1978

Figure 1-1 depicts the manual relationships in the FPS-164 manuals set as it applies to the IBM/CMS user.

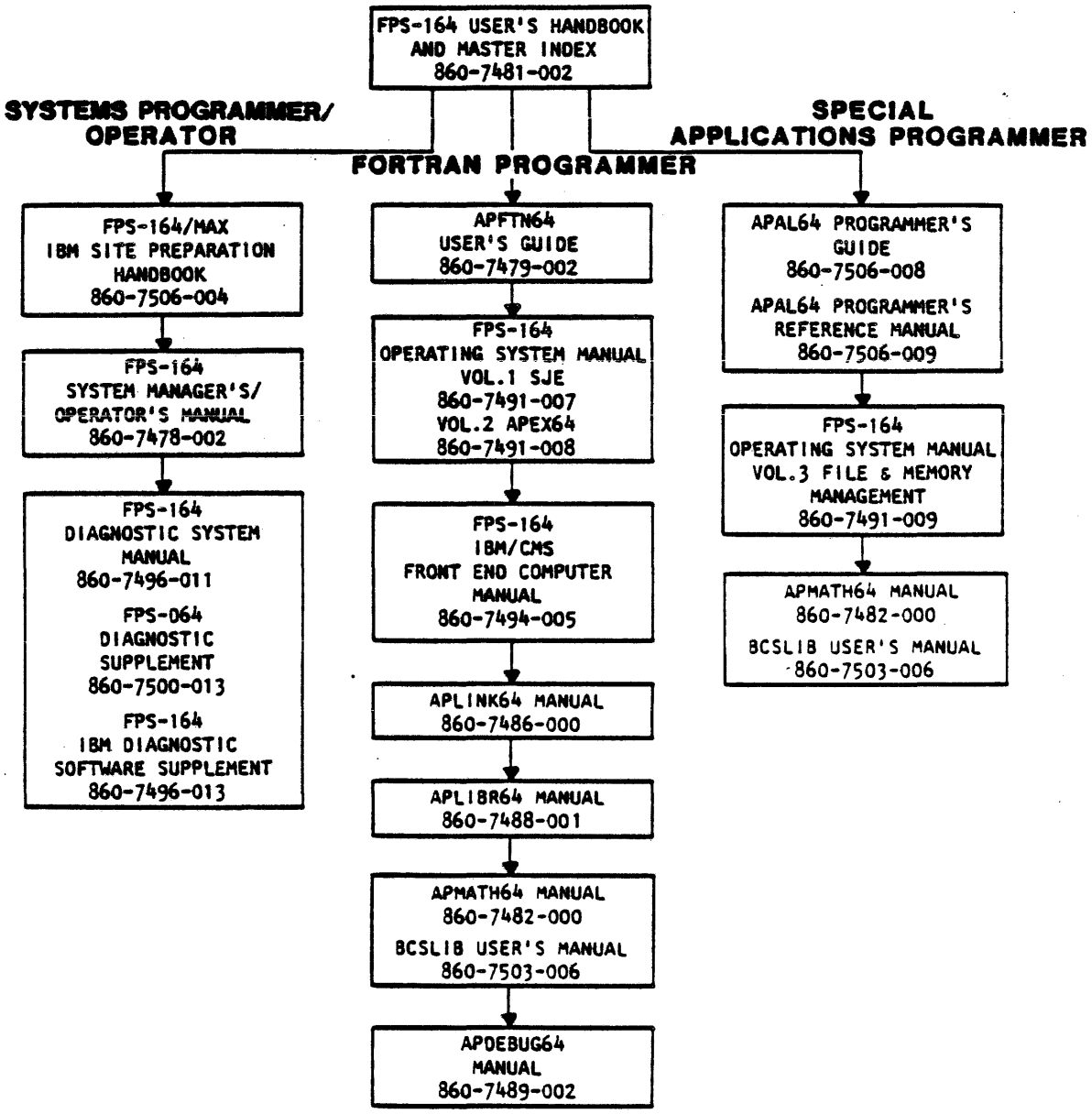


Figure 1-1 FPS-164 Manuals Group

CHAPTER 2

PROGRAM DEVELOPMENT

2.1 INTRODUCTION

Chapter 2 provides the user with the information needed to develop and execute jobs that run on the SC. This chapter contains three main sections. Section 2.2 lists the tools needed for program development and explains how to use them. Section 2.3 discusses program development and job execution for the System Job Executive (SJE) programmer. Section 2.4 discusses program development and job execution for the SC Executive (APEX64) programmer.

2.2 TOOLS NECESSARY FOR PROGRAM DEVELOPMENT

For successful program development, the user must know the following:

- how to gain access to PDS software
- how to run program development software (PDS) programs
- how to interpret PDS command error messages
- the default CMS filetype naming conventions
- the PDS commands and options

2.2.1 PDS Command Line Syntax

The command line syntax used with the SC program development software closely resembles the syntax used with CMS commands and compilers. Enhancements added to the SC program development software result in minor differences between the two. This section documents those differences, including most of the differences between standard CMS and FPS command line syntax. Sample uses of the SC program development software can be found in Section 2.2.5.2.

2.2.1.1 General Form of Commands

This manual uses a standard notation to describe the command syntax used to operate FPS system software. Use the following rules to construct commands:

1. Use delimiters, uppercase letters, and uppercase words exactly as shown in the command description. The command syntax uses spaces, commas, and parentheses as delimiters.

2. Replace lowercase letters and words by following the rules given in the descriptions.

3. Brackets [] surround optional parts of commands.

4. Use options followed by ellipses (...) more than once if necessary.

5. End each command entered on a terminal with a carriage return.

The following is an example of a command description:

LOGON user

This command identifies a virtual machine user to the operating system. An appropriate value can replace the underlined word in this command format:

user = the virtual machine user identification

The following example shows the general form of a command:

APLINK64 filespec[,filespec...][(options[])]

The forms of a filespec and option, respectively, are as follows:

filespec = filename [filetype [filemode]]

option = option [(value[,value]...)]

Option values can also be filespecs.

2.2.1.2 Command Line Options

The user has several powerful options available on the command line. These options are the $UPCASE, $FM, $BS, and CO options. The $UPCASE option allows the user to force lowercase output into uppercase.

The $FM option allows the user to set the default file mode. The $BS option allows the user to set the default binary file blocksize. Section 2.2.5.2 contains examples of these options.

The CO statement allows the user to continue the command line. CO causes the parser to prompt the user with ENTER ADDITIONAL PARMS:. Any number of continuation lines are allowed as long as the line ends with CO. Once the user enters a line of parameters without terminating it with CO, the line is considered ended and control returns to the FEC.

Within a CMS EXEC the user can continue command lines using CO and the standard EXEC &BEGSTACK or &STACK facility. Command input is redirected to a file by ending the command line with

CO (filename [filetype [filemode]]))

Filetype defaults to TXT and filemode defaults to * if not specified. A * for filemode causes all accessed CMS disks to be searched for the first occurrence of the designated file. The parser treats additional lines as though they were terminal input. Continued lines within such indirect files must still end with a CO.

Users running a program from an EXEC should note that line continuations are read off the stack within the EXEC if no filename is provided after CO. Thus, a typical EXEC file might be set up as follows:

```
&BEGSTACK
PROG4, PROG5, (IM(MYPROG), CO
UDC (MYHASI)
&END
APLINK64 PROG1, PROG2, PROG3, CO
```

In the above EXEC file, the stack provides the additional parameters which are PROG4, PROG5, an IMAGE file option, and a UDC option. Thus, when the APLINK64 statement is executed, the parameters provided in the stack are used when the parser reaches the symbol CO.

2.2.2 Default CMS Naming Conventions

Every CMS file must have a filename, filetype, and filemode. The user must always explicitly specify the filename. When the filemode is not specified on input files, all accessed CMS disks are searched in standard CMS search order until the file is found. When the filemode is not specified for output files, an attempt is made to use the same filemode as the major input file of the command. For example, the object file output from a compiler is placed on the same disk as the source input file if the disk can be written on. Otherwise, the filemode defaults to A1 or the first read/write disk found. When the filetype is not specified, it defaults to those listed in Table 2-1 below.

Table 2-1 Default CMS Filetypes, Record Formats, and Block Sizes

CMS FILETYPE	RECORD FORMAT	BLOCK SIZE	USE
(terminal)	V	130	console I/O
TEMPTXT	V	121	scratch file text
TEMPBIN	F		scratch binary file
TXT	V	121	miscellaneous text
BIN	F		miscellaneous binary
APAL64	V	121	APAL source input
APFTN64	F	80	APFTN source input
APTXT64	F		SC object module
APLIBR64	F		SC object library
APIMG64	F		APLINK output load module image
APAL64	V	121	APAL source output by APFTN
LISTING	V	121	general listing with carriage control
APERR64	V	121	error report
APLOG64	V	121	APDBUG log file
FORTRAN	F	80	APLINK output interface between FEC & SC (HASI) (input to IBM compiler)
APSMB64	F		APLINK output symbols file
APMAP64	V	121	APLINK output load map
APIMG64	F		APDBUG image save file
APDAT64	F		APDBUG APEX dat save file
APOVL64	V	121	APLINK input overlay descriptor file

NOTE

APAL64 assembler input, APFTN64 compiler input, and APOVL64 overlay descriptor linker information input can have either a variable or fixed record format and can have any logical record length. The values listed in the table above are default values, not requirements.

The default blocksize for all FPS binary files depends on the blocksize of the CMS minidisk where the file is written, ie. The FPS binary file blocksize will be the same as the CMS minidisk blocksize.

NOTE

To achieve maximum data transfer rates on SJE COPYIN/binary and COPYOUT/binary, the user should format their CMS minidisks with a blocksize of 4096 bytes (4K).

2.2.3 Interpretation of Error Messages

When the FPS command line interpreter detects a syntax error, it echoes at the terminal the line containing the error. A dollar sign appears under the first erroneous character, and the interpreter returns an explanation to the user. Common utility failures result in simple informative messages, such as OUT OF SPACE ON DISK or ATTEMPT TO WRITE ON READ-ONLY DISK, detailing the cause of the error. Subtle or uncommon errors result in a message that contains a fileid and file system macro. This fileid and return code is documented in the VM/SP CMS Command and Macro Reference publication (refer to Table 1-2).

The SC program development software sets the condition return code according to the highest severity error encountered during program execution. The interpretation of this code is as follows:

0	normal
4	warning
8	error
12	severe error
16	terminal error

2.2.4 Accessing SC Software

Typically, all FPS software resides on one system minidisk, though multiple minidisk configurations are possible as well. Before developing and running FPS software, the user must CP LINK to the minidisk containing the FPS software and access the minidisk through the ACCESS command. The data center manager should be able to give the user the name(s) of the minidisk(s) containing the FPS software.

Before running some FPS software, the user may need to redefine the amount of virtual storage dedicated to the user's virtual machine. To do this, issue the following command:

```
CP DEFINE STORAGE 2M
```

In some cases, the user may receive the following message:

```
MAXIMUM STORAGE EXCEEDED
```

If this happens, the user must have the data center manager reconfigure the user's virtual machine to allow it to define two megabytes of virtual storage (occasionally, three megabytes is needed). Once storage is redefined to two megabytes, the user must enter the following initial program load (IPL) code:

```
IPL CMS
```

To use the FPS TXTLIB'S (namely APEX64 TXTLIB, UTIL64 TXTLIB, and APDBUG64 TXTLIB) the user must include them in the GLOBAL TXTLIB statement before loading the program. For example:

```
GLOBAL TXTLIB APEX64 UTIL64 CMSLIB VFORTLIB...
```

If the program halts and the programmer receives the message

```
CP ENTERED;  CMSZER - SHARED PAGE xxxxxx ALTERED
```

it sometimes means that the virtual machine has too little memory configured into it. One approach to handling this problem is to reconfigure the virtual machine with one more megabyte of memory (refer to the DEFINE STORAGE command in Section 2.2.4) and run an IPL CMS (the CMS initial program load). The user can then attempt to run the program again.

CAUTION

> The user must not call any of the APIMG64 files by the name SUM APIMG64, since this name indicates the SC's Single User Monitor as a whole. During SC initialization, APEX64 sometimes needs to reload the SUM APIMG64 into the SC. During this reloading, APEX64 searches all accessed disks in the standard CMS search order, using the first SUM APIMG64 file it finds.

2.2.5 Program Development Software Programs

The SC program development software provides tools for writing, debugging, and maintaining SC programs. With these tools the user can convert source modules into object modules, store the object modules in libraries, selectively link the modules together, and debug the modules. The following text describes the program development software tools. APFTN64 is an FEC-resident, FORTRAN-77-optimizing cross-compiler. It accepts a superset of ANSI FORTRAN-77 statements and produces an SC object module. APFTN64 provides a convenient means for converting FEC FORTRAN programs into microcode that can run on the SC. For more information on APFTN64, refer to the APFTN64 User's Guide listed in Table 1-1.

APAL64 is an FEC-resident cross-assembler. It accepts programs written in SC Assembly Language (APAL64) and produces an SC object module. Programs written in APAL64 can take full advantage of the hardware features of the SC. For more information on APAL64, refer to the APAL64 Programmer's Guide or the APAL64 Programmer's Reference Manual, listed in Table 1-1.

APLINK64 is the relocating linker. It links together specified object modules and referenced library members to form a load module, build the interface between the FEC and the SC (HASI) when running APEX64 applications, and build symbol files used in debugging. For more information on APLINK64, refer to the APLINK64 Manual, listed in Table 1-1.

APLIBR64 creates, modifies, lists, and maintains libraries of SC object modules. APLIBR64 maintains a directory of object module information to enable APLINK64 to operate efficiently. For more information on APLIBR64, refer to the APLIBR64 Manual, listed in Table 1-1.

APDBUG64 helps a programmer debug SC programs. APDBUG64 allows a programmer to symbolically debug programs on the SC (that is, set breakpoints, examine variables, etc.). APDBUG64 is available as a stand-alone program or as an FEC subroutine library or while running under SJE. This flexibility allows the user to debug in a stand-alone mode or within an SC program environment. For more information on APDBUG64, refer to the APDEBUG64 Manual, listed in Table 1-1.

Program development procedures demonstrating all of the SC program development software are shown in Section 2.3 and Section 2.4.2.

2.2.5.1 SC Software Options

This section provides a quick reference to the SC Program Development Software (PDS) options. Tables 2-2 through 2-6 list all of the options available with each PDS program. Asterisks indicate default options or modifiers. Underlining indicates user-supplied values. Uppercase letters indicate the shortest allowed abbreviations. Note the following differences from the usual IBM/CMS command line option conventions:

- The LIST/NOLIST option of APFTN64 controls listing of the source code. Listing of generated object code is accomplished by the CODE option.

- The OBJECT/NOOBJECT option controls production of an object (binary) file. There are no modifers for the options LOAD/NOLOAD and DECK/NODECK.

The APAL64 command is entered using the following form:

APAL64 srcfile (option (modifier) option (modifier1 , modifier2)

Table 2-2 lists the options available with APAL64.

Table 2-2 APAL64 Options

APAL OPTION	MODIFIER
(srcfile	
List(lstfile)	
* NOList	
* Object(objfile)	
NOObject	
* ERRors(errfile)	
NOERRors	
* HEXadecimal	
OCTAL	
NAMes(modifier)	
	* GLOBAL
	LOCAL
	UNREF
SHow(modifier[,modifier]...)	
	SUBR
	COND
	UNREF
	ALL
* NOSHow(modifier[,modifier]...)	
	* SUBR
	* COND
	* UNREF
	* ALL
DIAGnostics(level)	
	* WARNing
	ERRor
	SEVERE
	TERMinal
FAILure(level)	
	WARNing
	* ERRor
	SEVERE
	TERMinal

The APFTN64 command is entered using the following form:

APFTN64 srcfile (option (modifier) option (modifier1 , modifier2)

Table 2-3 lists the options available with APFTN64.

Table 2-3 APFTN64 Options

APFTN64 OPTION	MODIFIER
(srcfile	
List(lstfile)	
* NOList	
* Object(objfile)	
NOObject	
APAL(apalfile)	
* NOAPAL	
* ERRors(errfile)	
NOERRors	
OPTimize(level)	0
	* 1
	2
	3
	4
LINenum	
* NOLINenum	
H4	
* H8	
DEBUG	
SUBchk	
* NOSUBchk	
CODE	
* NOCODE	
DCLchk	
* NODCLchk	
* HEXadecimal	
OCTal	
NAMes(level)	
	REF
	ALL
* NONAMes	
MAP(level)	
	REF
	ALL

Table 2-3 APFTN64 Options (cont.)

APFTN64 OPTION	MODIFIER
* NOMAP	
DIAGnostics(level)	
	* WARNing
	ERRor
	SEVERE
	TERMinal
FAILure(level)	
	WARNing
	* ERRor
	SEVERE
	TERMinal

The APLIBR64 command is entered using the following form:

APLIBR64 libfile (option (modifier) option (modifier1, modifier2)

Table 2-4 lists the options available with APLIBR64.

Table 2-4 APLIBR64 Options

APLIBR64 OPTION	MODIFIER
(filename	
REplace(objfile[,objfile]...)	
INsert(objfile[,objfile]...)	
DELete(objfile[,objfile...)	
OUTput(libfile)	
List(lstfile)	
* NOList	
SHow=[(]modifier1[,...modifiern)]	
	PSECT
	ENT
	EXT
* NOSHow[(]modifier1[,...modifiern)]	
	* PSECT
	* ENT
	* EXT

Table 2-4 APLIBR64 Options (cont.)

APLIBR64 OPTION	MODIFIER
DIAGnostics(level)	
	* WARNing
	ERRor
	SEVERE
	TERMinal
* ERRors(errfile)	
NOERRors	
FAILure(level)	
	WARNing
	* ERRor
	SEVERE
	TERMinal

The APLINK64 command is entered using the following form:

APLINK64 txtfile1 [,txtfilen]... (option (modifier) option (modifier1, modifier2)

Table 2-5 lists the options available with APLINK64.

Table 2-5 APLINK64 Options

APLINK64 OPTION	MODIFIER
(filename[,filename]...	
* LIBrary	
NOLIBrary	
FORCe(name)	
OVERlay (filename)	
* NOOVErlay	
MDBase(address)	
	* 0
MDLIMit(address)	
	* FFFFFF (hex)
PSBase(address)	
	* 0
PSLIMit(address)	
	* FFFFFF (hex)
STACKSIZe(size)	
	* 0

Table 2-5 APLINK64 Options (cont.)

APLINK64 OPTION	MODIFIER
STACKBase(address)	
	* next address after user's MD text
* HASI(hasifile)	
NOHASI	
* ADC(name[,name]...)	
UDC(name[,name]...)	
* DISK(dskfile)	
	* the IMAGE file specification
MEM	
* IMage(imgfile)	
NOIMage	
SUBRoutine	
FAILure(level)	
	WARNing
	* ERRor
	SEVERE
	TERMinal
* List(lstfile)	
NOList	
SHow(modifier[,modifier]...)	
	* OBJ
	* PSECT
	* NAME
	VALUE
	XREF
	UNREF
NOSHow(modifier[,modifier]...)	
	OBJ
	PSECT
	NAME
	* VALUE
	* XREF
	* UNREF
* HEXadecimal	
OCTAL	
DIAGnostics(level)	
	* WARNing
	ERRor
	SEVERE
	TERMinal

Table 2-5 APLINK64 Options (cont.)

APLINK64 OPTION	MODIFIER
SYMbols(symfile)	
* NOSYMbols	
SYMOut(objname[,objname]...)	
LMBSize(size)	* 2048
DDBSIZE (size)	* 256
* ERRors	
NOERRors	

The APDBUG64 command is entered using the following form:

APDBUG64 imgfile

Table 2-6 lists the options available with APDBUG64.

Table 2-6 APDBUG64 Options

APDBUG64 OPTION	MODIFIER
(filename	
List(lstfile)	
* NOList	
SYMbols(symfile)	
* NOSYMbols	
APnum(number)	* 0
PSize (n)	* 4096
MDsize (n)	* 4096
Wait	
* NOWait	
TMram	
* NOTMram	

The System Job Executive (SJE) command is entered using the following form:

SJE [(options[])]

Table 2-7 lists the options available with SJE.

Table 2-7 SJE Options

SJE OPTION	MODIFIER
APnum(number)	* 0 (Note that 0 indicates any SC.)
CONTinue	None
NOCONTinue	None
ECHo	None
INput=(n)	n=file_specifier
OUTput=(n)	n=file_specifier
TIME	None

SJE also allows for various utility options, for example, "($FM(D)", which forces all output files created during this execution of SJE to be placed on disk D. These options are demonstrated in the following section.

2.2.5.2 Examples of Invoking PDS Programs

This section presents examples showing use of the PDS programs. These examples assist the user in learning to use the SC program development software, and also demonstrate some of its less obvious capabilities.

The first example shows how to assemble an APAL64 source file called MYCODE. The assembler searches all accessed disks for the file MYCODE with a filetype of APAL64:

APAL64 MYCODE (LIST FAIL(SEVERE)

This code generates an SC object module file (APTXT64) called MYCODE which goes into the same minidisk as the source, assuming that disk is accessed read/write. If not, APAL64 puts the APTXT64 file on the first read/write disk it finds. The listing file created by the list option defaults to MYCODE with a filetype of LISTING.

The FPS command line interpreter parses the command line into groups containing a filename, filetype, and filemode (specifies the CMS disk on which the file resides). The following example demonstrates this parsing:

```
APFTN64 MYFORT APF B1 (LIST(MYFORT MAP C)
```

This example compiles the file, MYFORT, which has a filetype of APF and is on disk B1. The listing created by the LIST option is placed in the file MYFORT, which has a filetype of MAP and is on disk C. Note that the user has overridden both the APFTN64 source and the listing file output default file types. Any default file type can be overridden in this manner.

The command line interpreter also allows the user to override defaults and minimize terminal activity, even while working with multiple files.

Consider the following example:

```
APLINK64 MYFIRST MYOBJ , MYSECOND (ADC(MYFIRST , MYSECOND)
```

This command links SC object code in MYFIRST MYOBJ with the object code in the file MYSECOND APTXT64.

NOTE

> The blank-surrounded commas cause the parser to interpret MYSECOND as a second file name rather than the disk for MYFIRST. The IBM DMSSCN routine, which parses the line once before giving it to the FPS command line interpreter, requires the blanks on both sides of the comma.

Three command line switches, besides the command currently being used, control CMS file activity. These switches are $FM (file mode), $BS (block size), and $UPCASE. With $FM the user can direct all output files created by the command to a particular minidisk, as in the following example:

```
APFTN64 CONVOLUT (NAMES(ALL) $FM(D)
```

Here, APFTN64 compiles the source in CONVOLUT APFTN64, placing the object code in CONVOLUT APTXT64 on disk D.

The default block size for SC object code and load module files is the same as the blocksize disk D was formatted as. (refer to Table 2-1), which the user can override through the $BS option, as in the following example:

```
APLIBR64 (INSERT(MYCODE) OUTPUT(MYLIB) $BS(2048)
```

This command creates a library (MYLIB APLIBR64) with a block size of 2048 from the object code in MYCODE APTXT64. Any block size which is a multiple of eight is valid as input. The $BS option is needed only when creating new binary files with a nonstandard block size.

NOTE

Changing the default blocksize may cause COPYIN/binary and COPYOUT/binary to be slower.

By using $UPCASE, the user guarantees that all text output files created by the command contains uppercase characters. Without using $UPCASE, lowercase characters within text output files remain lowercase.

Consider the following example:

```
APFTN64 SUBONE (LIST(UPONE) $UPCASE
```

This command compiles the SC subroutine SUBONE and creates a listing file called UPONE. Because the option $UPCASE is included in the command, the file UPONE will contain only uppercase characters.

