MICHIGAN STATE UNIVERSITY

COMPUTER LABORATORY

6000 SCOPE MEMO NO. 124.1

March 25, 1980

FREND

1.0 Introduction.

This 6SM describes FREND, the operating system developed for the 7/32 front-end computer system at Michigan State University. FREND serves as a front-end for the SCOPE-HUSTLER system on the CDC Cyber 750, handling interactive support as well as driving printers for the batch printer subsystem.

The entire front-end development project consisted of extensive changes to software on the Cyber mainframe, as well as the development of FREND. The Cyber software is described in 3 separate 6SMs: 6SM 135 (MANAGER and associated routines), 6SM 134 (1FP and associated routines), and 6SM 131 (ARGUS modifications for MERIT interactive support). The interaction between 1FP and FREND is described both in this 6SM and in 6SM 134. The interaction between MANAGER and FREND is described in 6SM 135.

2.0 External reference specifications.

2.1 Interactive support.

The front-end supports asynchronous, teletype compatable ASCII terminals at 110, 300, and 1200 baud. Both dial-up and hardwired terminals are supported. Auto-baud-rate detection is provided on the 110/300 baud lines. The total aggregate baud-rate supported is approximately 100,000 baud.

2.1.1 Echoplex operation.

All terminals can be run in either half-duplex mode, with no echo-back of characters, or in full-duplex mode, with FREND echoing back each received character.

2.1.2 Typed-ahead input.

Data may be entered ahead of any process on the Cyber mainframe which is reading that data. Up to 5 typed-ahead lines may be entered. Any additional lines are discarded, and a message is returned to the terminal.

2.1.3 Line length.

Input lines may be up to 240 characters in length. The maximum length is specified by the INLEN command. Lines longer than the current INLEN setting are automatically broken, and begin a new line. There is no restriction on the length of output lines.

2.1.4 Character sets.

FREND supports terminals in the ASCII character set, as well as the APL typewriter-pairing and bit-pairing character sets. Other alternate character sets may be defined in the future, as the need arises.

For interactive connections, all data is represented within the front-end in ASCII. APL output is translated as it is sent to the terminal. 1FP is responsible for translations between ASCII and the character set in use by the Cyber mainframe (either ASCII or Display-code).

For the batch printers. FREND supports both Display-code and ASCII. Printer data is not translated until the auto-driver-channel sends the characters to the printer - so print buffers may be found in either display-code or ASCII within the front-end.

2.1.5 Control characters.

FREND supports a number of special control functions, each one selected by a single specific control character. The ALTER command may be used to change the control characters which invoke a specific control function. The supported control functions are:

ABORT	abort the current process (ESC).
BKSP	character delete (BS).
CANCEL	line delete (CAN).
CONT	line continue (ETB).
ECHO	toggle echoback in full duplex (SYN).

EOL	end-of-line (CR).
HALTOUT	stop the current output line immediately (DC3).
LECHO	echo current input line (ACK).
LITERAL	literal next (DLE).
RETIN	retrieve the last typed-ahead input line (NAK).
STARTOUT	restart output stopped by STOPOUT (DC1).
STOPOUT	stop the current output line (DC4).
TERMOUT	terminate the current output line (SUB).

These characters are fully described in chapter 8 of the INTERACTIVE system user guide.

2.2 Batch printer support.

The front-end supports the use of ASCII line printers to print batch output. Control of these printers is handled by special terminals called "I/O commander terminals" (the operator's terminal and the 7/32 console are enabled as I/O terminals).

2.2.1 Character sets.

The batch printer subsystem will print either ASCII fancy (AF) or old MISTIC (OM, known also as Display Code). Automatic character set determination is done on a line-by-line basis. Non-printing ASCII characters are automatically deleted from ASCII output. with a suitable warning.

Currently, FREND is driving one full ASCII (96-character) printer and two upper-case-only (64-character) printers. An output file may be directed to one or the other of these at the user's discretion.

2.2.2 Job recovery

With the exception of a fatal Cyber mainframe crash, output jobs are recovered and returned to the output queue in the event of any hardware or software malfunction, either in the 7/32 or in the Cyber mainframe.

2.2.3 Line length.

The maximum length of a print line is 136 characters (plus a one-character carriage control). Lines longer than this are folded at column 136, with the remainder of the line printed, single spaced, on subsequent lines.

2.2.4 Special features.

Several new features are present in the FREND printer system that are not available through ARGUS. Among them are the capability to skip forward on print files, more flexible control when printing on special forms, and better recovery in case of a failure. Jobs with dayfiles have the dayfile printed at the beginning of the job, on the right hand side of the page (to even out ribbon and hammer wear).

2.3 FREND commands.

FREND supports a number of terminal commands. A command may be issued either by the user at a terminal (the command must be prefixed by the current front-end command character) or by the Cyber mainframe (via a data port). The command must appear alone on a line. Blanks and commas serve to delimit parameters, and the command is free field. All front-end commands applicable to users are described in chapter 8 of the INTERACTIVE system users guide. All non-user commands are described in the FREND OPERATOR GUIDE. All batch printer commands are described in the FREND BATCH PRINTER OPERATOR'S GUIDE.

The commands may be classified into four groups--user commands, I/O commands. operator commands. and console commands. In the following command descriptions, the standard documentation notation is used. Required parameters are enclosed in braces ("{}"). Optional parameters are enclosed in brackets ("[]"). Parameters from which one must be chosen are separated by the vertical slash ("¦") (the default value in such a case is underlined). Values that must be entered verbatim are in UPPER CASE. Values that must be filled in are in lower case.

2.3.1 User commands.

The following commands are available to any user on the front-end.

ALTCHAR,{charset¦OFF¦NONE}[,AUTO].
Selects an alternate character set charset; currently charset may be APLTYPE or APLBIT. If AUTO is specified, the terminal may be toggled between the alternate character set and the default ASCII set. When the terminal is in an alternate character set, FREND will translate AF and OM output into that set. NONE and OFF are synonyms; they disable all alternate character set processing.