The user can override the $FM specification any time, even in the same command line, as in the following example:

```
APLINK64 FIFFT (HASI(FIFFT FORTRAN B) IM LIST $FM(C)
```

This command places the interface between FEC and SC (HASI) on disk B, but puts the image and list files on disk C.

When executed, an ADC HASI automatically searches, not just disk C, but all accessed disks for the user's load module image file, FIFFT APIMG64. The file mode associated with a particular minidisk can change in the interval between the creation of the image file and the time that file is opened and loaded into the AP for execution. Such changes in file mode do not jeopardize execution.

2.3 SJE PROGRAM DEVELOPMENT AND EXECUTION

An SJE job executes entirely on the SC. The user develops a job, copies it over to the SC, executes it, and copies results back to the FEC.

This section provides information for the SJE programmer. Section 2.3.1 discusses program development for an SJE job, while Section 2.3.2 discusses job execution under SJE.

2.3.1 SJE Program Development

SJE program development involves three major steps. These steps are as follows:

- provide the files for the job
- compile or assemble user-supplied files
- create the executable image load module using the PDS routine APLINK64

The following sections describe these steps.

2.3.1.1 Providing the Files for the Job

Files for an SJE job come from two different sources: user-supplied files and FPS-supplied files. User-supplied files can be mainline source programs or subroutines written in either APFTN64 or APAL64. User-supplied files can also be object modules that reside in user libraries. FPS-supplied files come from FPS libraries, such as APMATH64 or UTIL64. APMATH64 contains a set of high-speed arithmetic routines, while UTIL64 contains data conversion utilities.

If during compilation or assembly the user fails to provide a file type as part of an input file, the compiler or assembler expects the appropriate file type. For example, the APFTN64 compiler expects its input files to have a file type of APFTN64, while the APAL64 assembler expects its input files to have a file type APAL64.

The files used in this SJE program development and execution example are a mainline program called SJEXMPLE that calls the subroutines DOUBLE and DISPLAY, and the APMATH64 library routine VIADD. SJEXMPLE, DOUBLE, and DISPLAY all have a CMS file type of APFTN64. VIADD is an FPS-supplied object module in the library APMATH64.

2.3.1.2 Compilation or Assembly of Source Files

After collecting the files necessary for a job, the user must compile or assemble the user-supplied source files. If the source files used in the job are written in APFTN64, then the user must compile them using the APFTN64 compiler. If any source files are written in APAL64, then they must be assembled using the assembler APAL64. Compiling or assembling source code creates SC object modules with a file type of APTXT64.

The following commands invoke the APFTN64 compiler to compile the source code routines SJEXMPLE, DOUBLE, and DISPLAY:

```
APFTN64 SJEXMPLE (APAL OCT LIST
APFTN64 DOUBLE  (APAL OCT LIST
APFTN64 DISPLAY  (APAL OCT LIST
```

If, after each APFTN64 command is entered, the file compiles correctly, SJE returns a message that indicates correct compilation. If the user encounters errors during compilation or assembly, the files must be corrected and recompiled.

Each of the above commands include PDS options. The APAL option causes the compiler to produce the APAL64 source files SJEXMPLE APAL64, DOUBLE APAL64, and DISPLAY APAL64. The OCT option causes the listing files produced by the LIST option to contain APAL64 code expressed with an octal radix.

2.3.1.3 Creating the Image Load Module Using APLINK64

After compilation or assembly produces the object modules (having a file type of APTXT64), the user must create an image load module. The image load module is a collection of object files linked together to form one module to perform one job. APLINK64 uses the object modules created during compilation or assembly as input files. In addition to these object modules, APLINK64 automatically includes FPS-supplied object modules that are referenced by user-supplied files. Thus, for this example, APLINK64 links the APMATH64 routine VIADD into the job without the user specifically supplying the file's name.

The following command creates the image load module for this example:

```
APLINK64 SJEXMPLE, DOUBLE, DISPLAY
```

The image load module created by this command is SJEXMPLE with a default CMS file type of APIMG64.

2.3.2 Execution Using SJE

Under SJE the entire job executes on the SC. Once the image load module exists, the job is ready to execute. To execute the job, the user must use the Job Definition Language (JDL) to interact with SJE and do the following:

- Invoke SJE.
- Attach to an available SC.
- Copy the image load module for the job and any related data files over to the SC. This step is not necessary if the user is going to execute the APIMG64 file from the FEC.
- Execute the job either from the SC or the FEC.
- Perform SJE interaction (if any) during the execution.
- Copy the results back to the FEC.
- Exit SJE.

2.3.2.1 JDL Command Overview

The SJE user controls and executes SC jobs through JDL commands. JDL is an interpretive command language that can be used to gain exclusive access to an SC, transfer files between the SC and the FEC, and run and debug programs in the SC.

JDL statements divide into two classes: control statements and service request statements. JDL control statements allow the user to gain access to an SC, release the SC, abort SC programs, gain attention of SJE, continue execution of an interrupted program, debug SC programs using APDBUG64, and exit from SJE. JDL service request statements allow the user to copy files in and out of the SC file system, execute SC load modules that reside in the SC file system, and manipulate and examine permanent files and directories on the D64.

Permanent files are identified by names that are listed in directories. Each user can have a private directory of permanent files, or several users can share a directory.

Directories form a five-level hierarchy, with the System Directory as the top level and user directories at subsequent levels. Files may reside in the System Directory or in user directories and can contain programs or data. Directories are not considered to be files and cannot be opened, closed, read, or written.

Table 2-8 and Table 2-9 give the syntax for each available JDL statement and briefly describe its use. For more information on the JDL command options, refer to Volume 1 of the FPS-164 Operating System Manual Set, listed in Table 1-1.

NOTE

When using either the JDL command COPYIN or COPYOUT, text files on the FEC can have a maximum record length of 1024 bytes. The COPYOUT command splits records longer than 1024 bytes into multiple records. The COPYIN command truncates records longer than 1024 bytes. SJE gives no notice when splitting or truncation of records occurs.

Table 2-8 JDL Control Statements

COMMAND	DESCRIPTION
ATTach [/Wait] [/TMram] [n] [/Priority=n]	Allows the user to wait for an SC if an SC is unavail able.
ABort	Terminates the executing SC program.
CONtinue	Resumes execution of an interrupted SC program.
DEBug [/Now | /Defer]	Invokes the FPS Interactive Symbolic Debugger (refer to The APDEBUG64 Manual listed in Table 1-1.
DETach	Allows the user to release the SC.
Quit	Allows the user to quit interacting with SJE and return to the FEC command level.

Table 2-9 JDL Service Request Statements

COMMAND	DESCRIPTION
ACcess <directory-name>	Sets the current directory allowing access to it and the files in the directory.
CHange [/options] <file-name>	Changes the attributes of a file on the D64.
CLEAR [/MM=value] [/TM=value] [/ALL]	Allows the user to initialize the main memory or table memory RAM (TMRAM) to the specified value.
COpy [/options] <source-name> <dest-name>	Copies files and/or directories from one place to another in the SC file system.
COPYIn [/Binary] [/DRives=[()]XY[,...)]] <source-name> [,<dest-name>]	Allows the user to copy a text or binary file from the FEC file system to the SC file system.
COPYOut [/Binary] <source-filename> [,<dest-filename>]	Allows the user to copy text or binary files from the SC file system to the FEC file system.
CReate [/options] <file-name>	Creates files or directories in the SC file system.
DElete [/ALl] [/DIrectory] <file-name-1>,...<file-name-n>	Removes the specified files or directories from the SC file system.
DIrectory [/option] [<file-name>,...]	Lists information concerning the specified files.
HELP [command]	Allows the user to display information on how to use a specific JDL command.

Table 2-9 JDL Service Request Statements (cont.)

COMMAND	DESCRIPTION
HOST ['] host_command [']	Allows the user to execute one FEC operating system command without terminating SJE execution.
PREserve [/paramenters] [<input-specifier>]	Saves one or more SC files in one FEC file on the FEC.
REName [/RKey=key] [/OKey=key] <old-name> [<new-name>]	Changes the name and/or read and owner keys of the specified file or directory.
REStore [/paramenters] [<output-specifier>]	Restores FMS files and directories from a preserve file on the FEC created by the PRESERVE command.
SEt [/options]	Sets certain characteristics of the user job.
SHow [/options]	Reports information such as the currently accessed directory, the status of the disks, and the SC CPU limit time to the user.
Type <file-name>>	Prints a text file to the terminal.

The JDL HOST command allows users to execute most CMS SUBSET commands from within SJE. The user can also use the HOST command to execute all control program (CP) commands. The following list contains the CMS SUBSET commands not allowed by the HOST command.

HX

START

LOAD

FLIST

NUCXDROP

The following examples show use of the HOST command.

```
R;
sje
SJE-I-WELCOME, SJE REL F00-000  VER 1.0      02/15/85  15:20
SJE>
att 2
DEV  160 ATTACHED
AP Job Number = 29.
The SUM currently executing is: F00-000.
SJE-I-ATTACH, Assigned AP number   2, priority=   3, jobnum=   29.
SJE>
host query time
APCMS SUBSET
TIME IS 15:21:38 PDT FRIDAY 02/15/85
CONNECT= 00:09:20 VIRTCPU 000:01.84 TOTCPU= 000:06.15
SJE>
host access 191 a
APCMS SUBSET
DMSACC724I '191' REPLACES ' A (500) '
SJE>
```

2.3.2.2 Executing an SJE Job from the SC

This section presents the SJE execution process from the SC for the sample job developed in Section 2.3.2. Input supplied by the user is underlined. First, the user invokes SJE:

```
SJE
SJE-I-WELCOME, SJE REL xxx-yyy VER zzz mm/dd/yy hh:mm
SJE>
```

NOTE

The values within the SJE welcome response are defined as follows:

xxx-yyy is the SJE release invoked

zzz is the SJE version invoked

mm/dd/yy is the current month, day, and year

hh:mm is the current hour and minute of the day

The user now attaches specifically to SC number three:

```
SJE>
ATT 3
DEV 160 ATTACHED
SJE-I-ATTACH, Assigned AP number 3, Priority=n, Jobnum=xxx
```

where:

xxx is the AP job number used if the user wants to display information about the job from the AP operator (APOPR).

N is the priority queue the user was placed in.

Next, the user copies the image load module 'SJEXMPLE' and the related data file DFILE1 to the SC:

```
SJE>
COPYIN/BINARY 'SJEXMPLE APIMG64',SJEXMPLE
SJE-I-COPYIN, File copied in.
COPYIN/BINARY DFILE1,DFILE1
SJE-I-COPYIN, File copied in.
```

Note that the IBM file name must be in single quotes in COPYIN/binary and COPYOUT/binary commands unless the file has a file type of BIN (binary file). When the file type is BIN, the user need only specify the file name. If the user wants to COPYIN or COPYOUT text files without putting the file spec in quotes, the text file must have a filetype of "TXT". Also, the user can enter filenames in either uppercase or lowercase. At this point, the user executes the program by typing in its SC file name:

```
SJE>
SJEXMPLE
```

After program execution begins, several things can occur. The job can call for interactive statements as required input. The job can also display results to the user's terminal, and transfer program results into a separate file. Once program results are in a file, the user can enter the JDL command COPYOUT to transfer the file over to the FEC. In any case, after the job completes, SJE displays the following exit message:

```
SJE-I-EXIT, Program exit.
```

Finally, the user detaches the SC and quits SJE:

```
SJE>
DETACH
SJE-I-DETACH, AP detached.
SJE>
QUIT
SJE-I-QUIT, SJE stopped.
```

2.3.2.3 Executing an SJE Job From the FEC

This section presents the SJE execution process from the FEC for the sample job developed in Section 2.3.2. Input supplied by the user is underlined.

First, the user invokes SJE:

```
SJE
SJE-I-WELCOME, SJE REL xxx-yyy VER zzz mm/dd/yy hh:mm

SJE>
```

The user now specifically attaches to SC number three:

```
SJE>
ATT 3
DEV 160 ATTACHED
SJE-I-ATTACH, Assigned AP Number 3, Priority = n, Jobnum=xxx
```

Next, the user copies the related data file DFILE1 to the SC:

```
SJE>
COPYIN/BINARY DFILE1,DFILE1
SJE-I-COPYIN, File copied in.
```

Note that the IBM file name (DFILE1) is given the same file name when it is copied into the SC. Note that DFILE1 must have a filetype of TXT to use COPYIN in this manner. Also, the user can enter file names in either uppercase or lowercase. At this point, the user executes the program by typing in its IBM file name preceeded by :HOST:.

```
SJE>
':HOST:SJEXMPLE APIMG64'
```

After program execution begins, several things can occur. The job can call for interactive statements as required input. The job can display results to the user's terminal, and transfer program results into a separate file to be copied back to the FEC. In any case, after the job completes, SJE displays the following exit message:

```
SJE-I-EXIT, Program exit.
```

Finally, the user detaches the SC and quits SJE:

```
SJE>
DETACH
SJE-I-DETACH, AP detached.
SJE>
QUIT
SJE-I-QUIT, SJE stopped.
```

2.3.3 Executing an SJE Job in CMS Batch Mode

To execute an SJE job using CMS batch, the user creates an EXEC command file that contains all of the FEC and SJE commands.

For example, an EXEC command file can appear as shown below:

```
*
*
*     COMMAND FILE FOR SJE BATCH JOB
*               SJEXMPLE
*
*
APFTN64 SJEXMPLE APFTN64
APFTN64 DOUBLE APFTN64
APFTN64 DISPLAY APFTN64
APLINK64 SJEXMPLE, DOUBLE, DISPLAY
&BEGSTACK
 ATTACH/WAIT
 COPYIN/BINARY 'SJEXMPLE APIMG64',SJEXMPLE
 COPYIN/BINARY DFILE1,DFILE1
 SJEXMPLE
                .
                .
 user interaction statements (if any)
                .
                .
 COPYOUT/BINARY DFILE1,DFILE1
 COPYOUT 'LFILE LISTING',LFILE
 DETACH
 QUIT
&END
SJE
```

In the above example, the three APFTN64 commands compile the source code files into object modules. The APLINK64 command then takes the object modules and produces an executable image load module. The EXEC command file then invokes SJE. When SJE is invoked, it uses the JDL commands supplied in the command stack located before the actual SJE command. The first command in the stack attempts to attach to an available SC. Upon successfully attaching to an SC, SJE copies the image load module and the data file into the SC. Next, the EXEC instructs the SC to execute the program. If the job requires user interaction during execution, the user also supplies these interactive commands or data in the stack. The EXEC then instructs SJE to retrieve the resulting files. Finally, the EXEC instructs SJE to detach from the SC and to quit.

For information on how to submit an EXEC command file for execution, refer to VM/SP CMS User's Guide, listed in Table 1-2.

2.3.4 SJE User Attention Command

To interrupt an SC program execution under SJE, the user must issue an attention interruption at the terminal, as described in the IBM VM/SP CMS User's Guide (refer to Table 1-2). Once the attention command is issued, SJE can take up to 15-20 seconds to get the SC's attention and interrupt the executing program. After the interrupt occurs, SJE prompts the user for the next command, which must be one of the following JDL commands: ABORT, DEBUG/NOW, or CONTINUE.

NOTE

When a user's job is temporarily transferred from the SC to the disk subsystem (rolled out), and the user is executing under SJE, an attention interrupt acts the same as it does when the job is executing. However, the attention interrupt is not processed until the job is transferred back to the SC from the disk subsystem (rolled in). Consequently, the wait can be much longer than 15-20 seconds.

Sometimes the user can experience difficulty interrupting SJE or APEX64 and returning to the SJE or CMS command level. When this difficulty occurs, the user can stop a run on the SC by entering the following command at the virtual console:

```
#CP EXT 1
```

NOTE

This assumes the user's terminal line end is set to the default "#". Terminal line end is set using the CP TERM LINEND command.

This command results in an external interrupt that has an interrupt code of 1. An external interrupt handler intercepts the interrupt and terminates all I/O to the SC by issuing a HALT DEVICE instruction to the device address through which the user communicates with the SC. This external interrupt immediately releases and detaches the SC, aborts APEX64 or SJE, and returns the user directly to the IBM/CMS operating system.

If #CP EXT 1 fails to return the user to the CMS environment, it may be necessary to abort the AP job using #CP IPL CMS. This command will always abort the AP jobs and return the user to the CMS environment. However, aborting a job in this manner will cause some of the AP accounting data to be lost and therefore should only be used if the prior two steps fail.

2.4 APEX64 PROGRAMMING

This section explains APEX64 FEC programs that use auto-directed calls (ADC) and user-directed calls (UDC). Following the discussion on FEC programs, this section also provides detailed information on APEX64 program development and execution.

2.4.1 UDC/ADC FEC Programs

A FEC mainline program can employ either the ADC method or the UDC method. Although UDC maximizes user control of operations, it is more complex to program with than ADC.

ADC programming is synchronous, causing the SC to wait while the FEC is running, and the FEC to wait while the SC is running.

Using UDC enables the FEC main program and the SC subroutine to run at the same time. The user's program on the FEC manages data transfers between the FEC and the SC. The user's FEC program also controls when SC subroutine execution begins.

Whether the ADC or UDC method is used, APEX64 can execute in one of four modes: step, substep, chain, and automatic. Automatic mode is the default, and chooses between chain, step, and substep mode. The choice between modes is based on the size of the load module and the size of the data arrays referenced by the load module. The FEC FORTRAN program can call the APEX64 routine APMODE to set the APEX64 execution mode. APEX64 execution modes optimize the use of FEC real memory in virtual memory machines. For more information on execution modes, refer to Volume 2 of the FPS-164 Operating System Manual Set listed in Table 1-1.

2.4.1.1 Using Auto-directed Calls (ADC)

The following code is a sample FORTRAN program that executes on the FEC. The numbers appearing on the right in this sample are not part of the statements, but are line numbers added here for the reader's convenience. This program calls subroutines for execution on the SC, and uses the ADC programming method to call APEX64 subroutines. This code demonstrates how to attach to and initialize an SC, how to perform calls to subroutines that execute on the SC, and one of two methods for releasing the SC.

```
C                                                              1
C     FILE ADCXMPLE APFTN64
C    INTEGER APNUM, ACTION, FOURK, ASSGN, STAT
                  .
                  .                                            5
                  .
```

```
      program initialization
           and flow
                                                                    10
                .
                .
                .
      FOURK = 4096
      APNUM = 0
      ACTION = 0
C
C     APNUM ATTACHES TO ANY AVAILABLE SC
C     ACTION WAITS IF THE SC(S) ARE BUSY
C     FOURK ALLOCATES 4K FOR MD AND PS ON THE SC
C
      CALL APINIT (APNUM,ACTION,FOURK,FOURK,ASSGN,STAT)             20
C
C     HANDLES A FAILURE TO ASSIGN
C
      IF (ASSGN .EQ. 0) CALL APSTOP (STAT)
      CALL DOUBLE (A,B,C,N)                                         25
C
C     CALL APROUTINE VECTADD WHICH CALLS VIADD
C
      CALL VECTADD (A,B,C,N,ONE)
C                                                                   30
C     RELEASE THE SC AND DISPLAY RESULTS
C
      CALL APRLSE
      CALL DISPLAY (A,B,C,N)
      END                                                           35
```

The FEC program first uses the CALL APINIT statement (line 20 in the above code) to call the APEX64 routine that initializes an SC and assigns it to the user. Then the FEC program uses the CALL DOUBLE and CALL VECTADD statements (lines 25 and 29 in the above code) to call SC subroutines. Control does not return to the FEC program until the SC subroutines have finished executing. When DOUBLE and VECTADD have finished execution on the SC, the program calls the APEX64 routine APRLSE (line 33 in the above code). This call releases the SC and makes it available to other users. The ADC user can also call APIBMR to release the SC. (APIBMR is described in Section 2.4.1.2.) Finally, the program calls the FEC subroutine DISPLAY. Note that when using ADC, the FEC program calls subroutines that execute on the FEC in the same manner that calls are made to subroutines that execute on the SC.

2.4.1.2 Using User-directed Calls (UDC)

The UDC method gives the user more control over SC operation. With UDC, the user calls APEX64 routines to explicitly control data transfers between the FEC and the SC. The user can also control when subroutines start executing on the SC. Because of this control, a programmer can cause SC routines to execute simultaneously with FEC subroutines, and SC subroutines to execute while the data for another SC subroutine is being transferred to the SC.

The following code is a sample FORTRAN program named UDCXMPLE that demonstrates programming with UDC. This program contains the APEX64 calls that control data transfers and SC execution that are not necessary when programming under ADC. The numbers appearing on the right in this sample are not part of the statement, but are line numbers added here for the reader's convenience.

This program also demonstrates simultaneous execution of SC subroutines and FEC subroutines, placing the SC in chain mode, and placing the SC in step mode.

```
C                                                                    1
C  FILE UDCXMPLE
C
C
C                                                                    5
C
C
      INTEGER SUM,CHAIN,step,MCP,ANY,WAIT,FOURK,ASSGN,STAT
      INTEGER N,ONE,FMTI,FROMAP,TOAP,AADR,BADR,CADR,NADR
      INTEGER STADR,DDBLEN,NOINT,DDB1(50),DDB2(50)                  10
      INTEGER A(100),B(100),C(100),D(100)
C
      DATA DDBLEN,CHAIN,step,MCP,FMTI /50,1,2,1,0/
      DATA NOINT,FROMAP,TOAP,ONE,N,ANY /0,0,1,1,100,0/
      DATA WAIT,FOURK /0,4096/                                       15
C
C  LOADING VECTOR ARRAYS
C
      DO 10 I=ONE,N
           A(I)=I                                                    20
           B(I)=I+10
           C(I)=I+20
           D(I)=I+30
10    CONTINUE
C                                                                    25
C  ESTABLISHING TRANSFER ADDRESS IN THE SC
C
      AADR=50
      BADR=AADR+N
      CADR=BADR+N                                                    30
      NADR=CADR+ONE
      STADR=NADR+ONE
C
```

```
C  INITIALIZING THE SC
C
      CALL APINIT (ANY,WAIT,FOURK,FOURK,ASSGN,STAT)          40
      IF (ASSGN .EQ. 0) CALL APSTOP (STAT)
C
C  SETTING SC EXECUTION MODE
C                                                            35
      CALL APMODE (CHAIN)
C
C  SPECIFIES DATA TRANSFER TO THE SC
C
      CALL APIDB (DDB1,DDBLEN)                               45
      CALL APPUT (DDB1,A,AADR,N,FMTI)
      CALL APPUT (DDB1,B,BADR,N,FMTI)
      CALL APPUT (DDB1,N,NADR,ONE,FMTI)
      CALL APPUT (DDB1,ONE,STADR,ONE,FMTI)
      CALL APXDDB (DDB1,NOINT,TOAP)                          50
C
C  CALLS A VECTOR INTEGER ADD ROUTINE
C
      CALL VIADD (AADR,STADR,BADR,STADR,CADR,STADR,NADR)
C                                                            55
C  START THE CHAINED MAIN CHANNEL PROGRAM EXECUTING
C
      CALL APSTIO (MCP)
C
C  CALL A FEC SUBROUTINE FOR EXECUTION                      60
C
      CALL HOSTSUB (D,N,SUM)
C
C
C  CAUSE THE FEC TO POSTPONE EXECUTION UNTIL THE
C  MAIN CHANNEL PROGRAM IS FINISHED EXECUTING               65
C
      CALL APWR
C  PLACE THE SC INTO step MODE
C                                                            70
C  BUILD AND EXECUTE THE DATA DESCRIPTOR BLOCK
C  THAT GOVERNS THE TRANSFER OF THE RESULTS FROM
C  THE SC TO THE HOST
C
      CALL APMODE (step)                                     75
      CALL APIDB (DDB2,DDBLEN)
      CALL APGET (DDB2,C,CADR,N,FMTI)
      CALL APXDDB (DDB2,NOINT,FROMAP)
C
C  RELEASE THE SC                                           80
C
      CALL APWD
      CALL APRLSE
```

The following steps explain the general procedure needed for an FEC FORTRAN program that uses UDC. The steps use statements from the above file UDCXMPLE to explain the various APEX64 calls. The line numbers shown to the right of the APEX64 calls are not part of the executable statements; these numbers correspond to the code in the sample file UDCXMPLE just cited. The user can use these numbers as an aid in locating the statement under discussion. For more explicit information on APEX64 commands and their parameters, refer to Volume 2 of the FPS-164 Operating System Manual Set, listed in Table 1-1.

1. Unlike an ADC program, the UDC program must specifically designate where data is placed and kept in SC memory during and after a data transfer. The user accomplishes this task by designating relative SC addresses for the blocks of data being used in an SC subroutine. The file UDCXMPLE designates five addresses in the SC, by using the following statements:

```
AADR=50
BADR=AADR+N
CADR=BADR+N                                          30
NADR=CADR+N
STADR=NADR+ONE
```

   The variables AADR, BADR, CADR, NADR, and STADR represent the beginning addresses in SC memory in which the data arrays A,B,C and data elements N and ONE reside, respectively.

2. The user can determine the mode in which the SC executes. If the user does not specify an SC execution mode the SC defaults to automatic mode. When the SC is executing in automatic mode, APEX64 chooses one of the following modes to operate in: CHAIN or step. The choice is based on load module size and the size of the data areas specified in the data descriptor block (DDB). Refer to Volume 2 of the FPS-164 Operating System Manual Set listed in Table 1-1 for more information on SC mode selection.

   The file UDCXMPLE first calls APMODE to place the SC into chain mode. Chain mode is selected to demonstrate how the SC subroutine VIADD and the FEC subroutine HOSTSUB execute simultaneously. After the SC is put into mode, subsequent calls to APPUT, APXDDB, and SC subroutines do not execute immediately, but are chained together for future execution. UDCXMPLE uses the following call statement to place the SC into chain mode. To specify chain mode, the parameter CHAIN must be equal to 1.

```
CALL APMODE (CHAIN)                                  36
```

3. Before the SC can execute, the user must assign and initialize an available SC. Initialization is performed exactly as it is when programming with ADC. The user calls the APEX64 routine APINIT. UDCXMPLE uses the following statement to assign and initialize an SC:

```
CALL APINIT (APNUM,ACTION,FOURK,FOURK,ASSGN,STAT)    40
```

4. After SC initialization occurs, the FEC program begins preparing for an FEC-to-SC data transfer. To accomplish the data transfer the FEC must build a description of the transfer in a data descriptor block (DDB). The DDB is built by first calling APIDB to initialize the DDB in which the transfer information is placed. Next, calls to APPUT describe the data to be transferred from the FEC to the SC. The parameters within the APPUT calls specify where to get the data in FEC memory, where to place the data in the SC, how much data is being sent, and the type of data being transferred. The APPUT calls do not initiate the data transfer; the calls add information to the FEC-resident DDB. The DDB stores the information for the APPUT calls until the FEC main program executes APXDDB. UDCXMPLE uses the following code for its first DDB build:

```
CALL APIDB (DDB1,DDBLEN)                              45
CALL APPUT (DDB1,A,AADR,N,FMTI)
CALL APPUT (DDB1,B,BADR,N,FMTI)
CALL APPUT (DDB1,N,NADR,ONE,FMTI)
CALL APPUT (DDB1,ONE,STADR,ONE,FMTI)
```

When executed, DDB1 transfers array A, array B, the variable N, and the variable ONE to the relative addresses AADR, BADR, NADR, and STADR in SC memory.

5. After a DDB has all the descriptions for a transfer, the transfer can be made. The FEC program calls the APEX64 routine APXDDB to execute a DDB and start the data transfer. APXDDB uses the information in the DDB to build an FEC channel program that transfers the data to the SC.

   The moment the transfer actually occurs depends on the mode in which the SC is executing. If the SC is in step mode, the call to APXDDB executes the transfer immediately. If the SC is in chain mode (refer to the UDCXMPLE in this section), execution of the transfer is delayed. In the delayed case, the call to APXDDB is chained into the main channel program (MCP) and executed later by a call to the APEX64 routine APSTIO. UDCXMPLE uses the following statement to chain APXDDB for execution:

```
CALL APXDDB (DDB1,NOINT,TOAP)                         50
```

6. After the transfer of data for an SC subroutine is either complete or chained into the MCP, the FEC program can call the SC subroutine. This call transfers the subroutine code to the SC. The code transferred is in the form of a UDC HASI file. (Refer to Section 2.4.2.4 for information on creating HASI files during APEX64 program development.) Unlike an ADC HASI file, a UDC HASI file contains no data transfer information. A UDC HASI file contains only the information needed to transfer the subroutine itself. The data the subroutine operates on is transferred through calls to APIDB, APPUT, and APXDDB. When the FEC program calls this HASI file, it uses SC main memory addresses to refer to all the parameters used by the SC subroutine. UDCXMPLE uses the following statement to call the SC subroutine VIADD:

```
CALL VIADD (AADR,STADR,BADR,STADR,CADR,STADR,NADR)  54
```

   Again, because UDCXMPLE has placed the SC in chain mode, the call to VIADD does not execute immediately. The call is chained into the MCP following the APXDDB call that was chained into the MCP during step 5.

7. At this point, UDCXMPLE has built its MCP. The FEC subroutine HOSTSUB and the SC subroutine VIADD can now execute simultaneously, and the FEC can perform functions that do not interfere with the subroutine running in the SC. Although this example does not demonstrate it, the FEC can also transfer data to or from areas of SC main memory that the SC is not using while an SC subroutine executes.

   To start execution of the MCP, UDCXMPLE calls the APEX64 routine APSTIO. This call executes the MCP from beginning to end. For UDCXMPLE, APSTIO first executes the APXDDB that performs the data transfer described by DDB1 in step 4 and then executes the call to VIADD. UDCXMPLE uses the following statement to begin MCP execution. To specify MCP execution, the variable MCP must be equal to 1.

```
CALL APSTIO (MCP)                                   58
```

   After the MCP execution has begun, the FEC is free to execute FEC-related tasks. UDCXMPLE immediately calls HOSTSUB to demonstrate that a FEC subroutine can execute at the same time an SC subroutine executes.

8. When the FEC program needs the results of an executing SC subroutine, it calls the APEX64 routine APWR. APWR causes the FEC main program to suspend further FEC action until the SC finishes execution of an SC subroutine. In this example, UDCXMPLE calls APWR to ensure the SC has finished executing VIADD before attempting to transfer results back to the FEC. The call statement used is shown below:

```
CALL APWR                                           67
```

9. At this point, UDCXMPLE places the SC into step mode for execution. While in step mode, the SC executes DDB's and SC subroutines at the time the calls to APXDDB and the subroutines are made. Thus, no chaining together of commands occurs while the SC is in step mode. UDCXMPLE uses the following call to place the SC into step mode, where step must be equal to 2:

```
CALL APMODE (step)                                    75
```

10. When the FEC program is ready to retrieve data from the SC, the program initializes a DDB by calling APIDB and performing APGET calls. Like APPUT calls, APGET calls do not actually cause SC-to-FEC data transfers, but place descriptions of the transfers in FEC-resident DDB's. UDCXMPLE uses the following statements to build the DDB that controls the data transfer:

```
CALL APIDB (DDB2,DDBLEN)                              76
CALL APGET (DDB2,C,CADR,N,FMTI)
```

When executed, DDB2 transfers N elements of the resulting sum array beginning from the relative SC address CADR to the FEC array C.

11. After DDB2 is built, the FEC program calls APXDDB to perform the data transfer described by DDB2. UDCXMPLE uses the following statement to begin the data transfer:

```
CALL APXDDB (DDB2,NOINT,FROMAP)                       78
```

Because the SC is now in step mode, the call to APXDDB begins the data transfer.

12. When the transfer begins, the FEC program calls the APEX64 routine APWD. APWD causes the FEC to suspend further execution until the current data transfer is complete. UDCXMPLE uses the following statement to suspend execution:

```
CALL APWD                                             82
```

13. Once the FEC finishes using the SC, the program releases it by calling either APRLSE or APIBMR. UDCXMPLE uses the following statement to release the SC:

```
CALL APRLSE                                           83
```

2.4.2 Program Development for APEX64

This section provides an APEX64 program development example.

The text uses actual FEC commands that run PDS programs, build IBM object modules, and build executable load modules. The commands operate on a fictitious set of files, which includes:

- an FEC FORTRAN program called MYJOB that uses UDC
- two SC subroutines APJOB1 and APJOB2 that are coded in APFTN64
- the subroutine HOSTSUB that executes on the FEC
- APMATH64 library routine VIADD that executes on the SC with APJOB1 and APJOB2

Although this example assumes an FEC program that uses UDC, the development process for a job that uses ADC is identical to a job that uses UDC, with one exception that is explained in Section 2.4.2.4.

An APEX64 program is developed in eight steps. Some of these steps are not necessary, such as building libraries with APLIBR64 and invoking APDBUG64. However, these steps are included in this example for users who wish to employ them.

The following list summarizes the program development process:

- Supply the necessary files for the job. These files include the FEC program, all subroutines executing on the FEC and the SC, and any routines used from libraries.
- Compile or assemble subroutine code for execution. The PDS compiler APFTN64 is used for subroutines written in AP FORTRAN, while APAL64 is used for subroutines written in SC Assembly Language.
- Using APLIBR64, combine subroutines that excute on the SC into libraries. This optional step makes handling of files easier.
- Create an Host/SC Software Interface (HASI) file using APLINK64 for the assembled or compiled object files.
- Invoke APDBUG64 to debug SC subroutines as they execute on an SC. This step is optional.
- Create FEC object modules for the FEC program and SC subroutine HASI files by compiling or assembling them using FEC facilities.

- Create an FEC-executable load module using the GLOBAL command to link the object modules and necessary text libraries together. APDBUG64 can be included in the load module to allow for job debugging.

- Execute the FEC program.

2.4.2.1 Supplying the Files

To begin APEX64 program development, the user must provide the necessary files. Several types of files can be involved in an APEX64 job. These file types include: a user-supplied FEC program, user-supplied subroutines that execute on the SC, user-supplied subroutines that execute on the FEC, and FPS-supplied library routines that execute on the SC.

The FEC program, which makes calls to APEX64 routines and subroutines, can be written in either FEC FORTRAN, FEC assembler language, or any other language that uses FORTRAN calling structures. If the FEC program is written in FORTRAN, its CMS file type must be FORTRAN. If the FEC program is written in FEC assembler language, its CMS file type must be ASSEMBLE.

The subroutines can execute on either the FEC or the SC. Subroutines that execute on the FEC are also written in either FEC FORTRAN, FEC assembler, or any other language that uses FORTRAN calling structures. The CMS file type for these higher-level subroutines is FORTRAN, while the CMS file type for subroutines written in FEC assembler is ASSEMBLE.

Similarly, the subroutines that execute on the SC are written in either APFTN64 (a superset of FORTRAN 77) or APAL64. For more information on FORTRAN 77, refer to the American National Standard Programming Language of FORTRAN, listed in table 1-3. The CMS filetype needed for subroutines written in SC FORTRAN (APFTN64), while the filetype is APAL64 for subroutines written in APAL64.

A third set of files can be used in an APEX64 job; these files are routines from FPS-supplied libraries. These libraries include a set of high-speed arithmetic routines in the library APMATH64, a set of data conversion utilities in the library UTIL64, and a set of lower-level FORTRAN I/O routines in the libraries APRLIB64 and APSLIB64. Some restrictions do exist in using the FPS-supplied routines. The data conversion utilities in UTIL64 are called from FEC FORTRAN programs. The routines in APMATH64 can be called from either an FEC FORTRAN program or a subroutine executing on the SC. The APRLIB64 and APSLIB64 routines are explicitly called only from subroutines coded in APAL64 executing on the SC.

2.4.2.2 Compilation or Assembly for SC Subroutines

After the user supplies the files for a job, the user must compile or assemble the subroutines that execute on the SC. The user must use APFTN64 to compile subroutines written in FORTRAN, and use APAL64 to assemble subroutines written in APAL64.

For this sample job, the user must compile the source files APJOB1 and APJOB2. The user enters the following commands to perform this compilation:

```
APFTN64 APJOB1 APFTN64 (LIST (AJOB1LIS) DEBUG
APFTN64 APJOB2 (LIST (AJOB2LIS)
```

The above commands create two object modules with file types of APTXT64 from the files APJOB1 and APJOB2. The first APFTN64 command compiles the file APJOB1 using the LIST and DEBUG options. The LIST option causes the compiler to provide a listing file of APJOB1 in a file called AJOB1LIS that has a file type of LISTING. The DEBUG option prepares APJOB1 for a debugging session by causing the compiler to save all the local symbols in the object module, provide line numbers in error abort traceback, and run the APFTN64 compiler at an optimization level of 0. The second APFTN64 command similarly compiles the file APJOB2 while using the LIST option.

2.4.2.3 Building Libraries

At this point, the user can place the object modules of SC subroutines into libraries. Combining subroutines into libraries enables a user to reference just the library instead of numerous files during the APLINK64 step. For example, consider an ADC mainline program that calls seven individual subroutines for execution on the SC. If these subroutines are not in a library, the APLINK64 statement must reference each subroutine by name. On the other hand, if a library contains the seven subroutines, APLINK64 need only reference that library name to gain access to all seven subroutines.

A library can contain SC subroutines that are not used in the job. APLINK64 uses only the subroutines referenced by the mainline program.

This sample job uses the following APLIBR64 statement to add SC subroutines to an existing library:

```
APLIBR64 LIBAP APLIBR64 (INSERT (APJOB1 , APJOB2)
```

This command places the two SC subroutine object modules APJOB1 and APJOB2 into the existing library LIBAP.

2.4.2.4 Creating the HASI File Using APLINK64

After the user-supplied files that execute on the SC are compiled or assembled, the user uses APLINK64 to produce HASI files from object modules.

The image file is an SC load module that contains SC-executable microcode. The HASI controls FEC-to-SC transfers for the subroutines that are to be executed on the SC. If the calling program employs UDC, calls to APEX64 within the FEC program control when SC subroutine data is transferred to the SC, and where SC subroutine data is transferred in the SC. UDC FEC programs also control when SC routines are executed. (Refer to Section 2.4.1.2 for detailed UDC information.)

If APDBUG64 is to be used in testing SC subroutines, the user may also use the APLINK64 option SYM to create symbol tables.

This example uses three routines that execute on the SC. These routines are APJOB1, APJOB2, and the APMATH64 routine VIADD. APLINK64 must create UDC type HASI file(s) for each of these routines. This sample job uses the following APLINK64 statements to build the HASI files:

```
APLINK64 LIBAP (UDC (APJOB1 , APJOB2) SYM (APSYMS)
APLINK64 (UDC (VIADD)
```

The first APLINK64 statement produces a UDC-type HASI for APJOB1 and APJOB2. The next APLINK64 statement creates a UDC-type HASI for the FPS-supplied APMATH64 routine VIADD.

NOTE

The APLINK64 statement mentions the files APJOB1 and APJOB2 because this sample job uses UDC mode. In ADC mode such files need not be mentioned. By providing the library name only, APLINK64 automatically includes all subroutines referenced by the ADC routines or any routines including APROUTINE or APFUNCTION declarations.

2.4.2.5 Debugging SC Subroutines

At this point, the user can invoke APDBUG64 to verify SC subroutines. Invoking APDBUG64 enables the user to debug SC subroutines on an actual SC.

To use a file with APDBUG64, certain options can be included during the compilation or assembly and APLINK64 steps. For subroutines written in FORTRAN, the DEBUG option can be included during the compilation step. For subroutines written in APAL64, the NAMES (UNREF) option and modifier can be included during the assembly step. During the APLINK64 step, the user can include the SYM option.

CHAPTER 3

IBM/CMS SPECIFIC SOFTWARE OF THE FEC/SC SYSTEM

3.1 INTRODUCTION

This chapter presents additional software information that describes the allocation of an SC, shared SC execution time, and communicating with FEC files from the SC. The text assumes that the reader is an experienced CMS system user and is already familiar with how an SC operates. When in doubt about how to use an element of the SC software, the reader can consult the appropriate publication listed in Table 1-1.

3.2 ALLOCATION OF AN SC

This section discusses SC allocation. Included topics are the role of the FPS-164 Scientific Computer Manager Program (APMGR), number of users able to attach to an SC, selection of the SC, releasing and detaching the SC, and forced release processing.

Although this section primarily discusses topics that are functionally invisible to the user, some user-visible SC allocation commands are included.

3.2.1 The Role of the APMGR

Since the SC and FEC communicate as though the SC were an I/O device, an I/O path must exist between the SC and the CMS virtual machine that is attempting to access the SC through APEX64 or the SJE. The CP ATTACH command allows the user to make one of the device addresses recognized by the SC become part of the user's virtual machine configuration, and establishes the needed I/O path. However, since CMS users seldom possess the CP command privilege allowing them to execute the ATTACH command directly from their own virtual machine, an APMGR virtual machine executes the CP ATTACH command on their behalf. The APMGR virtual machine runs continuously in a disconnected state, communicating with CMS users through the virtual machine communication facility (VMCF).

To gain access to an SC, the APEX64 user calls APINIT and the SJE user issues the SJE ATTACH command (not to be confused with CP ATTACH). The user can specify either that a specific SC be attached or that any SC be attached. If the user specifies a particular SC, and if at least one of the addresses recognized by that SC is available (refer to Section 4.4.1), then the APMGR issues a CP ATTACH command for the first

available address and dedicates that address to the user's virtual machine. This procedure establishes the needed data path between the SC and the CMS virtual machine.

APEX64 completes the assignment process after the SC is attached by executing an assign channel program to the SC. The SC's Host Interface Support Processor (HISP) (refer to Section 4.2.3) interprets the assign channel program and grants the user's virtual machine access to the SC. If another user has assigned the SC, the user can wait for it to become available. In this case, the APMGR places the user into one of its internal priority queues. When the user reaches the head of the queue and no other users are waiting at a higher priority, the APMGR issues the CP ATTACH command to attach an AP device address to the user's virtual machine. The APMGR then sends a VMCF message to the waiting user to wake him up and tell him he now has access to the AP.

3.2.2 Number of Users

The number of users that can simultaneously wait for access to a given SC depends on the number of real device addresses physically configured for that SC and the MAXusers values set in the APMGR. An SC can be configured to recognize a range with a maximum of 8, 16, or 32 device addresses. Note that configuring the hardware for only one device address is no longer supported by the software. SC's reserve the first address for special APMGR processing. Note that the MAXusers values in the APMGR must not total more than the number of (addresses -1) configured in the hardware. See the AP Operator manual for a description of setting the MAXusers values for each priority queue of an AP.

Once a user has initially attached to an SC, the user shares SC execution time with other attached users. The MAXusers parameter set within the site parameter file controls the number of users able to share SC execution time. Section 3.3 describes the site parameter file and sharing of SC execution time.

3.2.3 Selecting an SC

Requesting that any available SC be assigned causes the APMGR to attach to the user's machine the first available SC in the system. The APEX64 user issues a call to APINIT, while the SJE user enters the JDL command ATTACH to assign and initialize an SC. If all SC's are busy, the APMGR places the user in the "wait queue". The user waits in that queue until an SC becomes available, provided the user requested that he wanted to wait.

The APMGR determines how many SC's are part of the system and what real device addresses are available to each SC. When the APMGR issues a CP ATTACH command to attach the SC to a user's virtual machine, the command must specify both a real device address and a virtual device address. The APMGR selects the real address from APCN64. This address is the address through which the real hardware communicates with the SC.

The virtual device address is the address through which the user's virtual machine communicates with the SC. When an I/O communication with the SC is requested through the virtual address, this request is automatically directed to the correct real address. APEX64 uses a virtual device address of 160 for SC communication. If a virtual device address 160 is currently in use by the virtual machine, it is detached before requesting the attach function from the APMGR.

3.2.4 Releasing and Detaching an SC

When the user finishes with the SC, it must be released and detached for other SC users to access. The APEX64 user performs these operations with a call to APRLSE (or APIBMR, discussed in Section 2.4.1.2). The SJE user simply issues the SJE DETACH command (not to be confused with CP DETACH). The release is accomplished by sending a release VMCF message to the APMGR and then executing a release channel program to the SC, which informs the HISP that the SC is now available to the next user. After release, the SC is detached from the user's virtual machine through a CP DETACH command.

3.2.5 SC Device Address 0

The first device address in the range configured for each SC is used by the APMGR for special processing. Attempts to attach the first address in the range for each AP to itself. The APMGR needs the first address in the range to perform AP Operator (APOPR) ABORT processing. If for some reason the APMGR can not attach the first device address in the range, it will attempt to attach any available device address in the range. If this happens, the APMGR will function correctly but the APOPR ABORT command will fail to abort users currently assigned to that AP.

3.2.6 Force-Release Processing

If a user's APEX64 or SJE job ends for any reason before APRLSE or APIBMR is called, the APMGR force releases the SC. This involves detaching the SC device address from the user's virtual machine and executing the release channel program to the SC. Note that when the APMGR has to perform force release processing, the accounting data, that is normally passed to the APMGR by APRLSE, is lost.

The APMGR will still write an accounting record for the offending user, but the account name field (set by the SJE set/account command), the SC/CPU time field, the SC disk I/O field, and the host I/O field will be missing.

3.3 ROLL-IN/ROLL-OUT (RIRO)

The roll-in/roll-out (RIRO) feature allows the SC to divide its processing time between multiple jobs according to their priority.

Under RIRO control, one job executes on the SC at a time. After a specified execution time interval, the job is rolled out to the disk and is placed on a queue of waiting jobs.

Up to 10 wait queues can be active at the same time, each with a different RIRO priority. After the current job is rolled out to the appropriate queue, the next job on the highest priority queue is rolled in and run for the time interval specified for that priority queue.

The wait queues can hold up to 31 jobs at a time. However, the number of jobs allowed in each queue can be limited by the MAXusers site parameter for that queue. Note that the total number users allowed in all active queues must not be greater than the number of real device addresses minus 1 configured for the SC. The queues are maintained in a round-robin fashion, with the most recently rolled-out job placed on the bottom of the queue. Depending on the priority scheme, which is system-dependent, an SC job assumes a default priority or a user-defined priority. The priority determines in which queue the job is placed. The queue determines how soon the job is rolled back in and how long the job runs before it is again rolled out. SJE jobs, both batch and interactive, and APEX64 jobs are treated equally within a queue.

The number and priority of queues, the SC CPU time interval and maximum number of jobs for each queue, and the default priority for different types of jobs are variable and are set at each installation. Refer to the FPS-164 System Manager's/Operator's Manual, listed in Table 1-1, for information on setting up RIRO queues.

NOTE

> In SJE, if the user is idle for more than three minutes when attached to the SC, the user is rolled out if someone is waiting to use the SC.

The following example shows how to initiate the FPS-164 Operator Program (APOPR):

```
APOPR
APOPR>
```

Once the "APOPR>" prompt appears, enter any APOPR command. (See the FPS-164 System Manager's/Operator's Manual.)

3.4 I/O TO FEC FILES

The user of an SC can gain access to FEC files directly from an executing SC program. Access can be accomplished by using the file name prefixes :HOST: and :HOSTCHAR:, which are placed directly in front of the file specifier as it appears in the calling program. The prefix :HOST: is used when the file specified is an SC-formatted binary file or a character file, while the prefix :HOSTCHAR: is used when the file specified is an FEC-formatted character file.

In general, the SC cannot gain access to an FEC magnetic tape device. However, under SJE the JDL commands PRESERVE and RESTORE do allow such access. Refer to Section 3.5 for FEC-specific examples of using PRESERVE and RESTORE.