ALTER,{RESET¦LIST¦char=function...}.
Sets the given character to the given function. RESET

resets to the default character translations. LIST lists all the translations currently in effect.

BINARY,[ON|OFF].

Turns on and off binary mode. When BINARY,ON is in effect, no control characters are interpreted. Data is read verbatim, as a continuous stream. Binary mode is exited by the break key, or by a "BINARY,OFF" command issued by the Cyber mainframe.

CCTL,{ON|OFF}.

CCTL,OFF suppresses all carriage controls on AF and OM files. These lines will be single spaced, with the first character printed. CCTL,ON reverts to the standard method of processing carriage controls on these files.

CDELAY[,CR=N][,LF=N][,HT=N][,VT=N][,FF=N]

Sets N as the delay in characters (nulls) for CR, LF, HT, VT, or FF.

CONSTAT.

Displays current socket and port connection numbers in decimal.

DEQ[,N][,LIST].

Dequeue N typed-ahead lines. If N is omitted, 1 line is dequeued. If LIST is specified, the lines are listed as they are dequeued. Dequeued lines are thrown away.

ECHO,{ON|OFF}.

Turns on and off echo-back of input characters. Normally is ON on 300 and 1200 baud connections, and OFF on 110 baud connections.

FECC,{c|DATA|RESET}.

Reset the current front end command character to c, where c is a single character, or one of the acceptable abbreviations for ALTER. If c = DATA, no fecc is processed. If c = RESET, the fecc is reset to a percent sign.

FESTAT.

Returns the number of users connected to the front-end, broken down by baud rate and destination connection (MERIT or MISTIC).

FLIP.

Toggles between two connections. The connections may be two MISTIC connections, one MISTIC and one MERIT, or two MERIT. The current port is printed after the flip is completed. FLIP is only valid if there are 2 connections.

INLEN,NNN.
Sets NNN as the new maximum input line length. Any input line will automatically be split when it exceeds the length. The value must be between 1 and 240. If a zero value is given. it is reset to 240.

JOBSTAT.
Prints the current status of the mainframe side of a connection. Restricted to MISTIC connections only.

LOGIN.
Establishes a connection to MISTIC. Only a total of two connections are allowed, including any MERIT connection.

LOGINMSG.
Displays the current login message.

MIX,{ON|OFF}.
MIX.ON allows output from two connections to appear at the terminal intermixed. MIX,OFF allows only the output from the connection currently flipped to (the primary connection) to appear at the terminal.

MSU.
If the second socket connection is to MISTIC, the connections in the socket are flipped so that MISTIC is the primary socket connection. The user must have 2 connections.

NET.
If the second socket connection is to MERIT, the connections in the socket are flipped so MERIT is the primary connection.

NETCNT,dest.
Initiate a connection to MERIT. Dest is the destination code (MS, UM, or WU). NETCNT,MS is allowed only from the 7/32 console and the operator's terminal (sockets 1 and 2).

NPC[,{OFF|PART|ON}[,{CTRL|MNEM|c}]].
Sets the non-printing character interpretation mode. OFF suppresses interpretation - all control codes are sent to the terminal. PART causes all non-printing characters that do not have associated hardware functions to be interpreted. ON causes all non-printing characters to be interpreted. MNEM interprets the characters in their mnemonic form (e.g. [ETX]). CTRL causes interpretation in the "CTRL" form (e.g. <CTRL-C>). The "c" form causes the character c to be printed in place of the non-printing character. NPC alone implies "NPC,PART,%". "SHOWNPC" is a synonym for "NPC".

PARITY,{EVEN|ODD|NONE|OFF|ON}.
Sets the output data parity to the specified value. OFF is equivalent to NONE. ON is equivalent to EVEN.

QUIT.
QUIT simulates a disconnect on the current port connection. If the user has two connections, the remaining connection becomes the primary connection. If there is only one connection, the user is disconnected from the front-end.

READER,{ON|OFF}.
Sets the reader flag in the socket. READER,ON causes a DC3 to be sent to the terminal when no input buffer slots are left in the port. A DC1 is sent when all input buffer slots in the port become free. READER,OFF stops the DC1/DC3 process.

RMARGIN,NNN.
Sets NNN as the current right margin. Output lines longer than this will be folded with the remainder printed single-spaced on subsequent lines.

TERMINAL,type.
Sets the indicated terminal type and associated attributes.

TERMSTAT.
Displays the current terminal status. This information includes right margin, input line length, control character delay values, parity selection, terminal type, CCTL, READER, MIX and NPC flag values, backspace echo character, and the alternate character set selection (if any).

TIME.
Displays the current time.

2.3.2 I/O commands.

The following commands are available only to those terminals authorized as I/O Commander Terminals. Currently, this includes the 7/32 console, the operator's terminal, and the I/O room Teleray terminal. All preceding commands are also available.

ACK,NN.
Acknowledges I/O diagnostic messages (such as PAPER OUT). This command keeps the messages from recurring every two minutes.

ALIGN,NN[,MMM].

Aligns forms. The specified printer prints the first MMM PRUs of the job over and over again, slowly, to allow forms alignment. "ALIGN,NN,0" will cause the printer to pick up from where it was, at full speed. The default number of PRUs is 10.

BANNER,NN.
Sets full banner pages on printer NN. This is the normal condition. This command reverses the effect of the NOBANNER command.

BKSP,NN[,MMM].
Backspaces the job on printer NN by MMM PRUs (default is 10).

CHANGECS,NN.
Flips back and forth between the AF and OM character sets when carriage controls are suppressed, since automatic character set detection cannot take place.

END,NN.
Ends the job on printer NN. The entire dayfile will always print, if one is present.

GO,NN.
Restarts the specified printer when it is waiting on a "PM" carriage control message.

KILL,NN.
Kills the job on the specified printer. This causes the dayfile to stop printing as well. The END command must have been issued previously.

NOBANNER,NN.
Turns off banner pages on the specified printer. A single line with the job sequence number is printed instead. This will be counteracted by either the BANNER or the ROUTE command.

OFF,{PR¦nn}.
Turns the specified printer logically off. Any job currently printing will complete before the printer stops. OFF,PR turns off all printers.

ON,{PR¦nn}.
Turns the specified printer logically on. ON,PR turns on all printers.

PAGELIM,NN,PPP.
Sets the page limit of the job currently printing to PPP pages. A value of zero means infinite.

PRNTSTAT[,NN].
Returns the status of the specified printer, or of all

printers if none is specified.

PRNTTEST,nn[,t[,C]].
Runs the printer test on the specified printer. The printer must be off and idle. If T is specified, only a single subtest is run. If C is specified, the subtest specified by T is run continuously, until the REW command is used.

PRU,nn,xxx[,yyy].
Sets the PRU limits on the specified printer. If yyy is not specified, the second PRU limit is set to zero. Any value greater than 65534 results in no limit.

REP,NN.
Causes the job on the specified printer to print an extra copy.

REW,NN.
Returns the job on the specified printer to the output queue and turns the printer off.

ROUTE,NN,R.
Routes the printer to print jobs from source R. This command also turns on full banner pages.

SKIP,NN[,MMM].
Skips the job forward MMM PRUs (default is 10).

SUP,NN.
Suppresses carriage control interpretation on the specified printer. The first column is printed.

USERFORM,NN.
Flags the printer as having user-supplied forms. A special flag in the accounting message dayfiled by MANAGER is set to circumvent charging the user for forms charges. This may be set before or during a job, and is turned off at the end of the job.

2.3.3 Operator commands.

The following commands are available at the operator's terminal. All preceding commands are available as well.

AUTOTEST,NNN.
Initiates the automatic PAL test sequence. The test is rerun every NNN minutes. The test may be stopped by using "SET,AUTOTEST,0".

BUS,X,{ON|OFF}.
Turns switchable bus X on or off. If OFF, all devices

connected through the bus are disconnected. If ON, all devices connected through the bus are re-initialized.

BUSIDLE,X.
Initiates a timed sequence of commands which will warn users connected through the bus that the bus is about to be turned off, and will then turn off the bus.

BUSSTAT.
Displays the status (either ON or OFF) of all switchable busses.

LOGINMSG,message.
Sets message as a temporary login message, which will be lost the next time MANAGER is initialized.

PALTEST,NN.
Initiates a test of the PAL and modem for socket NN. The socket must not be in use at the time.

SENDALL,message.
Sends the message to all terminals.

SENDBUS,X,message.
Sends the message to all terminals connected through switchable bus X.

SET,parm,value.
Sets the specified parameter to the specified value. The following are the parameters and their meanings:

```
AUTOTEST = 0 to stop the automatic PAL test
BLKMSU   = 1 to print a Gothic MSU on
             the banner page, or 0
INPINT   = 1 to ignore extra input interrupts
             in the PAL test, or 0
LOGINMAX = max number of dual MISTIC connections
MERIT    = 1 inhibits Merit network use, or 0
OPERMSG  = 1 to print I/O diagnostics at the
             operator's terminal, or 0
TRACE    = 1 to trace connections on the
             7/32 console, or 0
```

SOCK,NN,{ON|OFF}.
Turns socket NN on or off. If OFF, any user is disconnected. If ON, the socket is reinitialized.

2.3.4 Console commands.

The following commands are available on the 7/32 console. All preceding commands are available as well.

DATE,mmddyy.
Sets the current date as indicated.

DISP,NNNNN.
Displays the word at address NNNNN on the front panel. "%DISP,0" returns the display panel to the default address, showing the seconds counter and CPU utilization percentages.

MON,NN.
Monitors socket NN. This command sets the panel display to display the data word for the specified socket. The socket number is in decimal. The data word is 4 ASCII characters, of the form:

AABBCCDD
AA = auto-baud rate detect character
BB = last data character
CC = the last PAL input status.
DD = input interrupt count.

The address of the socket (in hex) is also printed at the console TTY. "%DISP,0" returns the display panel to the default mode.

TIME,HHMMSS.
Sets the current time as indicated.

3.0 System programming considerations.

3.1 Texts.

FREND requires 3 texts: FETEXT, FESYM, AND FEMAC.

FETEXT is a collection of macros and op-defs which define the operation codes for 7/32 assemblies. It is necessary for any assembly of 7/32 code. FETEXT works in conjunction with the ID732 pseudo-op in COMPASS, which allows 7/32 assemblies. The resulting assembly may be either absolute or relocatable.

FESYM contains symbol definitions for all FREND tables. In this regard, it can be thought of as SCPTEXT for FREND. It is more fully described in section 3.7.

FEMAC contains a collection of macros used throughout FREND. The most important macros are the task request macros. All tasks and parameters are defined in FEMAC. FEMAC also contains a collection of general purpose utility macros described in section 4.9.

All these texts are located as separate decks on the FE program library. No text needs any other text to assemble.

3.2 FREND program library.

FREND resides on its own program library. Each relocatable FREND deck is a separate update deck. When changing individual modules, it is generally not necessary to reassemble all of FREND. There is one common deck used by FREND which is also used by the texts, and therefore resides on FEPL. This is /LMBI, the definitions of the memory tables. This is used by FESYM and by INITIAL - therefore FEPL must be present as an XTEXT file whenever INITIAL is assembled.

3.3 FREND installation.

The following control card sequence will install FREND:

```
GETPL,FREND.
UPDATE,F.
RETURN,OLDPL.
GETPL,FE.  (XTEXT FILE)
RFL,64000.
AUTORFL,PART.
COMPASS,I,S=FETEXT,S=FEMAC,S=FESYM.
LDSET,HEXMAP=SBEX/MAPOUT.
LOAD,LGO.
NOGO,ABSBIN.
POST732,54.
```

The absolute FREND system now resides on file FESYS, and the load map on FILE MAPOUT.

3.4 Overlay structure.

FREND is divided into 3 overlays, a (0,0), a (1,0), and a (1,1). Since each overlay begins immediately after the preceding one, they are all simultaneously resident in the 7/32. However, the Cyber Loader requires only one overlay at a time to be core resident on the Cyber mainframe during the loading process. Therefore, this overlay structure dramatically reduces the core requirement for loading and linking FREND.