No data format translation occurs for I/O performed to FEC-resident files that include the :HOST: prefix. These files must already be in the proper SC format for I/O to occur. Properly formatted SC files include those created by program development software (PDS) routines, by the SJE COPYOUT/BINARY command, or by an FEC program using the data conversion utilities.

Data format translations occur for I/O performed to the FEC terminal, and also for I/O performed to FEC character format files that include the :HOSTCHAR: prefix.

Use of the :HOSTCHAR: prefix is limited only to APFTN64 formatted I/O operations. APFTN64 programs cannot issue unformatted (binary) I/O statements to FEC-formatted files. The :HOST: prefix is used for I/O performed to all SC-formatted files on the FEC. Table 3-1 below presents these four combinations of file and data formats. How to gain access to the file from both the SC and the FEC is also shown.

Table 3-1 FEC File Prefixes

FILE TYPE	PREFIX	SC ACCESS	FEC ACCESS
FEC binary	none	not allowed	unformatted FEC FORTRAN
FEC character	:HOSTCHAR:	formatted APFTN64	formatted FEC FORTRAN
SC binary	:HOST:	unformatted APFTN64	FEC program with data conversion utilities
SC character	:HOST:	formatted APFTN64	FEC program with data conversion utilities

To gain access to an SC-formatted FEC file from a program executing on the SC, the user must specify the FEC file using the following format:

```
':HOST:filename filetype [filemode]'
```

If the user wants to gain access to an FEC character format file from a program executing on the SC, the user must specify the FEC file using the format:

```
':HOSTCHAR:filename filetype [filemode]'
```

Because the file specifier contains blanks between filename, filetype, and filemode, the entire string must be enclosed within single quotes. If the string is not quoted, the SC terminates the line at the first space encountered. The following lines of code demonstrate a sample write operation to an FEC file within an executing program:

```
      OPEN (6,FILE=':HOST:FILEA APTXT64 A')
      WRITE (6,10)A,B,C
10    FORMAT (F5.3,2X,F5.3,2X,F5.3)
```

3.5 USING PRESERVE AND RESTORE

Both PRESERVE and RESTORE are JDL commands that operate under SJE. The PRESERVE command allows the user to store a copy of one or more files under the SC file management system on an FEC disk or tape unit. Conversely, the RESTORE command restores back to the SC system any file or set of files previously saved by a PRESERVE command. Both binary and text files can be preserved under the same save file.

For information on the options available with PRESERVE and RESTORE, refer to Volume 1 of the FPS-164 Operating System Manual Set listed in Table 1-1.

To use the PRESERVE and RESTORE commands, the user must be attached to an SC and be operating under SJE.

The following examples illustrate two uses of PRESERVE and RESTORE.

In the first example, all SC files are saved on magnetic tape using PRESERVE and then restored using RESTORE. These commands assume the virtual address of the tape drive to be 281.

```
ACCESS (system_password):
PRESERVE /TAPE=:HOST:281/ID=JAN.01.1983/TREE/UNIVERSAL

ACCESS (system_password):
RESTORE /TAPE=:HOST:281/ID=JAN.01.1983
```

NOTE

It is more efficient to attach the tape drive to a different virtual channel than the one to which the SC is attached. APEX64 always attaches the SC to channel 1. If the SC and the tape drive must be on the same virtual channel, the channel should be defined as a block multiplexer to the virtual machine. The tape I/O during PRESERVE or RESTORE is overlapped with I/O to the SC. Using a block multiplexer or separate virtual channels saves APEX64 from the processing required to handle channel busy conditions.

In the above PRESERVE example, the ACCESS command establishes rights to the system directory. The JDL command ACCESS is necessary when attempting to preserve or restore files that require passwords or keys. The TREE parameter specifies the currently accessed directory (in this case the entire file system) of the SC FMS to be saved, including all subdirectories. PRESERVE assigns the ID "JAN.01.1983" to the saved files (the preserve file) on the tape unit at virtual address 281. The ID "JAN.01.1983" is kept in the preserve file header for identification in a RESTORE operation.

The /UNIVERSAL option supports copying files between different FEC's.

NOTE

The tape drive at the indicated virtual address should already be attached to the virtual machine before starting SJE. If the tape drive is not attached and ready when the PRESERVE command is issued, the FEC prompts the user to attach and ready the drive while program execution is delayed. Since the wait for an available tape drive is usually many times longer than the wait for an available SC, wait for the tape drive outside of SJE. The default tape density is 1600 BPI and the default sequence number is 1, the first savefile on the tape. Tapes used for the PRESERVE and RESTORE commands must be 9-track unlabled tapes.

In the above RESTORE example, the ACCESS command again establishes rights to the system directory. The RESTORE command interfaces with the tape drive at virtual address 281. For this example, the first file on this tape must have an ID of "JAN.01.1983" or RESTORE aborts. All SC file system files and directories contained in the preserve file will be created and restored if they no longer exist.

NOTE

If a user wishes to replace existing SC files with those found in a PRESERVE file, include the REPLACE option in the RESTORE command.

In the second example, several SC files are saved in a file on disk A. The following code demonstrates this process:

```
PRESERVE/FILE=':HOST:PRSERVE1 DAT A'/ID=MYFILES &
/LIST=':HOSTCHAR:PRSERVE1 LIS' FILEA,FILEB,FILEC,FILED

RESTORE/FILE=':HOST:PRSERVE1 DAT'/ID=MYFILES &
/REPLACE FILED
```

The file "PRSERVE1 DAT A" is a preserve file having "MYFILES" for an ID and containing the four SC files FILEA, FILEB, FILEC, and FILED. The FEC character file "PRSERVE1 LIS" created by the /LIST option is written on disk A in the user's default directory on the FEC. This file contains information for the PRESERVE command issued, such as the name of the PRESERVE file, the ID for the PRESERVE file, and information about each file saved.

The above RESTORE command replaces the SC file FILED with the FILED found in "PRESERVE1 DAT A".

NOTE

The ampersand symbol ("&") in an SJE command line is the continuation mark.

3.6 SJE DATA CONVERSION UTILITIES

This Section discusses the library of data and file conversion utilities available to SJE users as FEC-FORTRAN-callable subroutines and functions. These utilities perform data conversion between FEC and SC data formats and permit the creation and retrieval of APFTN64-compatible file records. The data formats are shown in Section 4.4.9.

SJE supports the transfer of both text and binary files. Text files are converted between FEC and SC formats automatically during their transfer. Binary files are transferred without any conversion.

Data conversion utilities are necessary only if the user wants to perform both of the following operations one after another.

- transfer (between the FEC and SC file systems) files containing binary (machine format) data written using FORTRAN unformatted WRITE statements
- read the file on the destination system using the matching FORTRAN unformatted READ statements

The user can combine program calls to these utilities with FEC I/O services to create FEC files that SJE can transfer as binary files. Once these files reside on the SC file system, APFTN64 programs can access them using standard FORTRAN 77 I/O. After retrieving a file from the SC file system, the user can use the data conversion utilities along with FEC I/O services to create FEC files that can be read by FEC FORTRAN programs.

File conversion routines convert FORTRAN unformatted file records between FEC and SC formats. The data conversion routines convert to and from the following types of data:

- FEC integers to and from SC integers
- FEC real (floating-point) numbers to and from SC real (floating-point) numbers
- FEC double-precision numbers to and from SC real (floating-point) numbers
- FEC logical numbers to and from SC logical numbers
- FEC characters to and from SC characters

A file to be transferred can contain various types of data within a single record. Because the FEC and SC have different file record formats, each record within the file requires a separate record conversion.

Figure 3-1 illustrates the use of the data and file conversion routines to transfer an FEC binary file to an SC binary file.

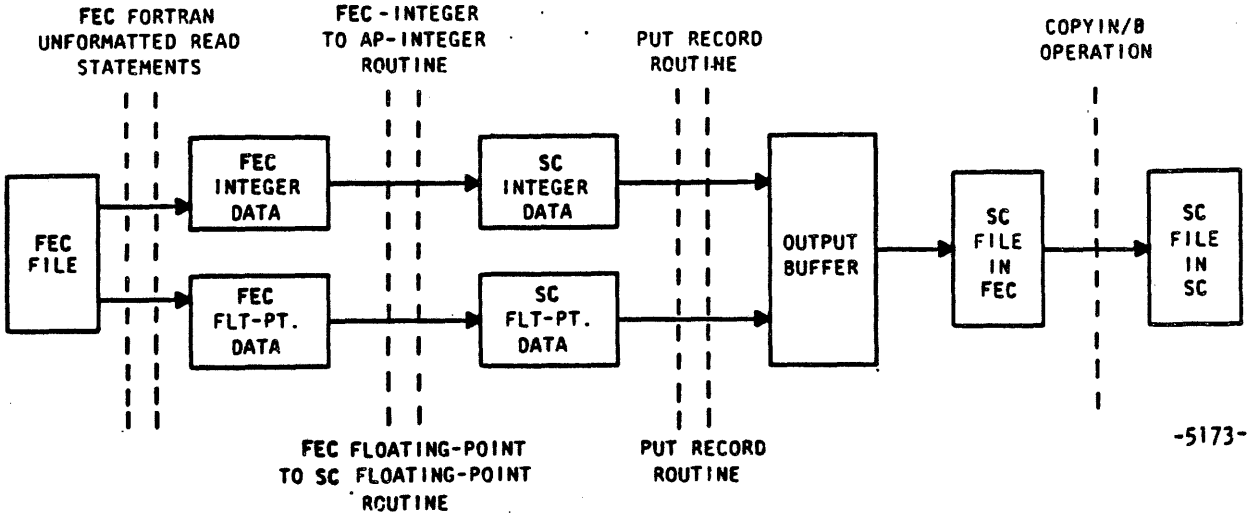


Figure 3-1 FEC File to SC File Conversion Steps

Assuming the existence of an FEC file written with FORTRAN unformatted WRITE statements, the user must write an FEC FORTRAN program that reads an FEC file record into memory using the appropriate FORTRAN unformatted READ operations. For each variable (or array of variables of identical type) now in memory, the program calls the appropriate data conversion routine to convert from FEC to SC data formats. The result of each data conversion must be stored contiguously in an intermediate buffer with all other converted data that constitutes the record. The program then calls the put record (PUTUR) routine to move the converted data from the intermediate buffer into a utility-managed buffer. During this movement, the PUTUR routine creates an SC-format record header and includes it in the buffer with the data. The PUTUR routine also takes care of writing the utility-managed buffer to an FEC file when it becomes full.

Once all of the data in the FEC-format file is processed in the above manner, the user has a new FEC file that contains SC-format data and file records. The user can then transfer the file to the SC file system using the SJE COPYIN/BINARY command. An APFTN64 program can read this file using FORTRAN unformatted READ statements identical to those used to read the original FEC file.

3.6.1 FEC Data File to SC Data File Conversion Procedure

The detailed processing steps in a program for converting an FEC FORTRAN data file into an equivalent FEC-resident APFTN64 data file are:

1. The user provides an 8208-byte data area (buffer) for use by the file conversion utilities.

2. The user then calls the initialize output buffer (INITOB) function, passing it the 8208-byte buffer, the name of the file to receive the APFTN64 records, and the file name length. The 8208-byte data area contains a 16-byte header and an 8192-byte (8K) buffer in which SC file records are built. The header contains information used by the file conversion utility during the file-building process. The INITOB function initializes the header and spreads zeros throughout the 8K buffer.

3. The user calls the data conversion utilities to convert all the data in the FEC FORTRAN record into SC format data. The following code demonstrates how to call conversion utilities:

```
                    .
                    .
                    .
READ (HSTFILE) REAL,INT1,INT2,DOUBLE
CALL FPHR2R (REAL,TEMP(1),1)
CALL FPHI2I (INT1,TEMP(3),1,4)
CALL FPHI2I (INT2,TEMP(5),1,4)
CALL FPHD2R (DOUBLE,TEMP(7),1)
                    .
                    .
                    .
```

   In this example, the program reads the data items REAL, INT1, INT2, and DOUBLE from one record in the file unit HSTFILE. Each type of data word is then converted by calling the appropriate conversion utility. As each data item is converted, it is placed in the array TEMP.

4. The user calls the put unformatted record (PUTUR) routine to create an SC FORTRAN record and buffer it into the 8K-word buffer. For example, the following statement places the array TEMP (created in step 3) into a temporary buffer called BUFF:

```
STATUS = PUTUR (BUFF, TEMP, 24)
```

   Because PUTUR places the four data items of TEMP into one record, the SC program can retrieve the converted data with a READ statement identical to one that reads the data from the FEC file.

When the 8K-word buffer becomes full, PUTUR automatically writes the entire 8K-word block into the output data file. The conversion of data in each record and the calling of PUTUR repeats until the entire output data file is complete.

5. The user calls the put endfile (PUTEF) routine. PUTEF writes an end-of-file record in the buffer, writes the buffer into the output data file, and closes the output file.

NOTE

The PUTUR routine continues to store the APFTN64 records in the 8K-word buffer until the buffer is full or the PUTEF routine is called. The routine then writes this 8K-word buffer to the output data file. The routine also saves enough information about a record that crosses the 8K-word block boundary to permit the record to continue in the next 8K-word buffer. Thus, the resulting FEC data file contains an image of a APFTN64 file that is blocked in 8K-byte records.

Figure 3-2 shows the logical flow for a program that performs the FEC to SC data file conversion.

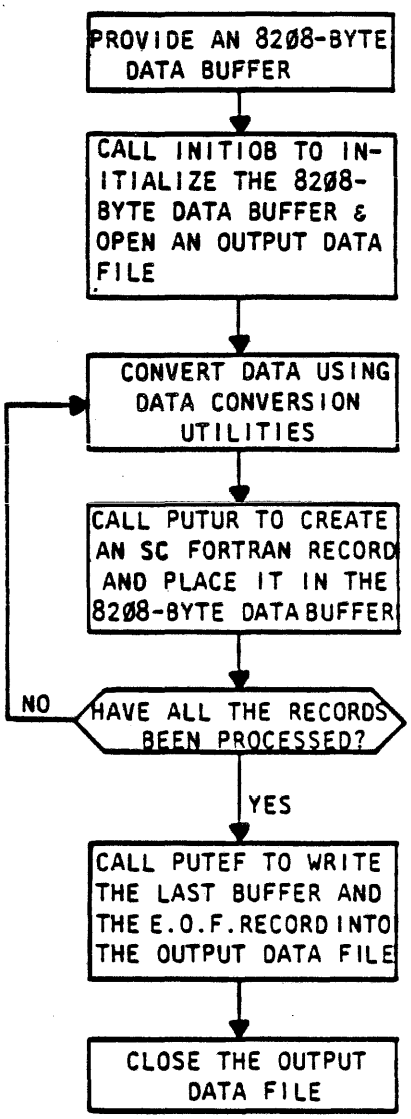


Figure 3-2 FEC Data File to SC Data File Conversion Logic

The following program is an example of the above steps:

```
C ***************************************************************************
C
C THIS PROGRAM CONVERTS 10 FEC INTEGERS TO 10 SC INTEGERS
C AND 5 FEC FLOATING-POINT (REAL) NUMBERS TO 5 SC FLOATING-POINT
C NUMBERS.  THEN THE PROGRAM WRITES THE SC NUMBERS INTO THE
C FEC FILE SJEDATA BIN IN THE FORM OF APFTN64 UNFORMATTED
C RECORDS.
C
C ***************************************************************************
C
C
C DEFINE THE 8208-BYTE OUTPUT BUFFER TO BE USED BY THE CONVERSION
C UTILITIES
C
      DOUBLE PRECISION SJEDAT(1024)
      DOUBLE PRECISION BUFFER(1026)
C
C SET UP TEMPORARY STORAGE FOR THE FEC-FORMAT AND SC-FORMAT NUMBERS
C AND OTHER PROGRAM VARIABLES
C
      INTEGER*4 HSTINT(10)
      REAL*4 HOSTRL(5)
      INTEGER*4 STATUS,NAMLEN
      INTEGER*4 FILNAM(3)
      DATA FILNAM/4HSJED,4HATA ,4HBIN /
C
C INITIALIZE THE UTILITY-MANAGED OUTPUT BUFFER AND OPEN THE OUTPUT
C FILE 'SJEDATA BIN'.
C
      NAMLEN = 12
      STATUS = INITOB (BUFFER,FILNAM,NAMLEN)
      IF (STATUS .NE. 0) GO TO "process error condition"
C
                              .
                              .
                              .
      Now read the 10 FEC integers into the array HOSTINT
      and the 5 FEC real numbers into the array HOSTRL.
                              .
                              .
                              .
C
C CONVERT THE 10 FEC INTEGERS TO SC INTEGERS USING THE FPHI2I
C CONVERSION ROUTINE
C
      CALL FPHI2I (HSTINT,SJEDAT,10,4)
C
C WRITE THE SC INTEGERS INTO THE UTILITY-MANAGED OUTPUT BUFFER USING
C THE PUTUR ROUTINE
C
      STATUS = PUTUR(SJEDAT,BUFFER,80)
C
```

```
C CONVERT THE 5 FEC REAL NUMBERS TO SC REAL NUMBERS USING THE
C FPHR2R ROUTINE
C
      CALL FPHR2R (HOSTRL, SJEDAT, 5)
C
C WRITE THE SC REAL NUMBERS INTO THE OUTPUT BUFFER USING PUTUR
C
      STATUS = PUTUR (SJEDAT, BUFFER, 40)
C
C WRITE THE END-OF-FILE RECORD TO THE OUTPUT BUFFER.
C
      STATUS = PUTEF (BUFFER)
C
      STOP
      END
```

3.6.2 APFTN64 File to FEC File Conversion Procedure

The conversion of an APFTN64 file into a FEC file is performed as the reverse of the process described in Section 3.6.1. The processing steps are:

1. The user provides an 8208-byte data area for use by the file conversion utilities.

2. The user then calls the initialize input buffer (INITIB) function, passing it the 8208-byte data area, the name of the file to receive the APFTN64 records, and the file name length. The 8208-byte data area contains a 16-byte header and an 8192-byte (8K-byte) buffer into which SC file blocks are read. The header contains information used by the file conversion utilities during the APFTN64 record-reading process. The INITIB function initializes the header and spreads zeros throughout the 8K-byte buffer.

3. The user calls the get unformatted record (GETUR) function to fill the 8K-byte buffer with APFTN64 records (if necessary) and get a record from the 8K-byte buffer. The record is placed into an area in FEC memory specified in the GETUR call. For example, the following statement retrieves the data record created in Section 3.6.1 with PUTUR.

   ```
   STATUS = GETUR (BUFF, TEMP, 24)
   ```

   This statement places a record from BUFF into the FEC array TEMP.

4. The user calls the data conversion utilities to convert the SC record obtained by GETUR into FEC data. The following code reconverts the data in the array TEMP to a FEC-compatible form:

```
CALL FPR2HR (TEMP(1),REAL,1)
CALL FPI2HI (TEMP(3),INT1,1,4)
CALL FPI2HI (TEMP(5),INT2,1,4)
CALL FPR2HD (TEMP(7),DOUBLE,1)
```

Subsequent APFTN64 records are obtained and converted by calling GETUR and the conversion routines until either the end-of-data indicator is encountered or the user decides to stop.

5. The user closes the input file.

NOTE

The GETUR routine performs all read operations on the input SC file. A read transfers 8K-bytes from the SC file to the 8K-bytes buffer if the buffer is empty. This routine also moves the contents of the next available APFTN64 record in the 8K-bytes buffer to the area specified by the routine call. The APFTN64 record header information determines the length of the move. Subsequent calls to GETUR retrieve additional records from the 8K-bytes buffer. When the buffer becomes emptied of records, the GETUR routine performs an additional read of the SC file to refill the buffer.

Figure 3-3 shows the logical flow for a program that performs the SC to FEC data file conversion.

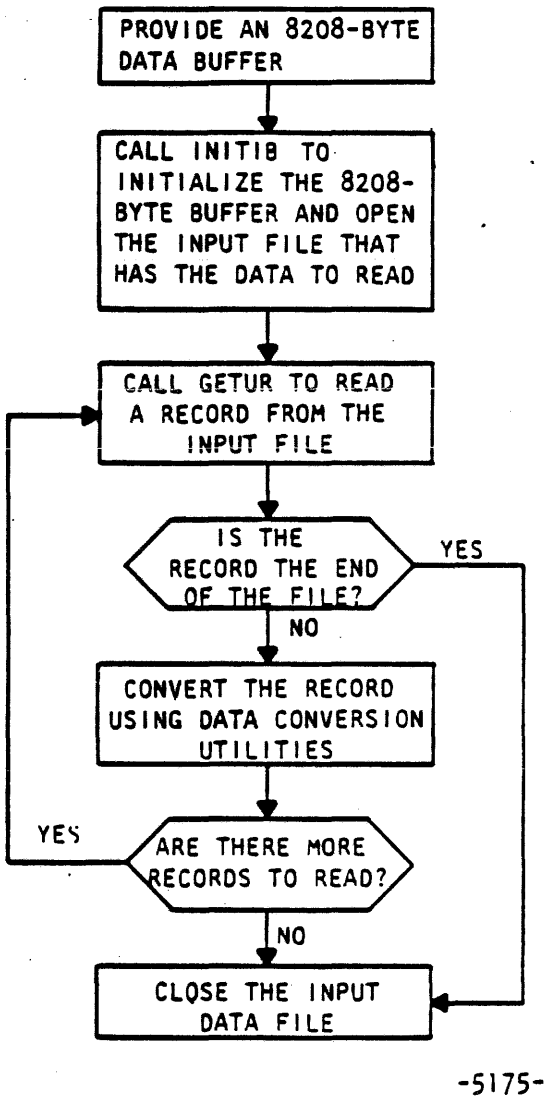


Figure 3-3 APFTN64 File to FEC File Conversion Logic

The following program is an example of the above steps:

```
C ***************************************************************************
C
C THIS PROGRAM READS TWO APFTN64 UNFORMATTED RECORDS FROM THE FEC
C FILE 'SJEDATA BIN'.  THE FIRST RECORD CONTAINS 10 SC INTEGERS
C AND THE SECOND RECORD CONTAINS 5 SC FLOATING-POINT (REAL) NUMBERS.
C THE PROGRAM CONVERTS THE NUMBERS FROM SC TO FEC DATA FORMATS.
C
C ***************************************************************************
C
C DEFINE THE 8208-BYTE INPUT RECORD BUFFER TO BE USED BY THE CONVERSION
C UTILITIES
C
      DOUBLE PRECISION BUFFER(1026)
      DOUBLE PRECISION SJEDAT(1024)
C
C SET UP TEMPORARY STORAGE FOR THE FEC-FORMAT AND SC-FORMAT NUMBERS
C AND OTHER PROGRAM VARIABLES
C
      INTEGER*4 HSTINT(10)
      REAL*4 HOSTRL(5)
      INTEGER*4 STATUS,RECLEN
      INTEGER*4 EOF
      INTEGER*4 NAMLEN
      INTEGER*4 FILNAM(3)
      DATA FILNAM/4HSJED,4HATA ,4HBIN /
      DATA EOF/-1/
C
C INITIALIZE THE UTILITY-MANAGED INPUT BUFFER AND OPEN THE INPUT FILE.
C
      NAMLEN = 12
      STATUS = INITIB (BUFFER,FILNAM,NAMLEN)
      IF (STATUS .NE. 0) GO TO "process end of file"
C
C READ THE FIRST APFTN64 UNFORMATTED RECORD INTO THE ARRAY "SJEDAT"
C
      STATUS = GETUR (BUFFER, SJEDAT,RECLEN)
      IF (STATUS .EQ. EOF) GO TO "process end of file"
C
C
C CONVERT THE SC FORMAT INTEGERS TO FEC FORMAT INTEGERS USING FPI2HI
C
      CALL FPI2HI (SJEDAT, HSTINT, 10, 4)
C
C READ THE NEXT APFTN64 UNFORMATTED RECORD
C
      STATUS = GETUR (BUFFER, SJEDAT,RECLEN)
      IF (STATUS .EQ. EOF) GO TO "process end of file"
C
C
C
C CONVERT THE SC REAL NUMBERS TO FEC REAL NUMBERS USING FPR2HR
C
```

```
      CALL FPR2HR (SJEDAT, HOSTRL, 5)
                              .
                              .
                              .
         The array HOSTINT now contains the 10 FEC-format
         integers and the array HOSTRL contains the 5
         FEC-format real numbers.
                              .
                              .
                              .
      STOP
      END
```

3.7 LINKING TO FEC-SPECIFIC LIBRARY ROUTINES

To link to routines located in an FEC-specific library such as the UTIL64.OLB library, the user must perform the following general procedures:

1. Build the main FORTRAN program.
2. Build any required FORTRAN subroutines.
3. Compile the main program.
4. Declare global libraries.
5. Compile the subroutines.
6. Load the main program and subroutines.

The following example illustrates the steps involved in linking the main FORTRAN (MAIN) to one FORTRAN subroutine (SUB1) and the UTIL64 library:

```
GLOBAL TXTLIB VFORTLIB CMSLIB APEX64 APDBG64 UTIL64
FORTVS MAIN
FORTVS SUB1
LOAD MAIN SUB1
START
```

CHAPTER 4

IBM/CMS FEC-SPECIFIC HARDWARE OF THE FEC/SC SYSTEM

4.1 INTRODUCTION

Chapter 4 provides IBM/CMS-specific hardware of the FEC/SC system. This chapter contains three main sections. The first section discusses FEC/SC communication. The second section provides a functional description of the SC's FEC interface. The third section presents hardware information that is unique to the FEC/SC system.

4.2 HARDWARE FOR FEC/SC COMMUNICATION

The Host Interface Support Processor (HISP) manages the details of data and command transfers between the FEC and the SC. It relieves the FEC and SC of much of the computational overhead involved with FEC/SC communication.

The following list includes the hardware elements involved with the operation of the HISP:

- the IBM CPU
- the SC CPU
- the IBM channel
- the SC's FEC interface, containing the HISP, the Formatter, and the Host Adapter

IBM systems have channels that perform I/O for the IBM CPU's. A channel only interrupts the IBM CPU when it must signal completion of a channel program or alert the FEC to an exceptional condition on the channel.

The IBM channel follows its usual protocol when communicating with the SC. It does not know that the SC is actually a separate computer.

The HISP plays the same kind of role for the SC as the channel plays for the FEC. It manages I/O transactions for the SC, interrupting the SC only when directed by APEX64 software on the FEC.

The IBM CPU, the IBM channel, the HISP, and the SC CPU thus divide the work of communication in a way that combines efficient data transfer with minimal CPU overhead. Sections 4.2.1 through 4.2.4 expand the discussion in this section by listing and describing the tasks that each interface element performs.

4.2.1 The Role of the IBM CPU

The IBM CPU runs a mainline program that contains calls to APEX64 routines. The APEX64 routines establish communication between the FEC and the SC. The mainline program also contains calls to subroutines that run in the SC. The SC object code for the subroutines is stored in data structures in FEC main storage or in the FEC file system.

When the program calls an SC subroutine, the interface between the host and SC (HASI) is invoked. The HASI calls certain APEX64 routines that transfer the requested SC subroutine to the SC and then cause the SC to execute it. Specifically, APEX64 routines do the following:

- build data descriptor blocks (DDB's) to send to the SC's FEC interface
- write IBM channel programs into FEC main storage
- direct the IBM channel to start executing the channel programs

The IBM CPU therefore limits its I/O-related activities to executing subroutines that program the IBM channel.

SJE is responsible for making calls to APEX64 to perform the job.

4.2.2 The Role of the Channel

The channel executes a channel program by sending commands and data to the HISP. It sends commands and data in the following order:

1. channel commands that prepare the interface to receive data from or transmit data to the channel
2. DDB's that describe the form of the upcoming data transfer
3. data to be transferred to the SC

A DDB consists of a series of data descriptors (DD's). Each DD specifies a single contiguous data array, its direction of transfer, its format, and its destination (PS or MD).

The channel also generates interrupts upon request from the SC. The SC notifies the HISP, and the HISP signals the channel to interrupt the FEC.

4.2.3 The Role of the HISP

The HISP does the following:

- It responds whenever the channel sends a channel command word (CCW) to the SC.
- It manages the SC end of the data and command transfer protocol on the channel.
- It accepts and stores DDB's sent by the channel.
- It interprets and executes the DDB's.

The data descriptors contained in the DDB's direct the HISP to do the following:

- transfer data between the channel and SC main memory
- communicate with the SC CPU via data transfers to the Virtual Front Panel (VFP) of the SC

In addition, the HISP generates interrupts as directed by the SC and translates them into channel status bytes. It also generates channel interrupts to notify the FEC of any abnormal conditions that occur during a data or command transfer. When an interrupt is generated, it is accompanied by status information that identifies it as a normal or abnormal channel program completion. An abnormal completion can refer to an error condition detected by the HISP (such as the SC physically halting) or an exceptional condition detected by the SUM (such as a floating-point underflow or the execution of a FORTRAN PAUSE statement by the user's SC program).

4.2.4 The Role of the SC CPU

The SC CPU runs the Single User Monitor (SUM) program. The SUM program does the following:

- It informs the HISP of the specific regions of SC memory that the SUM has allocated to the user.
- It directs the HISP to generate the appropriate FEC I/O interrupt when the user's program wishes to return control to the FEC.

4.2.5 The Role of the Formatter

For jobs done in APEX64, the Formatter converts data between the FEC and the SC representations of fixed-point, floating-point, and logical data. The Formatter simply acts as a path for data during SJE jobs. The Data Conversion Utilities (refer to Chapter 3) perform data conversions during SJE jobs.

4.3 THE SC'S FEC INTERFACE INTERNAL STRUCTURE

The three main hardware elements of the SC's FEC interface are the Host Adapter, the Formatter, and the HISP. Figure 4-1 is a block diagram of the SC's FEC interface used with IBM mainframes. Sections 4.3.1 through 4.3.3 describe these hardware elements of the SC's FEC interface.

4.3.1 Host Adapter

The Host Adapter is located in the SC I/O chassis. It converts signals on the interconnect cable from the channel into instructions and data that it sends to the HISP and the Formatter. The Host Adapter also connects to the SC diagnostic microprocessor (DMP). This connection allows the DMP to test most interface functions by simulating the actions of the FEC.

4.3.2 FEC Interface Support Processor

The HISP controls all interface operations. The major elements of the HISP are the microcode control store and control unit, the arithmetic and logic unit, the SC I/O bus interface, and the data file. The HISP communicates with the SC using the I/O bus interface. The data file holds DDB's passed from the FEC and parameters passed from the SC memory management monitor. The HISP passes control signals to the other sections of the interface over the operand bus.

The HISP maintains a 64-bit data element called the FEC command and status register (HCSR) for each user task. The HCSR stores commands and contains condition bits that indicate the status of the interface. The channel program can read and write the HCSR to determine the status of the HISP and send it commands.

4.3.3 Formatter

The Formatter performs format conversions of FEC and SC data words. It has an interface to the SC I/O bus, an input buffer, an output buffer, logic that performs format conversions, and a control section that interacts with both the Host Adapter and the HISP. The Host Adapter,

the SC I/O bus, and the HISP can be sources or destinations for the Formatter. Data normally travels between the Host Adapter and the SC I/O bus. The connection to the HISP is mainly for Formatter control purposes. Figure 4-1 shows an interface block diagram.

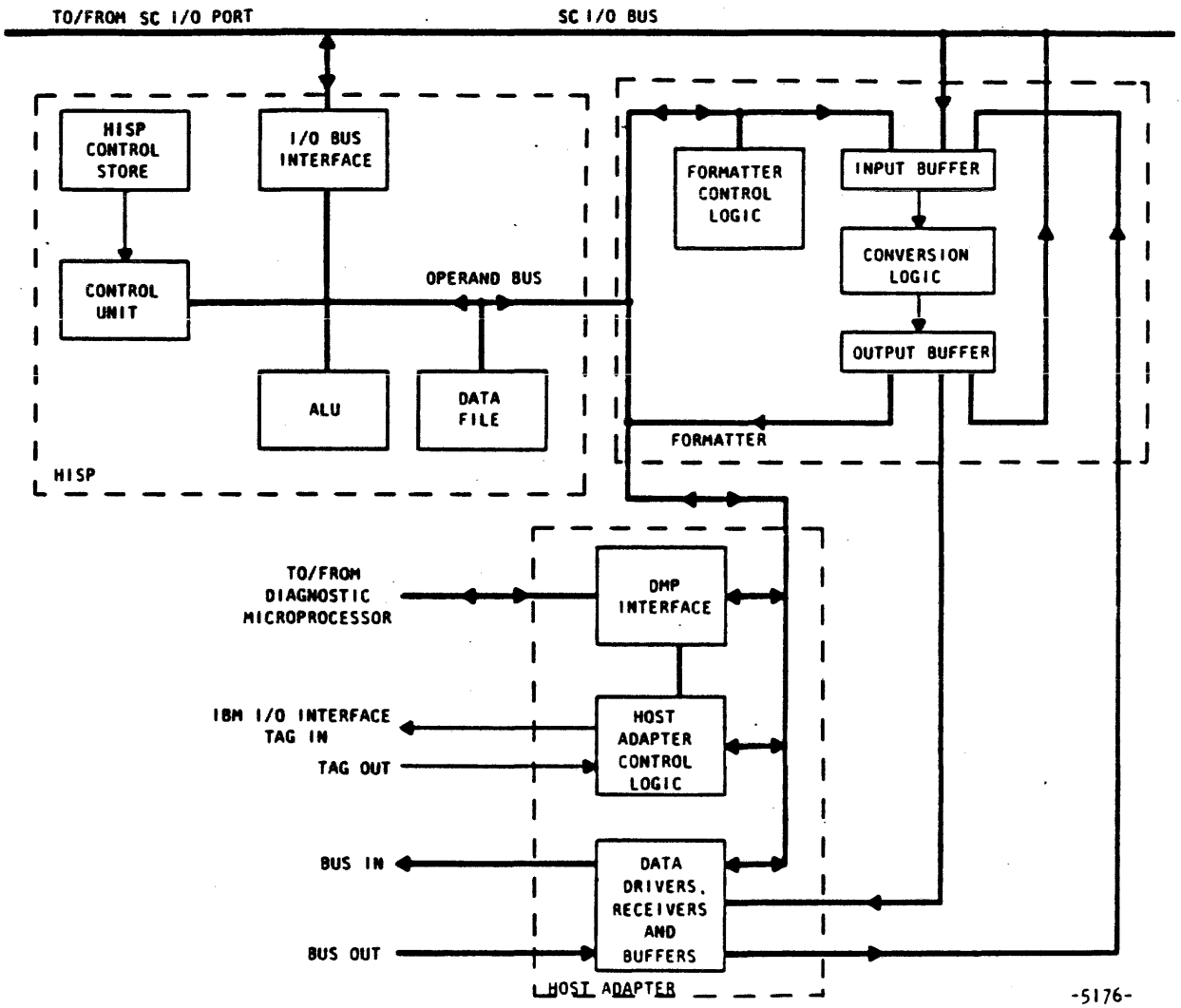


Figure 4-1 Interface Hardware Block Diagram

4.4 IBM/CMS FEC-SPECIFIC HARDWARE OF THE FEC/SC SYSTEM

Section 4.4 presents the hardware information a user needs to run an SC from an IBM FEC. In particular, the chapter discusses characteristics of the SC's FEC interface hardware that can influence the way a programmer codes software for the FEC/SC system. The text assumes that the user is experienced with the CP and CMS systems and is already familiar with the operation of the SC. When in doubt about proper use of the SC hardware, consult the publications listed in Table 1-1.

4.4.1 Address Recognition

The FEC interface can recognize a range of device addresses.

At installation time, FPS Customer Service engineers adjust the interface address recognition hardware to recognize one of the following address combinations:

- one address (address mode 0)
- one range of eight consecutive addresses beginning at an address that is a multiple of eight (address mode 1)
- one range of 16 consecutive addresses beginning at an address that is a multiple of 16 (address mode 2)
- two ranges of 16 consecutive addresses, each range beginning at an address that is a multiple of 16 (address mode 3)

Note that address mode 0 (one address) is not supported by the F00 release APMGR. The interface must be configured for at least 8 addressed for the software to function correctly.

The FPS-164 Scientific Computer Manager Program (APMGR) assigns each user process (virtual machine) running in the SC a unique device address in the SC. The APEX64 software running in the FEC manages access to the SC The HISP remembers which device address has been given control of the interface.

An interface that recognizes several addresses sends a busy or command retry signal to the channel if it is busy with a process running through one address when the channel initiates an I/O request at a second address.

4.4.2 Channel Interface Protocol

The FEC interface connects to an IBM block multiplexer channel. The interface can use any combination of the extended bus, high-speed transfer, and data-streaming options available with these channels. Manual switches inside the interface enable these features. The interface reports the settings of the switches in its response to a sense command from the channel.

4.4.3 Channel Commands

Table 4-1 lists and describes the channel commands to which the FEC interface responds.

Table 4-1 Channel Commands

OPERATION	HEX CODE	COMMAND DESCRIPTION
write HCSR	09	Directs the interface to store the next eight data bytes in the host command and status register (HCSR) of the HISP, and to execute the resulting command.
read HCSR	0A	Directs the interface to send the contents of the HCSR to the FEC.
write	01	Directs the interface to receive data from the FEC. The FEC must send either a WRITEP or a READTP command to the HCSR before sending this command. The interface expects a DDB to follow a write command. If the HCSR contains a WRITEP command, the interface expects data words to follow the DDB. If the HCSR contains a READTP command, the interface disconnects from the channel after receiving the DDB.
read	02	Directs the interface to send data to the FEC. A WRITEP HCSR command and a WRITE command specifying the DDB must precede this command.
no operation	03	Directs the interface to disconnect from the channel.
set unit check mask	11	Transfers the four-byte unit check mask from the FEC to the interface.
sense unit check mask	14	Transfers the four-byte unit check mask from the interface to the FEC.
dump	24	Transfers the contents of all alterable storage in the HISP to the FEC. Does not alter the contents.
assign with wait	07	Assigns the SC to the unit address at which the command arrives; delays channel program completion until the SC becomes available.

Table 4-1 Channel Commands (cont.)

OPERATION	HEX CODE	COMMAND DESCRIPTION
assign immediate	0F	Assigns the SC to the unit address at which it arrives; produces a "unit exception" interrupt if the SC is already assigned to another unit address. (For information on the unit exception interrupt, refer to Section 4.4.7.)
release	0B	Releases the SC for assignment to another unit address.
release waiting address	63	Removes a waiting unit address from the assign with wait command queue.
sense	04	Directs the interface to send status information to the FEC. The sense bytes contain error condition codes and other condition codes that describe the status of the interface and the SC. The IBM device protocol specifies the condition codes that correspond to the first six bits of the first sense byte (byte 0). Figure 4-2 describes the sense bytes format. Table 4-2 lists and describes the condition codes.
sense I/O type	E4	Returns seven bytes identifying the type of the SC, the HISP firmware revision level, and any optional features installed in the SC. (For further information, refer to Section 4.4.5.)

4.4.4 Sense Bytes

Figure 4-2 illustrates the format of the sense bytes, which provide various condition codes used in representing the status of the SC and the interface. The specific condition codes are described in Table 4-2 through Table 4-5.

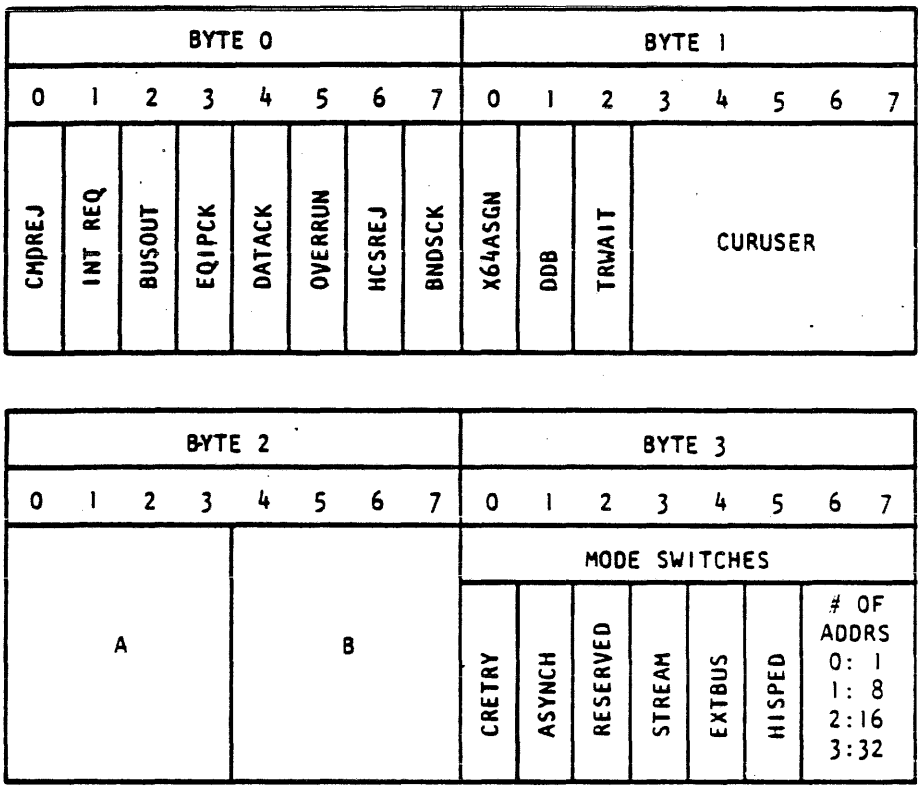


Figure 4-2 Interface Sense Byte Format

Table 4-2 Interface Sense Byte 0 Fields

BIT	MNEMONIC	MEANING
0	CMDREJ	Command reject. The interface has received a command it is not designed to execute, or an illegal sequence of commands.
1	INT REQ	Intervention required. Not used.
2	BUSOUT	Bus output check. Parity error in a data byte, command word, or on the mark-out tag lines.

Table 4-2 Interface Sense Byte 0 Fields (cont.)

BIT	MNEMONIC	MEANING
3	EQIPCK	Equipment check. Equipment malfunction; detail bits in the rest of the sense bytes indicate the type of malfunction.
4	DATACK	Data check. Error not involving interface parity.
5	OVERRUN	Overrun. Not used.
6	HCSREJ	HCSR rejected. Indicates HISPCD, HISPCR, APCRJ, and TIMEOUT errors in HCSR(1), as well as command sequence errors detected after one or more bytes have been transferred.
7	BNDSCK	Bounds check. Indicates memory protect violations during the processing of transfer packets, as well as oversize DDB's.

Table 4-3 Interface Sense Byte 1 Fields

BIT	MNEMONIC	MEANING
0	X64ASN	SC assigned. Indicates that the user identified by the CURUSER field has the SC assigned.
1	DDB	Data descriptor block. Indicates that a DDB is being processed.
2	TRWAIT	Indicates that at least one TRWAIT condition is outstanding.
3 -7	CURUSER	User ID of currently "active" physical I/O address (not necessarily the same as the address which is executing this sense command).

Table 4-4 Interface Sense Byte 2 Fields (Image of the Unit Address Switches on the Adapter)

BIT	MNEMONIC	MEANING
0-3	A	Setting of address switch A.
4-7	B	Setting of address switch B.

Table 4-5 Interface Sense Byte 3 Fields (Image of the Eight Mode Switches on the Adapter)

BIT	MNEMONIC	MEANING
0	CRETRY	Report HISP busy as command retry (or control unit busy).
1	ASYNCH	Enable asynchronous posting of TASKDN and TASKIN (or wait for next CCW). This option must always be disabled. Allowing the HISP to post interrupts asynchronously causes AP jobs to hang intermitantly because the IBM hardware or software sometimes looses the interrupt.
2		<reserved>
3	STREAM	Enable use of streaming handshake feature if supported by channel.
4	EXTBUS	Enable use of extended bus feature if supported by channel.
5	HISPED	Enable use of high speed transfer feature if supported by channel.
6&7	# of addrs	Number of unit addresses recognized by the Adapter.

NOTE

Regarding the remaining sense byte fields (4-15), the sense command transmits a reserved byte (byte 4), the last SC memory address read or written during DDB processing (bytes 5-7), and the HCSR (bytes 8-15).

4.4.5 Sense I/O Type Command

The Sense I/O Type Command (the hexadecimal E4 shown in Table 4-1) returns seven bytes identifying the type and model of the device and of its control unit. Figure 4-3 shows the data format defined by IBM convention.

BYTE	0	1 2	3	4 5	6
CONTENT	ALWAYS FF (HEX)	CONTROL UNIT TYPE	CONTROL UNIT MODEL	DEVICE TYPE	DEVICE MODEL

-5178-

Figure 4-3 Bytes Returned by Sense I/O Type Command

For the SC, the hexadecimal values of the seven bytes returned are as follows:

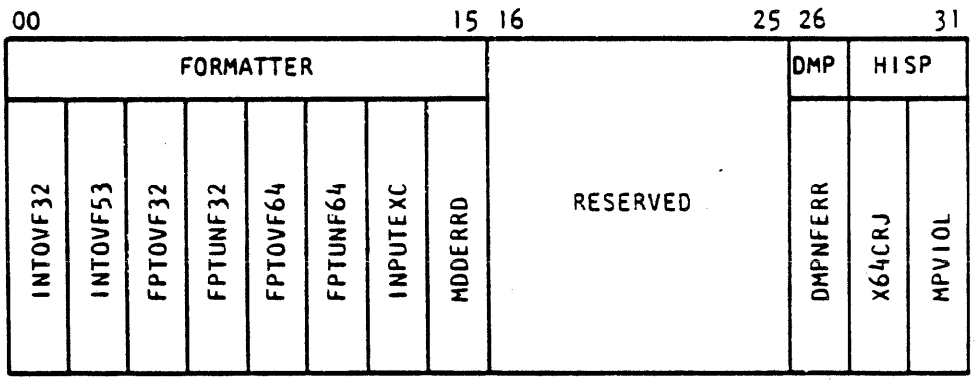
FF FO 64 rr F1 64 mm

Two of the seven bytes are parameters that can vary from SC to SC, as follows:

rr = HISP firmware revision level
mm = optional features installed in the SC
(for a standard product, mm = 0)

4.4.6 Unit Check Mask

Figure 4-4 describes the unit check mask.



-5179-

Figure 4-4 Unit Check Mask Bit Fields

A two-bit field is assigned to each condition. The following list defines the coding:

0 no unit check for this condition

1 unit check upon completion of DDB

2-3 immediate unit check, abort DDB

4.4.7 Channel Interrupts

The IBM channel can interrupt the IBM CPU to send it information on the status of the channel. It presents the CPU with a 64-bit channel status word that indicates the reason for the interrupt and the status of the interrupting device.

The SC supplies status information to the channel in the unit status field of the channel status word. Table 4-6 lists and explains the meaning of each bit in the unit status field.

Table 4-6 Unit Status Field Bits

BIT	DESIGNATION	MEANING
0	Attention	The user's SC subroutine has completed execution and is returning to the FEC program.
1	Status Modifier	Modifies the meaning of other status bits: with unit check and channel end bits set it invites the channel to retry a command as explained in Section 4.4.8; with device end and without unit check bits set, it tells the channel processor to skip the next CCW in the channel program; with busy bit set, it indicates that the interface is busy with a process that originated at another channel address.
2	Control Unit End	The interface can now respond to another channel command.
3	Busy	The interface is busy and cannot respond to a new channel command; the interface sets this bit in response to an initial selection sequence from the channel that occurs when the interface is busy doing a data transfer for another channel program. The HISP does not interrupt the channel to report this condition: it merely sets this bit in the channel status word when the channel attempts communication with the interface.