The FREND routines are divided into 3 overlays using the following criteria:

1. The overlays should be roughly the same size.
2. Routines which call each other should be together.
3. Only backward linkages are allowed between overlays, so routines called by everyone should be in the (0,0), while routines which call everyone should be in the (1,1).
4. Various variables used by many routines should be in the INST deck in the (0,0).

The FREND deck preceeds the (0,0). The FREND10 deck preceeds the (1,0). The FREND11 deck preceeds the (1,1) overlay.

3.5 POST732.

POST732 is a utility which reformats an absolute 7/32 core image into a form which can be loaded into the 7/32. In the reformatting process, the binary is rearranged so that each consecutive 12 bits contains a single 8-bit 7/32 byte. The resulting image can then be loaded into the 7/32 over the 7/32 DMA interface. POST732 makes the following changes in the overlays:

1. Each overlay is changed into a (0,0) overlay.
2. Each 54 table is padded out to exactly 16D words. This ensures that the 7/32 code is correctly aligned.

3.6 FREND installation.

The day "A" version of FREND was installed on 1/14/78 in LSD 46.00. This corresponded to FREND version 1.0.

The batch printer subsystem was installed in LSD 48.05/FEV 2.00 on 1/27/79. Extensive modifications were made to FREND. MANAGER, and 1FP.

3.7 FESYM - FREND symbol text.

3.7.1 FESYM macros.

FESYM table macros are used to create symbols for tables on the 7/32 system.

```
The following symbols are valid:
C.XXXXXX byte address of a 1 byte field
H.XXXXXX byte address of a halfword field
W.XXXXXX byte address of a fullword field
         (byte addresses are always the leftmost byte)
V.XXXXXX 1 byte mask for the specified field
S.XXXXXX number of rightmost bit in 1 byte field.
         (rightmost bit = 0)
Y.XXXXXX width of field less than 1 byte and greater
         than 1 bit.
J.XXXXXX bit number of leftmost bit, relative
         to the start of the previous half word.
         The leftmost bit = 0. These are
         used only by bits to be manipulated
         by SBIT, CBIT, TBIT, and RBIT.
```

The following macros are used to describe tables:

GROUP   defines a new table and establishes the prefix

CELL	establishes table reservation for bytes, halfwords, and fullwords.
FIELD	establishes table reservation for inidvdual fields within a cell.

3.7.2 <u>Use of FESYM macros.</u>

The FESYM macros are used to describe tables. They are used in the following fashion:

1. A GROUP macro defines the start of a table. A table name may be associated with the group. This specifies the characters to appear immediately to the right of the period for every symbol for this group. An additional prefix may be specified, which allows a leading letter to preceed all W. Symbols, for special uses such as pointers.

2. Each byte, word, or halfword within the table is described by the CELL macro. This macro also defines a symbol of the form C.XXX, H.XXXX, or W.XXXX. The CELL macro automatically aligns word and halfword fields to the proper table boundary.

3. Within a cell, individual fields may be described with the FIELD macro. This macro allows fields to be any width, so long as they stay within the previous cell. Fields are given C, H, W, V, S, and Y symbols. Any named field is automatically given C, V, and S symbols unless overridden by the explicit specification of symbols on the FIELD macro.

4. A table is ended with the ENDGROUP macro. This macro defines the length of the table. It also allows the end of the table to be aligned to a word or halfword before calculating the length.

3.7.3 <u>Symbol and coding conventions.</u>

1. The width of a fullword, halfword, or byte field is indicated by the first letter of the field (W, H, C)
2. Individual fields (using the FIELD macro) should generally only be described within byte fields. The S and V symbols are relative to a 1 byte field.
3. All fields (the FIELD macro) have a C, S, and V symbol generated for them unless only specific symbols are requested.
4. All bit flags to be manipulated with the FSET, FTEST,

FCLR, and FTOG macros should be described with a FIELD macro requesting H and J symbols. The FIELD macros should occur within a halfword cell.

4.0 Internal reference specifications.

4.1 FREND organization.

FREND consists of approximately 50 decks, arranged into 3 overlays. These decks contain tasks, interrupt service routines (ISR's), and general subroutines.

Each ident or deck is a relocatable module. All linkages between modules are accomplished through entry points. The modules are linked and relocated into an absolute core image by the CYBER loader.

In relocatable form, each module is a standard Cyber relocatable deck, where each Cyber word (60 bits) represents one 7/32 byte(8 bits). In actuality, each Cyber word contains 1 to 6 7/32 bytes, followed immediately by 0 to 5 Cyber words of zero, respectively. The format of the data word is: 4/BC, 56/data where BC is the 4-bit byte count, and data is the 7/32 data, right justified. After the CYBER loader forms an absolute core image, the POST732 program reformats this image into the 7/32 loadable format of 1 byte (8 bits) in each 12-bit Cyber byte. This core image is preceeded by a standard 77 table, and a 54 table where the FWA and length of the 7/32 core image are stored in the ECS image fields. The 7/32 core image immediately follows the 54 table.

This core-image file is loaded by MANAGER or LOAD732 over the Cyber mainframe-7/32 DMA interface.

Appendix A contains a list of all FREND routines.

4.2 Operating system structure.

FREND is an interrupt driven operating system. All interrupts are serviced by ISR's for each type of external device (PAL, line frequency clock (LFC), programmable interrupt clock (PIC) and the Cyber mainframe). ISR's always run in non-interruptable mode, using register set 0. Actions taken by the ISR's often generate requests for further processing. This processing, which is done in interruptable mode, is done by tasks. A task is a processing entity which is invoked by the FREND MONITOR. A task always runs in interruptable mode, in register set F. Once started, a task always runs to completion before another task begins. A task may request other tasks to do further work. These tasks may be requested immediately, or with a program specified delay.

All FREND processing is done either by ISR's or by tasks. MONITOR receives control after each task completes. It then finds the next task to be run, and enters it. When there are no more tasks to execute, FREND enters a wait state, with interrupts enabled. It remains in this state until an ISR requests a task.

4.3 Tasks.

The basic flow for the initiation and processing of a task is:
1. An ISR issues a REQTASK macro to request a task.
2. The REQTASK generates an SVC, which transfers control to the task request routine.
3. The task request routine moves the task request block onto the proper task request queue, and returns control to the ISR.
4. The ISR terminates, returning control to MONITOR.
5. MONITOR finds a non-empty task request queue, and removes the task request block from the queue. The task parameters are loaded into registers.
6. MONITOR jumps to the start of the requested task.
7. When the task is complete, it returns control to MONITOR. MONITOR then looks at the task request queues again.
8. If there are more tasks to process, steps 5 through 7 are repeated.
   These additional task requests may have come from the task just run, or from other ISR's which ran while the current task was running.
9. When there are no more tasks to execute, MONITOR enters a wait state. It will be reawakened only when an ISR completes.

Tasks are always requested through the REQTASK macro, which makes a supervisor call (SVC). The SVC immediately switches to register set 0, and invokes the SVC task request routine. The task request block (generated by the REQTASK macro), is entered into the appropriate task request stack (actually a 7/32 circular list). There are 3 such stacks: high priority, medium priority, and low priority. All tasks on the high priority stack will be processed before any tasks on the medium are processed. Likewise, all tasks on the medium priority stack will be processed before any tasks on the low priority stack. The following conventions should be followed for the task queues:
1. The high priority stack should be limited to tasks which are very time critical. An example is SKOTCL (socket output control) setting up the next output line to print - this must be done within one character time, or there will be a noticable pause in output.
2. Most tasks should go to the medium priority stack.
3. The low priority stack is used for non-time critical tasks.

4. Any task which recalls itself should recall itself to the low priority queue, or with delay.

The REQTASK macro generates a task request block. The block consists of a two word header, giving the task number, priority, parameter count, and ID. The parameters follow the header words, with one parameter per word. Parameters specified in registers are stored in the task block by the REQTASK macro, before the SVC. The block is copied verbatim into the proper task request stack.

By convention, a task is a closed subroutine, whose entry point is the task name suffixed by a pound sign ("#"). This limits task names to six characters. Tasks may have up to 4 parameters - these are defined for each task in FEMAC. When MONITOR starts a task, it removes the task request block from the request stack. The ID is placed in R4. and any parameters are placed in R5 - R8. The task is then entered by a BAL instruction (branch and link, RC is the return address). There are no specific exit conditions for tasks. Tasks may use all registers - tasks always run in register set F, with interrupts enabled.

Because a task is a closed subroutine, it may, under special circumstances, be called directly by another task as a subroutine. This happens during initialization. It also is done frequently for SOCMSG. Tasks should never be called as subroutines by ISRs, however, since they are not designed as re-entrant routines.

Delayed tasks are requested using the REQTASK macro, but by also specifying a DELAY parameter, giving the delay time in milliseconds. These tasks are entered into the timer queue, which is a linked list kept in order of expiration time. The list is maintained using the PIC. Whenever the PIC interval expires, all expired tasks are delinked from the chain and added to the appropriate task request stack. Because of additional overhead in the timer queue, delays of less than 100 milliseconds should be avoided.

Each task, with its required parameters, is defined in FEMAC using the DEFTASK macro. After a task is defined, it is requested using the REQTASK macro, specifying the correct parameters. Parameters are of the key-word type, and are order independent. If the correct parameters are not specified for a task, an assembly error results. An example of the REQTASK macro is:

```
REQTASK  TASK=SOCMSG,SOCKNUM=RA,MSGFWA=MESSAGE
```

Appendix B contains a list of all FREND tasks.

4.4 ISR (Interrupt Service Routine).

An ISR is automatically entered by FREND when an interrupt condition occurs. There are 2 basic types of interrupts: machine exception, and immediate. These are discussed in full in the 7/32 reference manual.

When an interrupt occurs, the 7/32 will switch to register set 0. disable interrupts, and enter the ISR at a predefined address (depending on the type of interrupt). If interrupts are disabled, the interrupt is queued until interrupts are reenabled. Note that ISR's always run in register set 0. Because this is also the register set used by SVC (REQTASK), care must be exercised when doing a REQTASK from within an ISR.

Exceptional condition: illegal instruction, machine malfunction (parity error), and divide fault. For each of these conditions, there is a PSW stored in low core (by INITIAL) which points to the ISR. The location of the PSW is fixed by the hardware.

Immediate interrupts are actually interrupts from I/O devices. There is a single I/O bus, and hence only 1 level of I/O interrupts. The priority of an interrupt from a device is determined solely by its position on the bus, not by its assigned bus address. The closer the device is to the 7/32, the higher its interrupt priority. When an immediate interrupt occurs, the 7/32 looks in core at address D0+(ad*2). where ad = the bus address of the device. At this address in core is a halfword which contains the address of the ISR. Because it is a halfword, the address must be 16 bits (and it must be even). If the address is odd. it is the address of a channel command block (CCB) used for automatic output to terminals. When the 7/32 enters the ISR. it presets registers

R0,R1 = old PSW
R2 = device number
R3 = device status

ISR's should always exit with the EXITINT macro. This macro clears the wait flag in the old PSW, ensuring that, if MONITOR was in wait state, it leaves wait state to process any task requests generated by the ISR.

Appendix C contains a list of all FREND ISR's.

4.5 Flow of control.

Once FREND is loaded, control is transferred to routine INITIAL, discussed below. After INITIAL has completed setting up the FREND tables, it transfers control to MONITOR. From this point forward, all processing is done either by tasks, or by ISR's.

4.5.1 Interactive subsystem.

4.5.1.1 Steady state condition.

In the steady state condition, is user is logged in and connected directly from his phone line to a socket. The socket is logically connected to a port. Within the 7/32, data flows from the socket to the port, and vice versa. Between the 7/32 and the Cyber mainframe, data flows from the port, over the DMA interface to 1FP. From 1FP, it flows to either MANAGER, ARGUS, or the user program. The interaction between 1FP and the 7/32 is described fully in section 4.19. The description to follow is a brief, step-by-step picture of data flow within the 7/32.