Table 4-6 Unit Status Field Bits (cont.)

BIT	DESIGNATION	MEANING
4	Channel End	The interface is finished with the data transfer portion of an operation, but not yet ready to execute another channel command.
5	Device End	The interface is finished with all processing required by the last command it received, and is ready to execute another command.
6	Unit Check	The interface has detected an error condition; the channel terminates any data transfer in progress and interrupts the IBM CPU. In general, the IBM CPU then executes a sense operation to determine the nature of the error.
7	Unit Exception	The interface has detected one of the four following conditions, which probably constitutes an error. • The SC CPU has interrupted the HISP with a TASKIN interrupt. • An assign immediate channel program was executed while the SC was already assigned to another user. • A HALT DEVICE instruction was executed while the user was in the assign-wait queue. During this action, the user also removed himself from the queue. • A Release Waiting Address channel program was executed through userid 0. Userid 0 is represented by the first address in the range of device addresses recognized by the SC. If the Release Waiting Address executes through other than userid 0, a unit check occurs. Upon recognizing one of the above four conditions, the channel terminates the current operation and interrupts the IBM CPU.

4.4.8 Interface Busy Conditions

The interface is busy when it is involved in either of the following operations:

- presenting the SC with an interrupt to which it has not yet responded
- transferring data

While it is busy, the interface can send a signal that invites the channel processor to retry a command after the busy condition ends. The interface signals command retry by setting the channel end, unit check, and status modifier bits in the unit status byte and notes internally that command retry has been indicated to the user. When the interface finishes what it is doing, it scans all the addresses to see if it has signaled command retry to any of them. If it finds an address that has received a command retry signal it sends a device end signal to the channel at that address to signal that it is ready to have the command repeated.

4.4.9 Format Conversion

The Formatter converts data words to and from FEC and SC data formats as they pass through the interface. It can convert the formats of words passing in either direction. The HISP loads format conversion instructions into the Formatter control registers to tell the Formatter hardware which conversions are required.

The Formatter in the IBM interface performs nine types of format conversion. It also flags underflow and overflow errors. This Section presents descriptions of these format conversions. The numbers above the word diagrams in the descriptions refer to bit positions in the words used by the SC and the IBM FEC.

Figure 4-5 shows the conversion between an IBM integer*4 and an SC integer (format type 0).

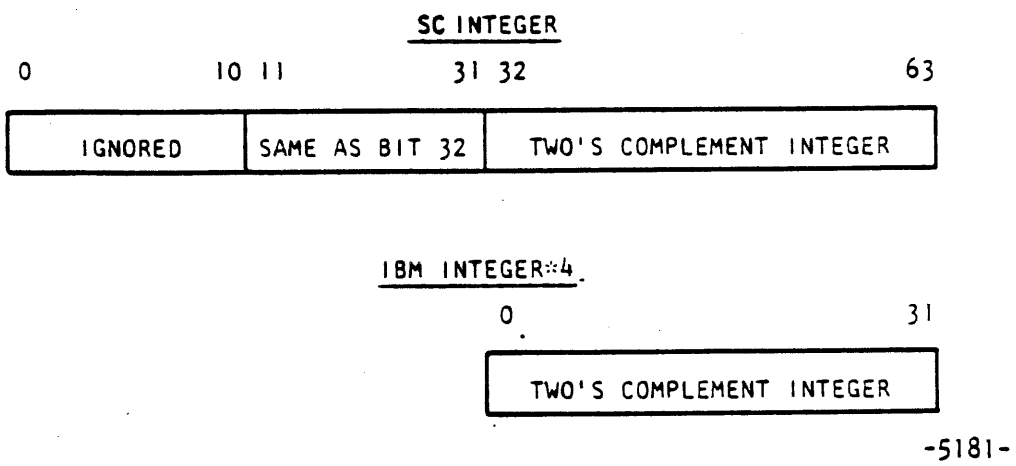


Figure 4-5 IBM Integer*4 to and From SC Integer*4

Figure 4-6 shows the conversion between an IBM integer*4 and an SC long integer (format type 1).

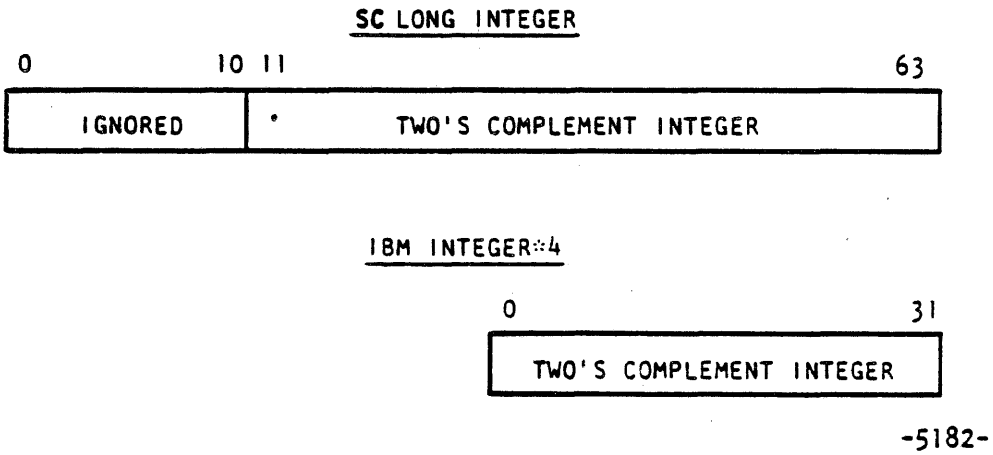


Figure 4-6 IBM Integer*4 to and From SC Long Integer

Figure 4-7 shows the conversion between an IBM integer*8 and an SC long integer (format type 2).

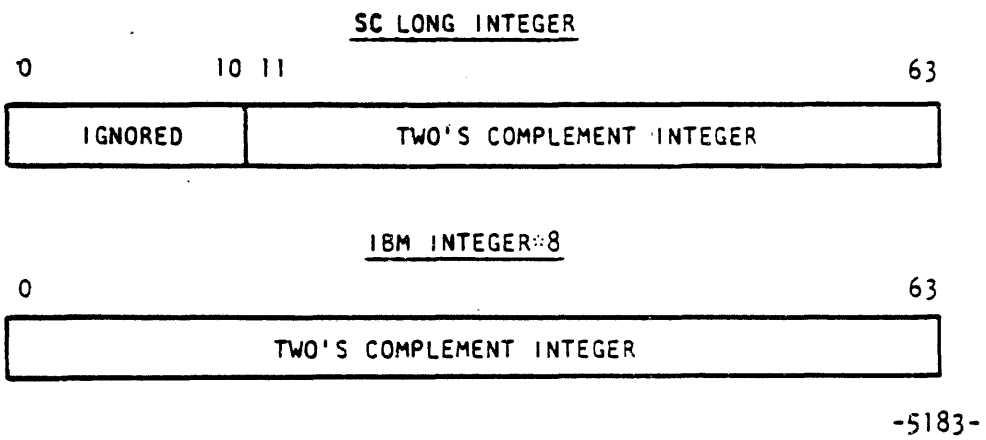


Figure 4-7 IBM Integer*8 to and From SC Long Integer

Figure 4-8 shows the conversion between an IBM integer*4 and an SC halfword packed integer (format type 3).

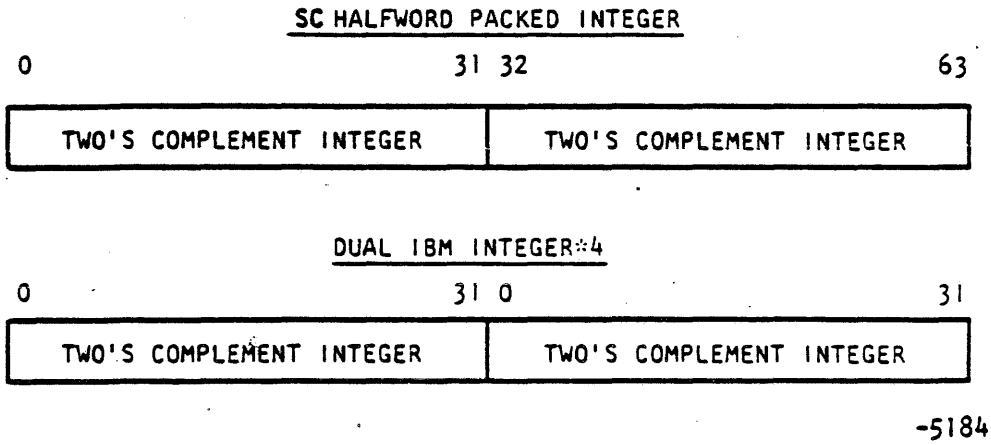


Figure 4-8 IBM Integer*4 to and From SC Integer Halfword Packed Integer

Figure 4-9 shows the conversion between an IBM load module and an SC word type (format type 4).

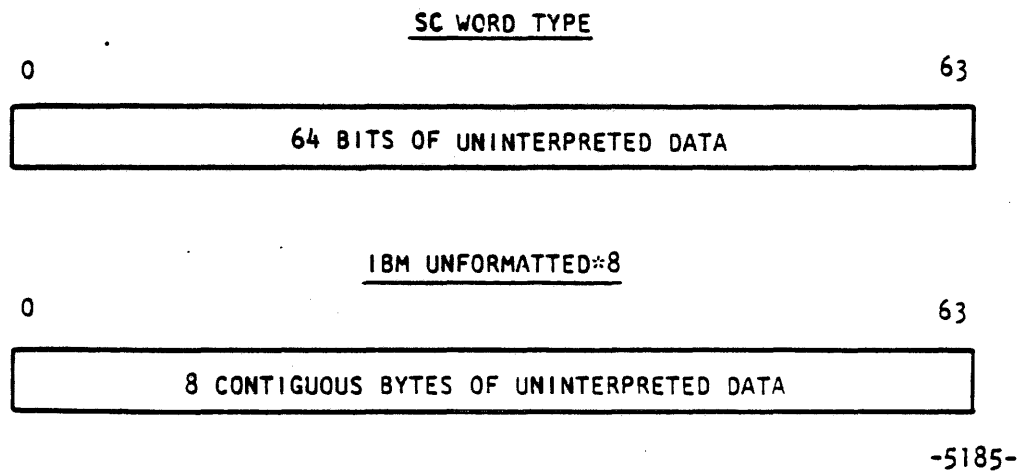
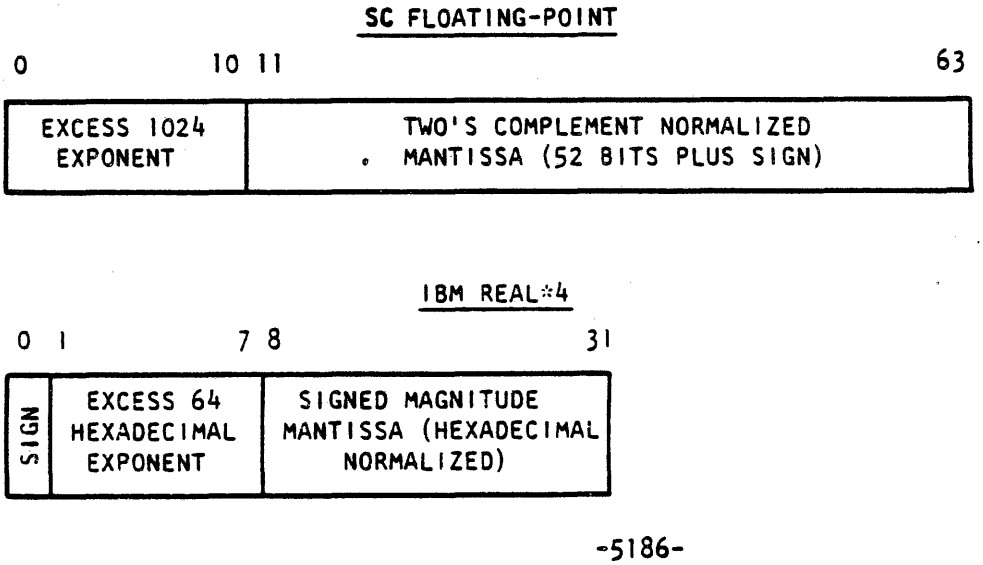


Figure 4-9 IBM Unformatted*8 to and From SC Word Type

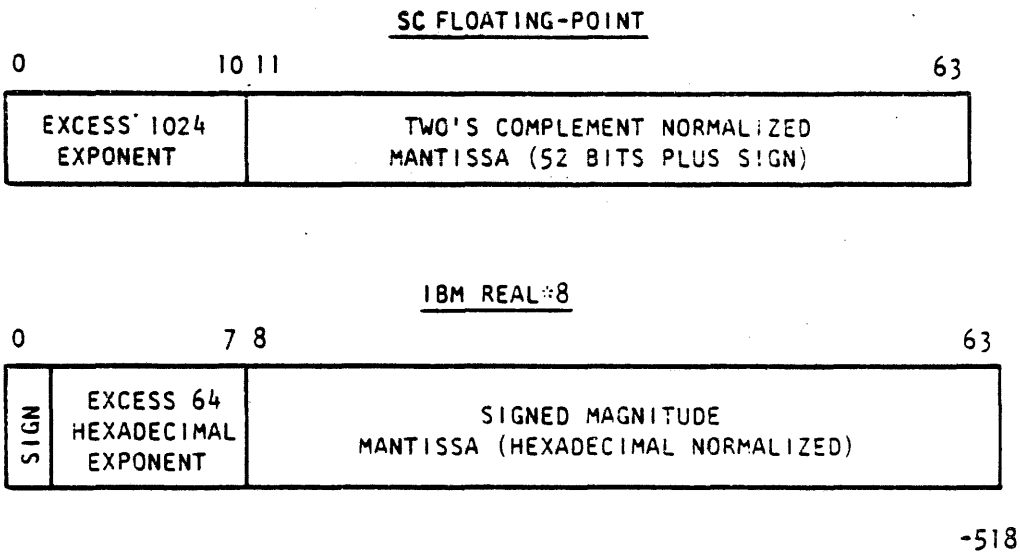
Figure 4-10 shows the conversion between an IBM Real*4 and an SC floating-point number (format type 5).



-5186-

Figure 4-10 IBM Real*4 to and From SC Floating-point Number

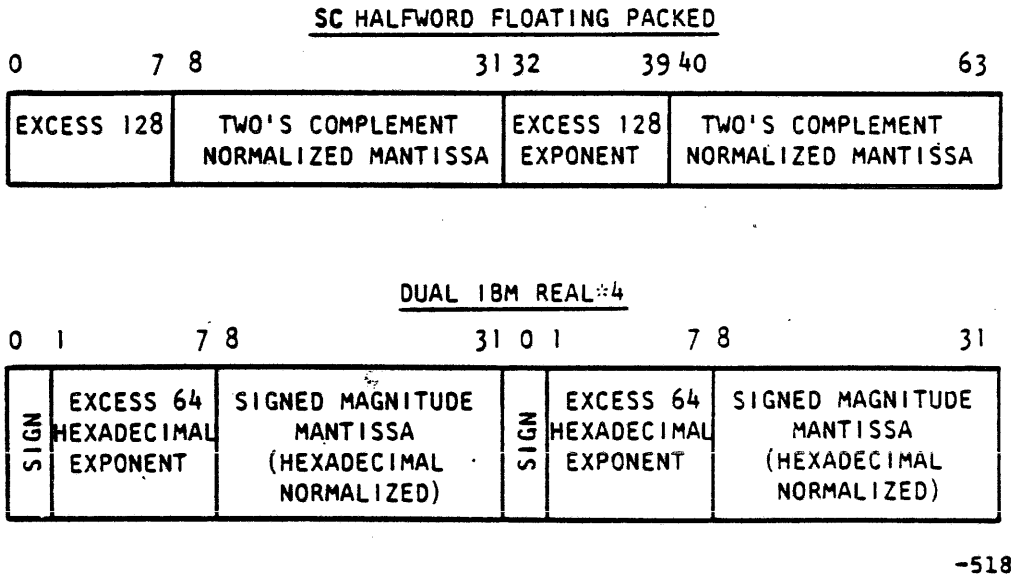
Figure 4-11 shows the conversion between an IBM Real*8 and an SC floating-point number (format type 6).



-5187-

Figure 4-11 IBM Real*8 to and From SC Floating-point Number

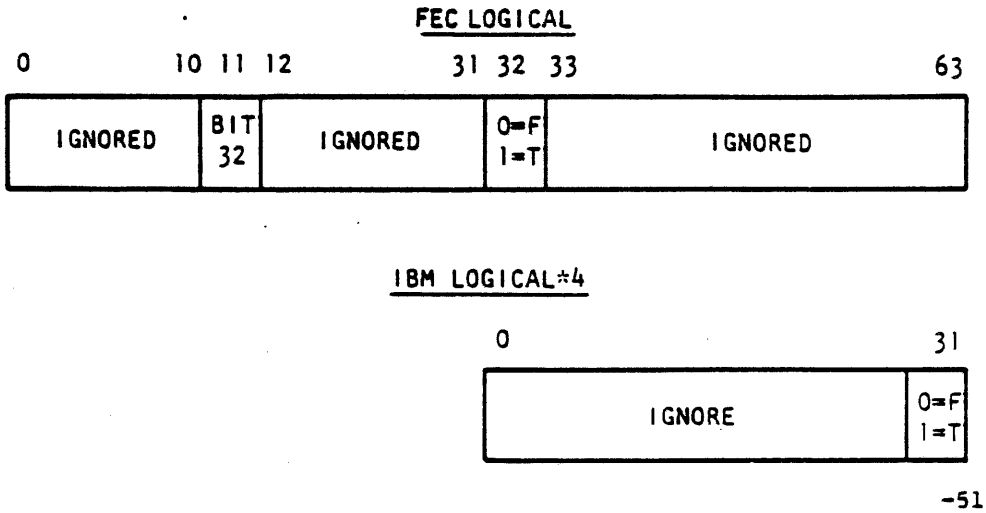
Figure 4-12 shows the conversion between a dual IBM Real*4 and an SC halfword floating packed number (format type 7).



-5188-

Figure 4-12 Dual IBM Real*4 to and From SC Halfword Floating packed Number

Figure 4-13 shows the conversion between an IBM Logical*4 and an SC logical (format type 8).



-5189-

Figure 4-13 IBM Logical*4 to and From SC Logical

4.4.10 FEC/SC Interconnect Hardware

The FEC I/O channel interconnect cables attach to the SC at its I/O panel. The SC I/O panel is at the bottom left rear (as viewed from the front) of the left SC cabinet bay. Figure 4-14 shows the I/O panel.

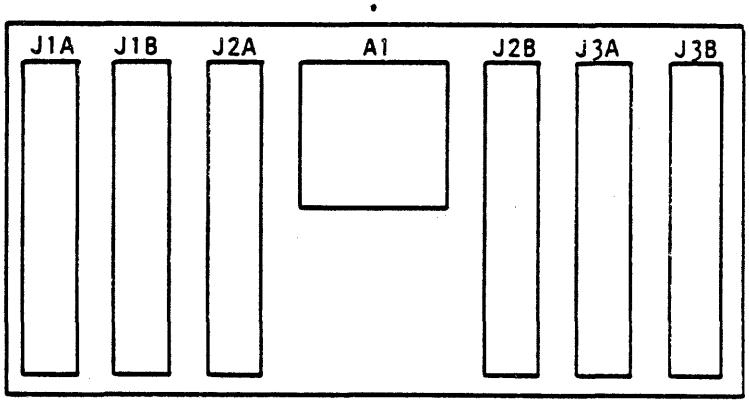


Figure 4-14 SC I/O Panel

The three connectors labeled J1B, J2B, and J3B in Figure 4-2 accept the IBM BUS0 IN, TAG IN and BUS1 OUT cables, respectively. Those labeled J1A, J2A, and J3A accept the BUS0 OUT, TAG OUT, and BUS1 OUT Note the inverted number order of the connectors. The plate labeled A1 is a door that provides access to the select-in/select-out jumper.

4.4.11 Configuring The Host Adapter Board

The mode switch at position 6-A on the IBM Host Adapter Board configures the SC. Switch numbers 1 and 2 select the addressing mode. Switch numbers 3, 4, and 5 select the bus mode. Switch 6 is reserved, switch 7 controls posting of TASKDN and TASKIN interrupts, and switch 8 controls the reporting of HISP busy. Table 4-7 explains the switch settings in detail.

Table 4-7 Configuration Mode Switch States (Switches 1 and 2)

SWITCH 1	SWITCH 2	ADDRESSING MODE NAME	MODE DESCRIPTION
CLOSED	CLOSED	MODE0	The SC responds to one address; address = SWA, SWB
OPEN	CLOSED	MODE1	The SC responds to eight addresses; the address range equals the SWA and MSB of the SWB. For example, the address range defined by SWA=3 and SWB=F is 38 through 3F (hexadecimal).
CLOSED	OPEN	MODE2	The SC responds to 16 addresses; SWA indicates range, while SWB is ignored. For example, the address range defined by SWA=3 and SWB=F is 30 through 3F (hexadecimal).
OPEN	OPEN	MODE3	The SC responds to two ranges of 16 addresses (32 total): SWA indicates the first range, SWB indicates the second range.

NOTES

1. SWA is the hexadecimal switch at position 3-A on the IBM Host Adapter Board. SWA is for the four MSB's.

2. SWB is the hexadecimal switch at position 4-A on the IBM Host Adapter Board. SWB is for the LSB's.

Table 4-8 explains the switch settings for switches 3, 4, and 5.

Table 4-8 Configuration Mode Switch States (Switches 3, 4, and 5)

SWITCH 3	SWITCH 4	SWITCH 5	ADDRESSING MODE NAME	MODE DESCRIPTION
CLOSED	CLOSED	CLOSED	NORMAL	Service only, data handshake.
OPEN	CLOSED	CLOSED	HIGH SPEED	Service/Data, data handshake.
CLOSED	OPEN	CLOSED	EXTENDED	BUS0 and BUS1, service only, data handshake.
OPEN	OPEN	CLOSED	HIGH SPEED/ EXTENDED	BUS0 and BUS1, service/data handshake.
OPEN	CLOSED	OPEN	DATA STREAM	Data streaming protocol is used with read or write; otherwise high speed is used.

Table 4-9 explains the switch settings for switches 6, 7, and 8.

Table 4-9 Configuration Mode Switch States (Switches 6, 7, and 8)

SWITCH	SWITCH STATE	ADDRESSING MODE NAME	MODE DESCRIPTION
6	OPEN	RESERVED	
	CLOSED	RESERVED	
7	OPEN	ASYNCH	Enable asynchronous posting of TASKDN and TASKIN.
	CLOSED	NOT ASYNCH	Wait for next CCW to post TASKDN and TASKIN. (Refer to note)
8	OPEN	CRETRY	Report HISP busy as command RETRY.
	CLOSED	NOT CRETRY	Report HISP busy as control unit busy.

NOTE

This option must always be closed. See ASYNCH in Table 4-5.