1. User enters a character.

2. The PAL interrupts the 7/32. PALISR receives control, reads the character from the PAL, and stuffs it into a buffer assigned to the socket. (a socket is always associated with a specific phone line).

3. Steps 1 and 2 repeat until the user enters the end-of-line character (usually a carriage return)

4. When PALISR detects an end-of-line, it completes the buffer assigned to the socket, and requests task SKINCL (socket input control), passing it the address of the buffer.

5. SKINCL analyzes the line in the buffer.

6. If the line starts with the FECC, SKINCL requests task FECSK (front-end command from socket), passing to it the address of the buffer. FECSK will then call COMMAND to process the front-end command. Based on the error reply from COMMAND, FECSK will return either an error message or a CR/LF to the socket. It does this by requesting task SOCMSG, passing to it the socket number and the message address.

7. If the line is simply data, SKINCL will call subroutine ADDPORT.

8. ADDPORT adds the address to the port circular input stack, and, if necessary, moves the address from the bottom buffer on the stack into word W.PTIN in the port.

8. SKINCL will then request task SENDCP to send an FP.INBS (inbut buffer status) message to the control port belonging to this socket.

9. SENDCP will look at the port input circular list and calculate the number of input lines in the port. It then builds an FP.INBS protocol record, inserting the input line count, and calls ADDPORT to add this record to the input circular stack for the control port to which the data port is assigned.

10. When MANAGER receives the FP.INBS record indicating that there is data on the 7/32 for this port, if the job is swapped out waiting for input, MANAGER calls MAN to free up the job.

11. The job will swap in and reissue the CIO read request. This causes 1FP to run. 1FP will read the W.PTIN word for this port on the 7/32. This word will contain a buffer address for a line of data. 1FP will then read the data buffer directly from the 7/32, and do any necessary translation before writing the data to the users job.

12. 1FP then writes a command to the 7/32 telling it that it just read some data, and interrupts the 7/32.

13. Routine ISR65 on the 7/32 will throw-away the buffer which was just read by 1FP, and refill W.PTIN with the next line from the port circular stack.

14. When the user enters the next line of data, the process begins at step 1. Note that a buffer is not assigned to the socket until the user enters the first character of a line.

For output from the Cyber mainframe to the 7/32 terminal, the flow is similar.

1. The user program makes a CIO request to write data to his program. This causes 1FP to be called to process the request. 1FP reads the address of an available buffer on the 7/32, and writes the data from the user program into the 7/32. (1FP does any necessary translation to ASCII).

2. 1FP writes a command to the 7/32 indicating the port to which the data belongs. It then interrupts the 7/32.

3. Routine ISR65 receives control. The address of the buffer containing the data is added to the top of the circular output stack for the port to which the data belongs. Task SKOTCL (socket output control) is then started.

4. SKOTCL ensures that no data is currently printing for the socket. It then removes the bottom entry from the port output circular stack, and sets up the CCB associated with the socket to print the data in the buffer. SKOTCL then starts the output to the socket.

5. The 7/32 microcode, through the CCB mechanism, automatically causes all characters in the buffer to be sent.

6. When the buffer is exhausted, an interrupt is generated and routine OUTISR gains control. This routine will release the buffer just printed, and then will request task SKOTCL again. The cycle then repeats from step 4.

7. It is possible for a user program to send commands to the 7/32. These are treated similarly to data records. However, SKOTCL recognizes that they are commands, and rather than printing them, it requests task FECPT (front-end command from port). FECPT will call COMMAND to process the command. It will then generate a protocol record to return the reply to the Cyber mainframe, and requests task MSGCP to send the protocol record.

8. Protocol records from control ports (MANAGER and ARGUS) are handled similarly to data records. However, routine ISR65 on the 7/32 recognizes that they are protocol records, and rather than starting task SKOTCL, it starts task CTLPT. This task processes all protocol records from the Cyber mainframe.

9. If the connection is port-to-port (MERIT inbound) rather than port-to-socket, ISR65 will start task PPIOCL, rather than SKOTCL.

The other complicated chain of events happens when a user dials in, or hangs up.

4.5.1.2 <u>Dial-in.</u>

(See also section 4.11)

1. Before a phone will answer, task SKINIT must run on the socket. This sets up many fields in the socket, and enables interrupts on the line. It also unbusies the line. SKINIT is run during initialization on all sockets.

2. When a user dials in, the PAL connected to the phone line will generate an interrupt. This causes control to transfer to PALISR. PALISR will sense that carrier is present, and will start task SKCARR with a delay of .25 seconds.

3. SKCARR runs, and statuses the PAL to ensure that carrier is still present. This is to guard against noise on the phone line.

4. For auto-baud rate lines, task SETBD is started with a 10 second delay. This is the auto-baud rate default timeout task.

5. If a character is entered before SETBD runs, PALISR will calculate the baud rate based on the received character. It will then start task SKOPEN. If no character is received before SETBD runs, SETBD will establish the default baud rate, and then start task SKOPEN.

6. SKOPEN does some additional setup on the socket, and requests "SOCMSG" tasks to return the system header and the login message to the socket. It then requests task OPENSP to open a connection to MANAGER.

7. OPENSP finds an available port, and links the socket and port together. It then links the data port to the control port for MANAGER (port PTN.MAN). Finally, OPENSP generates an FP.OPEN protocol record, and requests task MSGCP to place that record on the input circular stack for the MANAGER control port.

8. MANAGER will receive the FP.OPEN protocol record, and locate a free user table. It then associates the user table with the port, and returns an FP.ORSP protocol record to the 7/32, indicating that the open was accepted. MANAGER then starts LOGIN on the Cyber mainframe.

9. At this point, the job is running in the steady-state condition.

4.5.1.3 Hang-up or disconnect.

1. On a hangup or disconnect, the PAL interrupts the 7/32. PALISR gains control, and the PAL status indicates that carrier is not present. PALISR then starts task CLOFSK (close from socket).