APPENDIX A

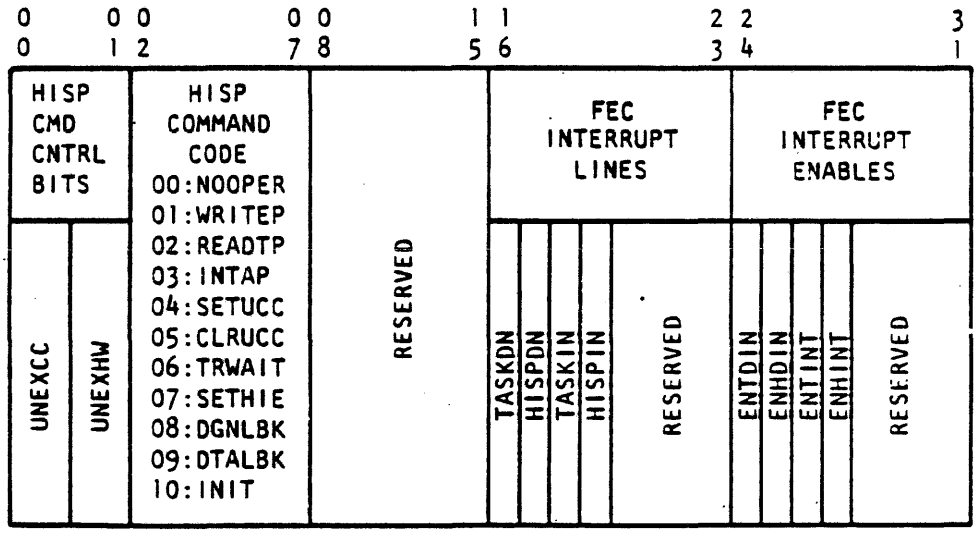
FEC COMMAND AND STATUS REGISTER

A.1 FEC COMMAND AND STATUS REGISTER (HCSR)

The host command and status register (HCSR) is used for communication between the FEC and the Host interface support processor (HISP). The HCSR is transferred from the FEC to the HISP by the WRITE HCSR CCW, and from the HISP to the FEC by the READ HCSR CCW or the SENSE CCW.

A.1.1 HCSR(u): Interrupt Control and HISP Commands

Figure A-1 describes the HCSR interrupt control and HISP commands format. Table A-1 defines the HCSR interrupt control and HISP commands fields.



-5191-

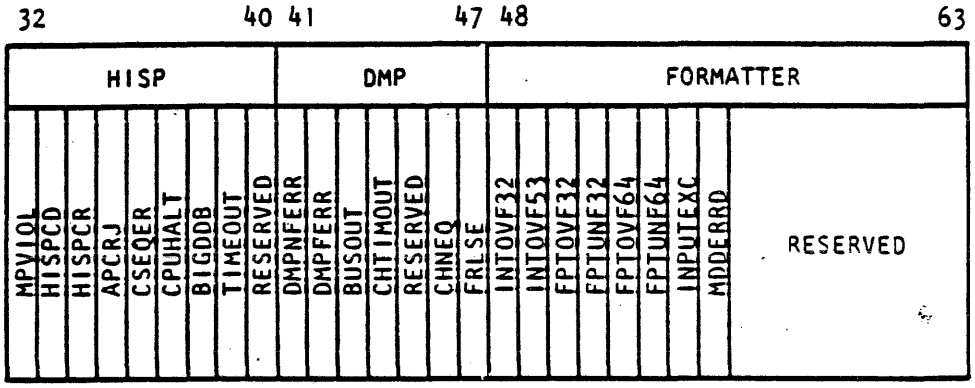
Figure A-1 HCSR(u): Interrupt Control and HISP Commands Format

Table A-1 HCSR(u): Interrupt Control and HISP Commands Fields

BITS	MNEMONIC	MEANING
00	UNEXCC	Unconditional execution bit. UNEXCC is set and cleared by commands to the HISP.
01	UNEXHW	Unconditional execution bit. UNEXHW is set by the HCSR written by the FEC.
02-07	COMMAND	HISP command code. This field contains the FEC command that the HISP is to execute. Valid commands are as follows: 0 no operation 1 write transfer packet 2 read transfer packet 3 interrupt SC 4 set UNEXCC 5 clear UNEXCC 6 wait for SC task completion 7 set FEC interrupt enable lines 8 diagnostic loop back 9 data loop back 10 HISP initialize 11-63 <reserved for future expansion>
08-15		<Reserved by FPS.>
16	TASKDN	Task done FEC interrupt line (attention).
17	HISPDN	HISP done FEC interrupt line (device end).
18	TASKIN	Task interrupt FEC interrupt line (unit exception).
19	HISPIN	HISP interrupt FEC interrupt line (unit check).
20-23	None	<Reserved by FPS.>
24	ENTDIN	Task done FEC interrupt enable line.
25	ENHDIN	HISP done FEC interrupt enable line.
26	ENTINT	Task interrupt FEC interrupt enable line.
27	ENHINT	HISP interrupt FEC interrupt enable line.
28-31	None	<Reserved by FPS.>

A.1.2 HCSR(1): Error Flags

Figure A-2 describes the HCSR error flags format. Table A-2 defines the HCSR error flag fields.



-5180-

Figure A-2 HCSR(1): Error Flags Format

Table A-2 HCSR(1): Error Flags Fields

BITS	MNEMONIC	MEANING
32	MPVIOL	Memory protection violation.
33	HISPCD	HISP commands disabled.
34	HISPCR	HISP command reject.
35	APCRJ	SC Command Reject
36	CSEQER	Command Sequence Error
37	CPUHALT	SC CPU Halt
38	BIGDDB	DDB Too Large for Available Buffer Space in HISP
39	TIMEOUT	HISP Time-out Waiting for SC Response to INTAP
40	None	<reserved>
41	DMPNFERR	DMP Nonfatal Error

Table A-2 HCSR (1): Error Flags Fields (cont.)

BITS	MNEMONIC	MEANING
42	DMPFERR	DMP Fatal Error
43	BUSOUT	Channel Bus Out Parity Error
44	CHTIMOUT	Channel Time-out (streaming only)
45		<reserved>
46	CHNEQ	Channel out Tags not Equal to Channel in Tags (streaming only)
47	FRLSE	Operation Aborted by Force Release CCW
48	INTOVF32	32-bit Integer Overflow
49	INTOVF53	53-bit Integer Overflow
50	FPTOVF32	32-bit Floating-point Overflow
51	FPTUNF32	32-bit Floating-point Underflow
52	FPTOVF64	64-bit Floating-point Overflow
53	FPTUNF64	64-bit Floating-point Underflow
54	INPUTEXC	Nonzero Unnormalized Input Value Encountered
55	MDDERRD	Main Data Double Bit Error Detected
56-63	None	<reserved>

APPENDIX B

FEC-SPECIFIC ERROR MESSAGES

B.1 APMGR ERROR MESSAGES

This section contains IBM/CMS-specific APMGR error messages and provides information to help the user interpret them and respond appropriately. The format of the messages follows standard CP and CMS message formats and is subject to the setting of the CP EMSG command. The EMSG setting determines whether a message is displayed at the virtual console and what parts of the message are displayed. Refer to the IBM manual VM/SP CP Command Reference for General Users listed in Table 1-2 for a complete description. The information in each message is printed in the form "FPSmmmnnns text", where:

- The expression "FPS" identifies the message as originating in the FPS UTIL64 or APEX64 software.
- The expression "mmm" indicates the component from which the message was issued. "MGR" indicates an APMGR message. The APEX64 user does not see APMGR messages, but those who maintain the APMGR can see them. The messages optionally appear at any combination of the APMGR's virtual console, virtual printer, or disk log file.
- The expression "nnn" contains the message serial number (by which the messages are ordered in the description below).
- The expression "s" identifies the type of message, where:

  E = error message
  I = information message
  W = warning message
  R = response message
  S = severe message
  T = terminal message

- The text of the message includes an explanation of the message, the system action to be expected, and the appropriate operator and/or programmer response. Each APMGR message is dated in the form mm/dd/yy hh:mm:ss, where:

  mm = the month
  dd = the day
  yy = the year
  hh = the hours
  mm = the minutes (this is the second "mm")
  ss = the seconds

FPSMGR500I mm/dd/yy hh:mm:ss INITIALIZATION OCCURRED SUCCESSFULLY.

Explanation: The APMGR performed its initialization process successfully.

System Action: The APMGR waits for requests through the virtual machine communication facility (VMCF), which allows two virtual machines to communicate with each other.

Programmer Response: None.

FPSMGR501I mm/dd/yy hh:mm:ss REQUEST FROM uuuuuuuu TO ATTACH ANY AP. NEW RQN = n.

Explanation: The virtual machine with a userid of uuuuuuuu requested that any available SC be attached. N is the job request number (RQN) assigned to the new job.

System Action: The APMGR attempts to attach the first available SC.

Programmer Response: None.

FPSMGR502I mm/dd/yy hh:mm:ss REQUEST FROM uuuuuuuu TO ATTACH AP nn NEW RQN = K.

Explanation: The virtual machine with a userid of uuuuuuuu requested that the SC whose logical unit number is nn be attached. K is the job request number (RQN) assigned to the new job.

System Action: The APMGR attempts to attach the first available address of the specified SC.

Programmer Response: None.

FPSMGR510I mm/dd/yy hh:mm:ss TERMINATION OCCURRED.

Explanation: The APMGR has received an external interrupt code of 1, which is a request to terminate.

System Action: The APMGR discontinues processing and returns to the CMS operating system.

Programmer Response: None.

FPSMGR511I mm/dd/yy hh:mm:ss ADDRESS cuu OF AP nn ATTACHED TO uuuuuuuu AS VIRTUAL ADDRESS vuu.

Explanation: The APMGR successfully attached real address cuu of logical SC number nn to virtual address vuu of the virtual machine whose userid is uuuuuuuu.

System Action: The APMGR continues with the current request or, if finished, waits for the next request to occur.

Programmer Response: None.

FPSMGR520W mm/dd/yy hh:mm:ss UNSUPPORTED EXTERNAL INTERRUPT CODE nnnn RECEIVED.

Explanation: The APMGR virtual machine received an external interrupt with a code of nnnn, which is not one of the external interrupt types used by the APMGR.

System Action: The APMGR ignores the interrupt and waits for the next request.

Programmer Response: None.

FPSMGR521W mm/dd/yy hh:mm:ss UNSUPPORTED VMCF CODE nnnn RECEIVED.

Explanation: An external interrupt sent as a result of a virtual machine communication facility (VMCF) request from the virtual machine whose userid is uuuuuuuu contained a VMCF subfunction code nnnn that was not among those the APMGR uses.

System Action: If the VMCF subfunction code nnnn specifies a VMCF SEND request, the APMGR issues the VMCF REJECT function against the request so that it does not remain outstanding. The APMGR ignores the invalid VMCF request and waits for the next request.

Programmer Response: If the virtual machine originating the request attempted to communicate with the APMGR, it used the wrong VMCF protocol. If the request was made by APEX64, then the VMCF parameter list used by APASGN is probably destroyed. This situation usually occurs following an unintentional memory modification; e.g., an out-of-range subscript. The VM/SP System Programmer's Guide contains a complete list of VMCF subfunction codes.

FPSMGR523W mm/dd/yy hh:mm:ss FORCE RELEASE OCCURRED FOR ADDRESS cuu UID mm OF AP nn.

Explanation: The APMGR found that the SC with a logical unit number of nn was assigned to logical userid mm through real address cuu, but cuu was not attached to any virtual machine.

System Action: The APMGR attaches the address, halts any outstanding I/O with a halt device instruction followed by a test I/O instruction, then executes a release channel program to free up the SC for subsequent use.

Programmer Response: The APEX64 user must always remember to release the SC with a call to APRLSE.

FPSMGR530E mm/dd/yy hh:mm:ss ERROR CODE nnnn FROM VMCF RECEIVE FOR uuuuuuuu.

Explanation: Error code nnnn was received from the virtual machine communication facility (VMCF) when the APMGR attempted to issue the VMCF RECEIVE subfunction for the virtual machine with a userid of uuuuuuuu. The purpose of the RECEIVE was to obtain the data describing the virtual machine's APMGR request.

System Action: The VMCF RECEIVE attempt did not complete. The APMGR discontinues processing for the current request and waits for the next request.

Programmer Response: If nnnn is 5, then the virtual machine making the request has terminated VMCF processing before the APMGR could finish the request. The requesting virtual machine terminated VMCF processing either by issuing the VMCF UNAUTHORIZE subfunction or by logging off. If nnnn is not 5, then the APMGR incorrectly issued the RECEIVE or an error occurred in CP during VMCF processing. Refer to the IBM manual VM/SP System Programmer's Guide listed in Table 1-2 for a complete description of VMCF error codes. If the error code indicates an apparent APMGR error, contact FPS Customer Service.

FPSMGR531E mm/dd/yy hh:mm:ss ERROR CODE nnnn FROM VMCF REPLY OR SEND/RECV FOR uuuuuuuu.

Explanation: Error code nnnn was received from virtual machine communication facility (VMCF) when the APMGR attempted to issue the VMCF REPLY subfunction to the virtual machine whose userid is uuuuuuuu. The purpose of the REPLY was to send the virtual machine the results of its attach request.

System Action: The VMCF REPLY attempt did not complete. The APMGR discontinues processing for the current request and waits for the next request.

Programmer Response: If nnnn is 5, then the virtual machine that made the request terminated VMCF processing before the APMGR could finish the request. The requesting virtual machine terminated VMCF processing either by issuing the VMCF UNAUTHORIZE subfunction or by logging off. If nnn is not 5, then the APMGR incorrectly issued the REPLY or an error occurred in CP during VMCF processing. Refer to the IBM manual VM/SP System Programmer's Guide listed in Table 1-2 for a complete description of VMCF error codes. If the error code indicates an apparent APMGR error, contact FPS Customer Service.

FPSMGR532E mm/dd/yy hh:mm:ss REQUEST FROM uuuuuuuu WAS OF INCORRECT LENGTH.

Explanation: The virtual machine with the userid of uuuuuuuu made a request to the APMGR, but the data sent to the APMGR to define the request was incorrect in length.

System Action: The APMGR sends a reply indicating the error condition to the virtual machine and discontinues processing of the current request, waiting for the next request to occur.

Programmer Response: Correct the request data and retry the request. If the requestor was APEX64, then the virtual machine communication facility (VMCF) parameter list used to communicate with the APMGR is probably destroyed. This situation usually occurs due to an unintentional memory modification; e.g., from an out-of-range subscript.

FPSMGR533E mm/dd/yy hh:mm:ss INVALID REQUEST TYPE xxxx RECEIVED FROM uuuuuuuu.

Explanation: The virtual machine with a userid of uuuuuuuu made a request to the APMGR, but the data sent to the APMGR defining the request contained a request type that is unknown to the APMGR.

System Action: The APMGR sends a reply indicating the error condition to the virtual machine, and the APMGR discontinues processing of the current request, waiting for the next request to occur.

Programmer Response: Correct the request data and retry the request. If the requestor was APEX64, then the data sent to the APMGR through the virtual machine communication facility (VMCF) interface is probably destroyed. This situation usually occurs due to an unintentional memory modification; e.g., from an out-of-range subscript.

FPSMGR534E mm/dd/yy hh:mm:ss UNABLE TO OBTAIN SENSE DATA FROM AP nn AT REAL ADDRESS cuu SIOCC c CSW uucc.

Explanation: The APMGR attempted to obtain sense data from device address cuu of the SC whose logical unit number is nn. The APMGR wanted to obtain the sense to determine if forced release processing was necessary for the SC, but the channel program used to obtain the sense failed. The condition code from the start I/O instruction used to initiate the sense was c. The unit status and channel status fields of the channel status word that was returned as a result of the failing channel program was uucc.

System Action: The APMGR continues processing. It will attempt force release processing again for this SC at a later time.

Programmer Response: If the start I/O condition code c was 2 or 3, the AP is off-line or nonexistent. If the condition code was 0 or 1, then the unit and channnel status fields uucc are valid. The unit status field contains unit and channel status values (uu and cc respectively). The bit settings corresponding to the uu portion can be found in Table 4-6. The cc values can be found in the IBM manual IBM System 370 Principles of Operation, listed in Table 1-2.

FPSMGR535E mm/dd/yy hh:mm:ss UNABLE TO ATTACH REAL ADDRESS cuu OF AP nn DURING FORCED RELEASE PROCESSING. COMPLETION CODE ccc.

Explanation: The APMGR determined that the SC with a logical unit number of nn was assigned through device address cuu, but the user who assigned it gave up the SC without first releasing it for subsequent use. Therefore, the APMGR attempted to force release the SC, but could not attach address cuu to itself so as to execute the release channel program. The CP ATTACH command used to perform the attempted attach finished with a completion code of ccc.

System Action: The APMGR continues processing. It will attempt force release processing again for the SC at a later time.

Programmer Response: Message FPSMGR536E is issued along with this message and gives the actual text of the reply that CP made to the failing ATTACH command. Correct the problem described by FPSMGR536E. Often, the SC is off-line at the address through which it is still assigned.

FPSMGR537E mm/dd/yy hh:mm:ss FORCED RELEASE FAILED FOR AP nn AT REAL ADDRESS cuu SIOCC c CSW uucc.

Explanation: The APMGR determined that the SC with a logical unit number of nn had been assigned through device address cuu, but the user who assigned it had given up the SC without first releasing it for subsequent use. Therefore, the APMGR attempted to force release the SC. The channel program used to perform the release, however, failed. The condition code from the start I/O instruction used to initiate the release channel program was c. The unit status and channel status fields of the channel status word returned as a result of the failing channel program was uucc.

System Action: The APMGR continues processing. It will attempt force release processing again for this SC at a later time.

Programmer Response: If the start I/O condition code c was 2 or 3, the SC address cuu is off-line or nonexistent. If the condition code is 0 or 1, then the unit and channel status fields uucc are valid. The unit status field contains unit and channel status values (uu and cc respectively). The bit settings corresponding to the uu portion can be found in Table 4-6. The cc values can be found in the IBM manual IBM System 370 Principles of Operation (listed in Table 1-2).

FPSMGR540S mm/dd/yy hh:mm:ss UNABLE TO WRITE TO LOG FILE filename. FSWRITE RETURN CODE nnnn

Explanation: The APMGR attempted to write a message to its log file, but did not receive a return code of zero from FSWRITE. The return code received is nnnn. <fileid> is the file name, file type, and file mode of the log file.

System Action: The APMGR attempts to write the failing message to its virtual console so that the message is saved. Message logging to the disk is stopped. If the failing message is the initialization message (FPSMGR500I), then the APMGR terminates.

Programmer Response: The FSWRITE return codes are found in the IBM manual VM/SP CMS Command and Macro Reference (listed in Table 1-2). The most common failures involve insufficient space available on the disk containing the log file or an illegal file ID for the log file. If the problem cannot be resolved immediately, the APMGR can be restarted without disk file logging enabled.

FPSMGR550T mm/dd/yy hh:mm:ss INVALID COMMAND LINE OPTIONS.

Explanation: The command line entered to start the APMGR contains a syntax error.

System Action: The APMGR terminates immediately.

Programmer Response: Restart the APMGR with a valid command line.

FPSMGR551T mm/dd/yy hh:mm:ss UNABLE TO ALLOCATE STORAGE FOR AP CONFIGURATION DATA.

Explanation: During its initialization process, the APMGR attempted to dynamically allocate storage via a DMSFREE request to CMS, but the allocation request failed. The APMGR was trying to obtain storage to build the internal data structures required to manage access to the SC's under its control.

System Action: The APMGR terminates immediately.

Programmer Response: Make sure that the APMGR has virtual storage sufficient for initialization. If it does not, the SC configuration data within module APCN64 could be invalid, causing the APMGR to calculate incorrectly the amount of storage needed.

FPSMGR552T mm/dd/yy hh:mm:ss VMCF AUTHORIZE UNSUCCESSFUL. RETURN CODE nnnn.

Explanation: The APMGR attempted to request permission to perform virtual machine communication facility (VMCF) processing, but execution of the VMCF AUTHORIZE subfunction (required to gain authorization) failed with return code nnnn.

System Action: After freeing dynamically allocated storage, the APMGR terminates.

Programmer Response: The IBM manual VM/SP System Programmer's Guide (refer to Table 1-2) contains a complete description of VMCF return codes. Either this is an APMGR error or the user is running a VM/370 system prior to release 6.0. Contact FPS Customer Service if the problem cannot be resolved.

FPSMGR553E mm/dd/yy hh:mm:ss SNDMSG FAILED-CMS USERID = uuuuuuuu -VMCF ERROR CODE = nnnn.

Explanation: The APMGR encountered an error while attempting to send a VMCF message to user uuuuuuuu. The IBM manual VM/SP System Programmer's Guide contains a complete description of VMCF return codes.

System Action: The APMGR continues processing. The APMGR will attempt to force release the offending user and remove him from the APMGR's internal priority queues.

Programmer Response: If nnnn is a 5, the user uuuuuuuu terminated VMCF processing before the APMGR could finish the request. The requesting virtual machine terminated VMCF communication by issuing a VMCF unauthorize subfunction or by logging off, HXing, or re-IPled CMS. If nnnn indicates an apparent APMGR error, call Customer Service.

FPSMGR554E mm/dd/yy hh:mm:ss APMDFT:INTERNAL ERROR - DEADUSER ARRAY OVERFLOW.

Explanation: An internal APMGR array that should never overflow.

System Action: The APMGR terminates

Programmer Response: You should never see this message. Call Customer Service.

FPSMGR560I mm/dd/yy hh:mm:ss VMCF MESSAGE RECEIVED FROM uuuuuuuuRQN = nn MTYPE = K.

Explanation: The APMGR received A VMCF message from user ID uuuuuuuuu, JOB number nn. K is the message type defined in the routine header for APSMSG.

System Action: The APMGR continues processing.

Programmer Response: None.

FPSMGR561I mm/dd/yy hh:mm:ss SENDING VMCF MESSAGE TO uuuuuuuu . RQN = n MTYPE = K.

Explanation: The APMGR sent a VMCF message to user ID uuuuuuuu. nn is the AP job number. K is the message type defined in the routine header for SNDMSG.

System Action: The APMGR continues processing.

Programmer Response: None.

B.2 APMGR ERROR MESSAGES

The following APMGR error messages may be displayed on the VM operator's console.

FPS300W mm/dd/yy hh:mm:ss AP MANAGER UNABLE TO OPEN FPS ACCOUNTING FILE.

Explanation: The APMGR encountered an error while attempting to open the FPS accounting file.

System Action: The APMGR continues but does not write accounting records.

Programmer Response: Make sure the APMGR has an available R/W disk accessed that is not full and restart the APMGR. The APMGR's console log file may contain additional messages that can be used to determine the cause of the failure.

FPS301W mm/dd/yy hh:mm:ss AP MANAGER UNABLE TO WRITE TO FPS ACCOUNTING FILE.

Explanation: The APMGR encountered an error writing an accounting record.

System Action: The APMGR continues but accounting data is lost.

Programmer Response: Make sure the APMGR's disk is accessed R/W and is not full. Then restart the APMGR. The APMGR's console log file may contain additional messages that can be used to determine the cause of the failure. the cause of the failure.

FPS302W mm/dd/yy hh:mm:ss AP MANAGER UNABLE TO OPEN LOG FILE.

Explanation: The APMGR encountered an error opening the APMGR log file.

System Action: The APMGR continues but does not write log numbers.

Programmer Response: Make sure the APMGR has an available R/W disk accessed that is not full and restart the APMGR. The APMGR's console log file may contain additional messages that can be used to determine the cause of the failure.

FPS303W mm/dd/yy hh:mm:ss AP MANAGER UNABLE TO WRITE TO LOG FILE.

Explanation: The APMGR encountered an error while writing to the log file.

System Action: The APMGR continues processing but log file data is lost.

Programmer Response: Make sure the APMGR has an available R/W disk accessed that is not full and restart the APMGR. The APMGR's console log file may contain additional messages that can be used to determine the cause of the failure.

FPS 304I mm/dd/yy hh:mm:ss AP MANAGER UNABLE TO SEND VMCF MESSAGE TO AP OPERATOR.

Explanation: The APMGR detected an error condition when attempting to send a VMCF message to a user running the APOPR program.

System Action: The APMGR continues processing.

Programmer Response: The user probably logged off, re-ipled CMS, or HX'ed out of the APOPR. Enter EXIT or quit to terminate APOPR execution.

FPS305E mm/dd/yy hh:mm:ss AP MANAGER UNABLE TO ASSIGN AN AP TO JOB nnn HEX.

Explanation: The APMGR could not attach an AP device address to the user's virtual machine.

System Action: The APMGR continues processing but the offending RQN nnn may need to be aborted using the APOPR ABORT command.

Programmer
Response: Make sure the AP device addressed are varied online. Make sure the sum of the max users values for the AP is not greater than the number of configured device addressed minus 1.

FPS306E mm/dd/yy hh:mm:ss AP MANAGER UNABLE TO DEASSIGN AN AP FROM JOB HEX.

Explanation: The APMGR could not de-assign an AP from job nnn.

System Action: The APMGR continues processing but the offending job nnn may need to be aborted using the APOPR ABORT command.

Programmer
Response: Examine the status of job nnn with the APOPR. If it still exists, use the APOPR ABORT command to abort the job.

FPS 307T mm/dd/yy hh:mm:ss AP MANAGER INITIALIZATION FAILED.

Explanation: An error was detected during APMGR initialization.

System Action: The APMGR terminates

Programmer
Response: Examine the APMGR's console log file to determine the cause of the failure. Correct the problem described in the console log file and re-start the APMGR.

FPS308I mm/dd/yy hh:mm:ss AP MANAGER/DAPEX COMMUNICATION BREAKDOWN - FORCE RELEASE IN PROGRESS FOR AP JOB nnn HEX.

Explanation: The user job nnn ended without calling APRLSE. The APMGR sttempts to do force release processing for the offending job.

System Action: The APMGR continues processing but the job nnn may need to be aborted using the APOPR ABORT command. Also, some of the accounting data for job nnn was lost.

Programmer
Response: CALL APRLSE or APIBMR to release the SC. Do not HX, IPL CMS, or logoff while the SC is assigned.

FPS310W mm/dd/yy hh:mm:ss AP MANAGER UNABLE TO READ FPS SITE PARAMETER FILE.

Explanation: The APMGR detected an error while reading the site parameter file.

System Action: The APMGR continues but uses default site parameters.

Programmer Response: Examine the APMGR's console log file to determine the cause of the failure. Correct the problem and restart the APMGR.

FPS311W mm/dd/yy hh:mm:ss AP MANAGER UNABLE TO WRITE FPS SITE PARAMETER FILE.

Explanation: The APMGR detected an error attempting to write to the site parameter file "PARAMS APIMG64"

System Action: The APMGR continues but the site parameters are not saved in "PARAMES APIMG64"

Programmer Response: Examine the console log file to determine the cause of the failure. Correct the problem and restart the APMGR.

FPS312W mm/dd/yy hh:mm:ss AP MANAGER UNABLE TO READ AP OPERATOR NAMES FROM SITE PARAMETER FILE.

Explanation: The APMGR detected an error while reading the site parameter file.

System Action: The APMGR continues but uses default site parameters.

Programmer Response: Examine the APMGR's console log file to determine the cause of the failure. Correct the problem and restart the APMGR.

FPS313S mm/dd/yy hh:mm:ss INTERNAL AP MANAGER ERROR - NO RQNS AVAILABLE.

Explanation: A severe internal APMGR error occured. The APMGR was unable to find a free job number to assign to a new job. This should never happen.

System Action: The APMGR terminates.

Programmer Response: Re-start the APMGR. Contact Customer Service.

B.3 DAPEX ERROR MESSAGES

This section contains IBM-specific APEX64 error messages and provides information to help the user interpret them and respond appropriately to them. The information is presented in the following order:

1. SC standard condition code in unpacked format (severity, message number, originator) followed by its 32-bit decimal representation. The messages are ordered by message number.
2. The APMSG message corresponding to the condition code.
3. A brief explanation of the code.
4. The normal corrective action.
5. A severity rating.

******* 7 1024 200 -536050688

TERMINAL: Nonzero unnormalized value encountered (APHERR -1024).

Explanation: Bit 54 (the INPUTEXC bit) is set in the FEC control and status register (HCSR). This indicates a format error in either the input or output data. The terminal severity of this error indicates that the Unit Check Mask is set up to terminate DDB processing as soon as the error is detected. This message is accompanied by another message identifying the FEC virtual address of the bad data.

Corrective Action: This user error must be traced by the user.

****** 2 1024 200 1074562048

WARNING: Nonzero unnormalized value encountered (APHERR -1024).

Explanation: Bit 54 (the INPUTEXC bit) is set in the HCSR. This condition indicates a format error in either the input or output data. The warning severity of this error message indicates that the unit check mask is set up to report the error at the end of DDB processing and continue processing the program, unless the user has called APERR to terminate on warning messages.

Corrective Action: This user error must be traced by the user.

****** 7 1026 200 -536050686

TERMINAL: Undefined error detected (bit 8 of HCSR(1) on) (APHERR -1026).

Explanation: The APEX64 routine APHERR found an undefined bit on in the HCSR. This error message indicates a hardware, Host Interface Support Processor (HISP) firmware, or APEX64 software malfunction.

Corrective Action: Contact FPS Customer Service.

****** 7 1030 200 -536050682

TERMINAL: Bus Out Parity Error detected (bit 11 of HCSR(1) on) (APHERR -1030).

Explanation: This condition results from either an SC interface error or an IBM channel error. If the same error occurs with other devices on the channel, it is probably a channel error. Otherwise, it is an SC error.

Corrective Action: Run the program again. If the error persists, contact FPS Customer Service.

******* 7 1031 200 -536050681

TERMINAL: Channel Time-out error detected (bit 12 of HCSR(1) on) (APHERR -1031).

Explanation: Channel error during data streaming. If another data streaming device on the channel has the same problem, it is a channel problem. Otherwise, it is an SC problem.

Corrective Action: Run the program again. If the error persists, contact FPS Customer Service.

******* 7 1032 200 -536050680

TERMINAL: Sequencer parity error (bit 13 of HCSR(1) on) (APHERR -1032).

Explanation: Severe Host Adapter malfunction.

Corrective Action: Contact FPS Customer Service.

******* 7 1033 200 -536050679

TERMINAL: Channel not equal (bit 14 of HCSR(1) on) (APHERR -1033).

Explanation: The Host Adapter detected a channel error during data streaming. Out responses from the channel do not equal responses sent by the SC to the channel.

Corrective Action: Contact FPS Customer Service.

******* 7 1034 200 -536050678

TERMINAL: Force Release CCW (bit 15 of HCSR(1) on) (APHERR -1034).

Explanation: The APMGR executed the "Force Release" CCW to unassign the SC which was assigned to this job. This condition is expected only when the job is aborted by issuing the APOPR ABORT command.

Corrective Action: None

****** 7 1035 200 -536050677

TERMINAL: Undefined error detected (bit 24 of HCSR(1) on) (APHERR -1035).

Explanation: The APEX64 routine APHERR found an undefined bit on in the HCSR. This condition indicates a hardware, HISP firmware, or APEX64 software malfunction.

Corrective Action: Contact FPS Customer Service.

****** 7 1036 200 -536050676

TERMINAL: Undefined error detected (bit 25 of HCSR(1) on) (APHERR -1036).

Explanation: The APEX64 routine APHERR found an undefined bit on in the HCSR. This condition indicates a hardware, HISP firmware, or APEX64 software malfunction.

Corrective Action: Contact FPS Customer Service.

****** 7 1037 200 -536050675

TERMINAL: Undefined error detected (bit 26 of HCSR(1) on) (APHERR -1037).

Explanation: The APEX64 routine APHERR found an undefined bit on in the HCSR. This represents either a hardware, HISP firmware, or APEX64 software malfunction.

Corrective Action: Contact FPS Customer Service.

****** 7 1038 200 -536050674

TERMINAL: Undefined error detected (bit 27 of HCSR(1) on) (APHERR -1038).

Explanation: The APEX64 routine APHERR found an undefined bit on in the HCSR. This condition indicates a hardware, HISP firmware, or APEX64 software malfunction.

Corrective Action: Contact FPS Customer Service.

****** 7 1039 200 -536050673

TERMINAL: Undefined error detected (bit 28 of HCSR(1) on) (APHERR -1039).

Explanation: The APEX64 routine APHERR found an undefined bit on in the HCSR. This condition indicates a hardware, HISP firmware, or APEX64 software malfunction.

Corrective Action: Contact FPS Customer Service.

****** 7 1040 200 -536050672

TERMINAL: Undefined error detected (bit 29 of HCSR(1) on) (APHERR -1040).

Explanation: The APEX64 routine APHERR found an undefined bit on in the HCSR. This condition indicates a hardware, HISP firmware, or APEX64 software malfunction.

Corrective Action: Contact FPS Customer Service.

****** 7 1041 200 -536050671

TERMINAL: Undefined error detected (bit 30 of HCSR(1) on) (APHERR -1041).

Explanation: The APEX64 routine APHERR found an undefined bit on in the HCSR. This condition indicates a hardware, HISP firmware, or APEX64 software malfunction.

Corrective Action: Contact FPS Customer Service.

****** 7 1042 200 -536050670

TERMINAL: Undefined error detected (bit 31 of HCSR(1) on) (APHERR -1042).

Explanation: The APEX64 routine APHERR found an undefined bit on in the HCSR. This condition indicates a hardware, HISP firmware, or APEX64 software malfunction.

Corrective Action: Contact FPS Customer Service.

******* 7 1505 200 -536050207

TERMINAL: Requested AP is off-line (APASGN -1505).

Explanation: APEX64 attempted to gain access to an (or any) SC by a request to the APMGR, but the requested SC (or all SC's if the request was for any SC) was off-line.

Corrective Action: Ensure that the desired SC is working and on-line, then retry the request.

******* 7 1506 200 -536050206

TERMINAL: AP manager is not available (APASGN -1506).

Explanation: APEX64 attempted to gain access to an SC by a request to the APMGR, but the APMGR did not respond to the request because the APMGR was not running.

Corrective Action: Re-start the APMGR. (See the installation instructions for details.)

APPENDIX C

RESERVED ROUTINE NAMES

APEX64 uses a standard convention for naming subroutines and common blocks: the letters "AP" are used as the first two letters of every subroutine or common block name. To avoid conflicts with APEX64, users must not give their subroutines names that begin with "AP". This appendix is an alphabetical list of the subroutines whose names do not follow the standard naming convention of beginning a subroutine name with the letters "AP". To avoid conflicts, users must not use these names when naming their own subroutines.

A1TOS1	F32TOR	MGRUID	S4TOS1
A4TOS1	F64TOH	MUL128	SCLF64
ABSF64	F64TOR	MUL32	SEEKF
ADD31	F64TOS	MUL64	SETBRK
ADD64	FTI32R	MULCPX	SHF128
ADDCPX	FTOF32	MULF64	SHF32
ADDF64	FTOF64	NAMEF	SHF64
AND32	FTOI32	NEG32	SIZEMD
AND64	GETPS	NEG64	SIZEPS
CLI	GETREG	NEGF64	SIZMEM
CLOSEF	HADR	NOT32	STOF64
CLRBRK	HMEM	NOT64	STOI32
CMP32	HTOF64	OPENF	STOI64
CMP64	HTOI32	OR32	SUB32
CMPF64	HTOI53	OR64	SUB64
CPTIME	HTOU64	PNLCLR	SUBCPX
DATE64	I32DAT	PNLCMD	SUBF64
DBHDLR	I32TOF	PUTREG	TELLF
DBRNLP	I32TOH	RCPF64	TIME64
DBUG64	I32TOI	RDBINF	U64TOH
DGASGN	I32TOS	RDLINF	UCMP32
DIV32	I53TOH	RDPRMF	UCMP64
DIV64	I53TOI	REWNDF	WTBINF
DIVCPX	I64TOS	RTOF32	WTDMA
DIVF64	ITOI32	RTOF64	WTLINF
EXITF	ITOI53	RUNAP	WTPAGF
EXP32	L64TOL	RUNDMA	WTRUN
EXP64	LOADBT	S1TOA1	XOR32
EXPCPX	LOADPS	S1TOA4	XOR64
EXPF64	LTOL64	S1TOS4	

$BS 2-2
$FM 2-2
:HOST: 3-5,6
:HOSTCHAR: 3-5,6

Accessing SC software 2-6
ADC (auto-directed calls) 2-28
ADC FEC program 2-28
Address recognition 4-6
Allocating an SC 3-1
APAL64 2-7,8
APAL64 options 2-8
APCRJ A-3
APDBUG64 2-7,13
APDBUG64 options 2-13
APEX error messages, IBM-specific B-14
APEX64 2-28
  HASI file creation 2-38
  job compilation or assembly 2-37
  library building 2-38
  necessary job files 2-37
  program development 2-36
  program execution 2-41
  programming 2-28
APEX64 program execution 2-28
APEX64 programming 2-28
APFTN64 2-7,8,9,26
APFTN64 file to FEC file conversion logic 3-17
APFTN64 file to FEC file conversion procedure 3-15
APFTN64 options 2-9
APLIBR64 2-7,10
APLIBR64 options 2-10
APLINK64 2-7,11,26
APLINK64 options 2-11
APMGR error messages B-1,10
APMGR, role of 3-1
Auto-directed calls (ADC) 2-28

Batch mode execution of an SJE job 2-26
BIGDDB A-3
Building libraries 2-38
Bytes returned by sense I/O type command 4-12

Channel command word (CCW) 4-3
Channel commands 4-7
Channel interface protocol 4-6
Channel interrupts 4-13
Channel, role of 4-2
Channels, IBM 4-1
CHNEQ A-3
CO 2-3
Codes, condition 2-5
Command A-2
Command and status register (HCSR), host A-1
Command construction rules 2-1
Command description example 2-2
Command line options 2-2
Command line syntax 2-1
Command options, syntax 2-2
Command retry 4-15
Commands, channel 4-7
Commands, general form of 2-2
Commands, SJE JDL 2-19
Compilation files in the SJE job 2-17
Compilation or assembly for SC subroutines 2-38
Compilation or assembly of source files 2-17
Compiling or assembling files in an APEX64 job 2-37
Compiling/assembling using FEC facilities 2-40
Condition codes 2-5
Configuration mode switch states (switches 1 and 2) 4-21
Configuration mode switch states (switches 3, 4, and 5) 4-22
Configuration mode switch states (switches 6, 7, and 8) 4-23
Configuring the Host Adapter Board 4-20
Configuring the IBM Host Adapter Board 4-20
Control and HISP commands format A-1
Conventions used in manual 1-2
Conventions, default CMS naming 2-3
Conversion utilities, data 3-8

Conversion utilities, example 3-14
Conversion utilities, how to use 3-9
Conversion utilities, processing steps 3-11,15
CPU, IBM, role of 4-2
CPU, SC, role of 4-3
CPUHALT A-3
Creating the executable load module 2-40
Creating the HASI file using APLINK64 2-39
Creating the image load module using APLINK64 2-18
CSEQER A-3

DAPEX error messages B-14
Data and file conversion routines 3-9
Data conversion utilities 3-8
  conversion steps 3-15
  example 3-14,18
  logic flow 3-16
  processing steps 3-11
  when necessary to use 3-9
Data conversion, FEC/SC 3-8
Data descriptor block (DDB) 4-2
Debugging SC subroutines 2-39
Default CMS filetypes, record formats, and block sizes 2-4
Default naming conventions, CMS 2-3
Delimiters, conventions for use of 2-2
Detaching an SC 3-3
Development of programs under APEX64 2-36
DMPFERR A-3
DMPNFERR A-3
ENHDIN A-2
ENHINT A-2
ENTDIN A-2
ENTINT A-2
Error flags A-3
Error message interpretation 2-5

Error messages, APEX, IBM-specific B-14
Error messages, APMGR B-1
Example program usage 2-14
Examples of invoking PDS programs 2-14
Executing an SJE job from the FEC 2-25
Executing an SJE job from the SC 2-23
Executing an SJE job in CMS batch mode 2-26
Executing the SJE job 2-18
Execution 2-41
Execution from the FEC 2-25
Execution of the APEX64 load module 2-41
Execution using SJE 2-18
Extended bus 4-6

FEC command and status register (HCSR) A-1
FEC compilation/assembly during the APEX64 job 2-40
FEC data file to SC data file conversion logic 3-13
FEC data file to SC data file conversion procedure 3-11
FEC file prefixes 3-5
FEC file to SC file conversion steps 3-10
FEC interface support processor 4-4
FEC program 2-28
  ADC 2-28
  UDC 2-29
FEC to SC file conversion steps 3-10
FEC/SC interconnect hardware 4-20
File conversion, SC data to APFTN64 unformatted record 3-8
Files for an APEX64 job 2-37
Files for an SJE job 2-17
Flags format A-3
Forced-release processing 3-3
Format conversion 4-4,15
Formatter 4-4
Formatter, role of 4-4
FPS-164 manuals, table showing relations of 1-5
FPTOVF32 A-3
FPTOVF64 A-3
FPTUNF32 A-3
FPTUNF64 A-3

General form of commands 2-1,2

Hardware for FEC/SC communication 4-1
Hardware specific to IBM FEC computers 4-6
HASI file creation, APEX64 2-38
HCSR (host command and status register) 4-4; A-1
HCSR(1)
  error flags A-3
  error flags fields A-3
HCSR(u)
  Interrupt control and HISP commands fields A-2
  Interrupt control and HISP commands fields format A-1
  Interrupt control and HISP commands A-1
High-speed streaming 4-6
High-speed transfer 4-6
HISP (host interface support processor) 4-1,4,15; A-1
HISP command and interrupt control fields A-2
HISP, role of 4-3
HISPCD A-3
HISPCR A-3
HISPDN A-2
HISPIN A-2
Host Adapter 4-4
Host command and status register (HCSR) 4-4; A-1
Host interface support processor (HISP) 4-4

I/O to FEC files 3-5
IBM CPU, role of 4-2
IBM Host Adapter Board, configuring 4-20
IBM integer*4 4-15
IBM integer*4 format 4-17
IBM integer*4 to and from SC integer*4 4-15
IBM integer*4 to and from SC long integer 4-16
IBM integer*8 4-16
IBM integer*8 to and from SC long integer 4-16
IBM logical*4 format 4-19
IBM logical*4 to and from SC logical 4-19
IBM real*4 format 4-18,19
IBM real*4 to and from SC floating-point number 4-18
IBM real*8 format 4-18
IBM real*8 to and from SC floating-point number 4-18
IBM unformatted*8 to and from SC word type 4-17
IBM-specific APEX error messages B-14
IBM/CMS FEC-specific hardware of the FEC/SC system 4-6
Image load module for the SJE job 2-17
INPUTEXC A-3
Interconnect hardware, FEC/SC 4-20
Interface busy conditions 4-15
Interface hardware block diagram 4-5
Interface sense byte 0 fields 4-9
Interface sense byte 1 fields 4-10
Interface sense byte 2 fields (image of the unit address switches on the adapter) 4-11
Interface sense byte 3 fields (image of the eight mode switches on the adapter) 4-11
Interface sense byte descriptions 4-9,10,11
Interface sense byte format 4-9
Interpretation of error messages 2-5
Interrupt control and HISP command fields A-2
Interrupts, channel 4-13
INTOVF32 A-3
INTOVF53 A-3

JDL command overview 2-19
JDL commands, SJE 2-19
JDL control statements 2-20
JDL service request statements 2-21

Library building under APEX64 2-38
Line syntax, command 2-1

Linking to FEC-specific library routines 3-19
Load module creation under APEX64 2-40
Load module format 4-17
LOGON, use of 2-2

Manuals, FPS-164, table showing relations of 1-5
MDDERRD A-3
Messages, APMGR error B-1
Messages, error, IBM-specific B-14
Messages, error, interpretation of 2-5
MPVIOL A-3

Naming conventions, default CMS 2-3
Number of users 3-2

Options, command line
  $BS 2-2
  $FM 2-2
Options, SC software 2-7

PDS command line syntax 2-1
PDS commands and options 2-7
  APAL64 2-8
  APDBUG64 2-13
  APFTN64 2-9
  APLIBR64 2-10
  APLINK64 2-11
  SJE 2-14
PDS programs, examples 2-14
PRESERVE 3-6
Program development for APEX64 2-36
Program development software programs 2-7
Program development, SJE 2-16
Program development, tools necessary for 2-1
Programming under APEX64 2-28
Programs, example usage 2-14
Protocol, channel interface 4-6
Providing the files for the job 2-17
Punctuation, as delimiters in command language 2-2

Release channel program 3-3
Releasing an SC 3-3
Releasing and detaching an SC 3-3
Restore 3-6
Roll-in/roll-out (RIRO) 3-4

SC allocation 3-1
SC CPU, role of 4-3
SC device address 0 3-3
SC floating-point format 4-18
SC halfword floating packed format 4-19
SC halfword packed integer format 4-17
SC I/O panel 4-20
SC integer 4-15
SC logical format 4-19
SC long integer 4-15,16
SC selection process 3-2
SC software options 2-7
SC word type format 4-17
SC's FEC interface internal structure 4-4
SC, releasing and detaching 3-3
SC
  accessing software 2-6
  debugging subroutines 2-39
  software options 2-7
Selecting an SC 3-2
Sense bytes 4-8
Sense I/O type command 4-12
SJE 2-16
SJE data conversion utilities 3-8
SJE JDL commands 2-19
SJE options 2-14
SJE program development 2-17
SJE program development and execution 2-16
SJE user attention command 2-27
  batch mode execution 2-26
  creating the image load module 2-17
  execution from the FEC 2-25
  execution from the SC 2-23
  job compilation 2-17
  job execution 2-18
  necessary job files 2-17
  program development 2-16
Software options, SC 2-7
Software, accessing 2-6
Status register (HCSR), host command and A-1

Subroutine naming conventions C-1
SUM APIMG64, name reserved for SC's SUM 2-6
Supplying the files 2-37

TASKDN A-2
TASKIN A-2
The role of the APMGR 3-1
The role of the channel 4-2
The role of the formatter 4-4
The role of the HISP 4-3
The role of the IBM CPU 4-2
The role of the SC CPU 4-3
The SC's FEC interface internal structure 4-4
TIMEOUT A-3
Tools necessary for program development 2-1

UBRAER A-3
UDC (user-directed calls) 2-29
UDC FEC program 2-29
UDC/ADC FEC programs 2-28
UNEXCC A-2
UNEXHW A-2
Unit check mask 4-12
Unit check mask bit fields 4-12
Unit status field bits 4-13
User attention command 2-27
User-directed calls (UDC) 2-29
Users, maximum number of 3-2
Using auto-directed calls (ADC) 2-28
Using preserve and restore 3-6
Using user-directed calls (UDC) 2-30
Utilities, data conversion 3-8

Virtual Front Panel (VFP) 4-3

First Class
Permit No. A-737
Portland,
Oregon

BUSINESS REPLY

No postage stamp necessary if mailed in the United States

Postage will be paid by:

FLOATING POINT SYSTEMS, INC.

P.O. Box 23489

Portland, Oregon 97223

Attn: Technical Publications

READER'S COMMENT FORM

Your comments will help us improve the quality and usefulness of our publications. Please fill out and return this form to: Floating Point Systems, the mailing address is on the back.

Title of document _____

Name/Title _____ Date _____

Firm _____ Department _____

Address _____

Telephone _____

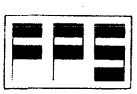
I used this manual...

- ☐ as an introduction to the subject
- ☐ as an aid for advanced training
- ☐ to instruct a class
- ☐ to learn operating procedures
- ☐ as a reference manual
- ☐ other _____

I found this material...

	Yes	No
accurate/complete	☐	☐
written clearly	☐	☐
well illustrated	☐	☐
well indexed	☐	☐

Please indicate below, listing the pages, any errors you found in the manual. Also indicate if you would have liked more information on a certain subject.



FLOATING POINT
SYSTEMS, INC.

...the world leader in array processors

CALL TOLL FREE (800) 547-1445
Ex. 4999. P.O. Box 23489 (S 500),
Portland, OR 97223 (503) 641-3151,
TLX: 360470 FLOATPOIN BEAV