2. CLOFSK will delink to socket from the port. It requests task SENDCP to send an FP.CLO protocol record to the 7/32.

3. CLOFSK then clears the socket, and disconnects and busies out the phone line. Task SKINIT is started with a five second delay (unless the bus through which the device is connected has been turned off, indicating that users should not be allowed into this socket again). After SKINIT runs, another user may dial in. Note that CLOFSK has not cleared or released the port, because MANAGER does not know that the user is disconnected until it receives the FP.CLO protocol record.

4. SENDCP places the FP.CLO protocol record on the input stack of the MANAGER control port.

5. MANAGER receives the FP.CLO, and starts LOGOUT on the user.

6. Once LOGOUT has completed, the mainframe will have no more activity with this user, so the port can be released. MANAGER sends an FP.CLO protocol record to the 7/32.

7. CTLPT will process the FP.CLO protocol record. It simply starts task CLOFPT (close from port).

8. CLOFPT will clear the fields in the port, and mark it as being available for use.

9. If the user has logged out without hanging up, the process starts at step 6. However, once CLOFPT runs, it finds that the port is still connected to the socket. Therefore, CLOFPT starts CLOFSK to disconnect the user.

10. CLOFSK will clear the socket, disconnect and busy out the line, and request SKINIT with a 5 second delay. Since CLOFPT has broken the connection between the socket and the port, CLOFSK does not worry about any data ports.

4.5.2 Batch printer subsystem.

4.5.2.1 Steady state condition.

In the steady state condition. a printer is connected and printing a job. The printer socket is logically connected to a port. The data flows from an output file on disk into MANAGER. MANAGER calls in 1FP (via CIO) to write the data over a channel and the DMA into a port in the 7/32. Within the 7/32, data flows from the port to the printer socket. The following is a brief summary of data flow for a printer within the 7/32.

1. Task PRINT calls subroutine NEXTLIN to get a line to print from the port. NEXTLIN discovers that there is no data in the port, and requests task SENDCP to transmit an FP.OTBS protocol record to the mainframe, and drops out.

2. Task SENDCP transmits the FP.OTBS protocol record to the mainframe over MANAGER's control port.

3. MANAGER sees the FP.OTBS, does a read from the output file on disk, and does a front-end block write request.

4. 1FP moves the data from MANAGER's field length into the 7/32 port connected with the printer, and interrupts the 7/32.

5. Interrupt routine ISR65 processes the interrupt from the mainframe, and requests task PRINT to process the data (the port is now full).

6. Task PRINT begins execution, ensures that the printer is not currently busy, and calls subroutine NEXTLIN, the processor for printer state PRNT (printing a job).

7. Subroutine NEXTLIN unpacks the next print line from the block data buffer into a line buffer, determines the character set of the data, and returns the buffer address to PRINT.

8. PRINT then calls subroutine STARTOT to start up the printing of the new line.

9. STARTOT sets up the CCB fields to point to the buffer of data to print and the correct translation table, and calls subroutine PRCCTL to

do carriage control processing.

10. PRCCTL translates the first character of the line into the correct printer function code, puts the function code into a buffer, and sets the CCB to point to the carriage control code buffer.

11. STARTOT then simulates an interrupt on the printer in order to start up the auto-driver channel, and then drops out.

12. The auto-driver channel outputs the contents of the carriage control buffer, and then upon hitting end-of-buffer invokes interrupt routine I#PRINT (in PRTISR).

13. I#PRINT sees that the carriage control buffer is spent, and returns it to the system and begins outputting the rest of the print line by setting the proper CCB fields.

14. The auto-driver channel outputs the entire contents of the print line buffer, and then upon hitting end-of-buffer once again invokes I#PRINT.

15. I#PRINT sees that the print line buffer is spent, returns it to the system, and requests task PRINT because there is no longer anything to print. The flow begins again at step 6.

4.5.2.2 Connecting the printer.

1. When the printer is off and idle, the socket is not connected to a port, and interrupts are disabled on the device. The printer state is OFF (printer off and idle).

2. When the system first comes up (or when the ON front-end command is issued), task OPENSP is requested to connect the socket to a port. The state is set to WTOP (wait for open).

3. Task OPENSP finds a free port, links it to the socket, and requests task MSGCP to send an open request to the mainframe.

4. Task MSGCP sends an FP.OPEN protocol to the mainframe over MANAGER's control port.

5. MANAGER sets up fields for the printer and sends an FP.ORSP open response back to the 7/32.

6. Task CTLPT sees the open response for the printer, sets the state to WTPR (wait for print job), and requests task GETPRT.

7. GETPRT requests task MSGCP to send an FP.GETO output file request to MANAGER, and sets state WTNP (wait for output request response).

8. Task CTLPT sees the negative FP.NEWPR response from MANAGER, sets state WTPR, and requests task GETPRT with a 2 second delay. Flow continues at step 7.

4.5.2.3 Starting a new print job.

1. GETPRT requests task MSGCP to send an FP.GETO output file request to MANAGER (as above).

2. CTLPT sees a positive FP.NEWPR response, moves information (such as job name and page limit) into the socket, sets state PREP (pre-print) and requests task PREPRT.

3. PREPRT sets up working-storage fields in the socket, enables interrupts on the printer, sets state BANR (printing banner pages), and requests task PRINT.

4. PRINT sees state BANR and calls subroutine BANR, the banner page processor.

5. BANR returns a new line of the banner page each time it is called, updating the banner page line number in the socket. Each line is returned in a buffer which is printed in the manner of the "steady state" example above.

6. BANR is finally called for the last line on the banner page, and either sets state PRNT if there is no dayfile to print (and we have reached the steady state), or sets state DAYF to process the dayfile.

7. PRINT calls subroutine DAYF for each line of the dayfile. State DAYF is much like state PRNT, with the major exception being that the data beginning in column two of each line is shoved over to the right side of the page by inserting a bunch of blanks.

8. DAYF sees that the print file has hit end-of-record (signifying the end of the dayfile)

and sets state PRNT (and we have hit the steady state example).

4.5.2.4 The end of a print job.

1. Task PRINT, running in state PRNT, sees that EOI has been hit on the print file, and sets state EOI.

2. Task PRINT calls subroutine EOIPROC. the processor for state EOI. If the copies count is non-zero, the state is set to COPY (which will print a message, rewind the output file, and set state PRNT again). If the copies count is zero, the state is set to ACCT (doing accounting).

3. Task PRINT calls subroutine ACCT, the processor for state ACCT. The subroutine formats an FP.ACCT accounting message (containing lines/pages print), sets state WTAC (wait for accounting response), and requests task MSGCP to send the message to MANAGER.

4. Task CTLPT sees the FP.ACCT response from MANAGER, moves fields from the response into the socket, sets state ACMS (accounting message), and requests task PRINT.

5. Task PRINT calls subroutine ACMS in state ACMS, which formats a message to print containing the accounting information, and sets state DONE.

6. Task PRINT calls subroutine DONE to process state DONE. If the printer is still logically ON, the state is set to WTPR and task GETPRT is requested. Flow continues in step 7 of "connecting a printer", above. If the printer has been turned logically OFF, flow continues in step 1 of "disconnecting a printer", below.

4.5.2.5 Disconnecting a printer.

1. The printer is disconnected either after finding itself logically OFF after finishing a print job, or after the REW front-end command is issued to rewind the job currently printing. In either case, we begin in state DONE.

2. Task PRINT calls subroutine DONE, the processor for state DONE. The subroutine sees that the

printer is logically OFF, and calls subroutine SENDCLO.

3. Subroutine SENDCLO requests task SENDCP to send an FP.CLO close request to MANAGER, and sets state WTCL (wait for close response).

4. MANAGER responds in kind with an FP.CLO close response, which is seen by task CTLPT. Task CTLPT requests task CLOFPT.

5. Task CLOFPT breaks the port-to-socket connection and requests task CLOFSK.

6. Task CLOFSK disables printer interrupts, resets the CCB and socket, and sets state OFF. All activity then ceases.

4.6 Connections and the ID.

A connection is simply a link between ports and sockets. Each socket has 2 fields to indicate other ports or sockets to which it may be connected. Each port has one field to indicate the other port or socket to which it can be connected. The connection number is the number of the port or socket to which the structure is connected. The connection type specifies whether the connection number is that of a port or socket. Connections are established by tasks OPENSP (socket to port) and OPENPP (port to port). There are currently no routines to open socket-to-socket connections.

The ID is a unique number which is associated with each connection chain, that is, with each unique user. When a user dials in (or a print job begins), an ID is generated and ASSIGNED to the socket for the communications line. Any port to which the user subsequently connects is assigned the same ID. Whenever a port and socket, or a port and a port, are connected, both sides of the connection must have the same ID. Most tasks which deal with two sides of a connection check to ensure that the ID'S AGREE. Any task which will deal with a specific port or socket usually is also called with the ID which belongs to that port or socket. The task then checks the ID in the table with the ID with which it was called, and generates an error if they disagree. This prevents a task from operating on a port or socket if the original owner has for some reason disappeared. It is especially important for delayed tasks, which may run quite a while after they were originally called (i.e. After plenty of time has transpired to allow the user to log out or hang up). The ID also provides a very important check on the internal consistancy of the operating system.

ID's are generated by OPENSP and OPENPP through the S=GETID supervisor call. The ID is a 15 bit number assigned sequentially and circularly starting with 4. The ID 1 is reserved for memory blocks assigned to the timer queue. and the ID 2 and 3 are reserved for memory blocks assigned to buffers. These are used by the memory MANAGER.

4.7 Initialization.

4.7.1 Transfer of control.

The FREND transfer address points to INITIAL. Therefore INITIAL receives control immediately after MANAGER loads FREND into the 7/32. At this point, INITIAL sets the initialization-complete flag (in the LMBI) to zero. It then performs all necessary initialization. At the completion of initialization, the initialization-complete flag is set to 1, signaling MANAGER that FREND is ready to run. INITIAL then transfers control to the FREND MONITOR. MANAGER waits 20 seconds for FREND initialization to complete. After that time. MANAGER will declare the 7/32 dead.

INITIAL sets up all FREND system tables and pointers. It also sets up the interrupt pointers.
The following tables are established:

1. Low core PSWS, locations 0 - CF.
2. All device interrupt addresses.
3. All CCBS (1/socket)
4. The task request stacks (low, med, and high)
5. The LMBI pointers (PW.XXXX)
6. The MISC table (date/time, version)
7. The FPCOM table (1FP/FREND communication)
8. The 3 buffer circular lists (80 char buffers, 240 character buffers, and the release list).
9. The bus status table.
10. All sockets.
11. All ports and the port circular buffer lists.
12. The memory allocation table and allocatable memory.
13. The timer queue (within allocatable memory).

Note that all 7/32 tables are built at initialization time by this routine. There are no assembled-in tables.

4.7.2 Device initialization.

INITIAL uses the device descriptions in the DEVICE deck. This deck describes all peripheral devices connected to the 7/32. For each device, INITIAL takes the appropriate action, as described below:

Bus Switch.
The bus switch reservation is requested.
If reservation is not granted within 5 seconds,
the bus will be marked as "off" in the bus status table,
and a message will be issued to the console teletype.

PAL (line adaptor).
For each PAL, a socket, CCB, and port is allocated.
The socket and CCB are permanently associated with the
device. The port becomes one of the pool of available
ports.

Console TTY.
Treated the same as a PAL.

Printer.
Treated the same as a PAL.

PIC and LFC (clocks).
ISR addresses are setup. The PIC is started when the
timer queue is initialized.

4.7.3 LMBI table creation.

The LMBIPT macro designates certain tables as fixed size.
These tables all occur
at the beginning of the LMBI, before any dynamic tables.
The PW pointers for these tables are constructed by INILMBI
directly from information in the LMBIPT macro. These
tables are:

- FPCOM (7/32 / 1FP intercommunications table).
- MISC (date/time and version).
- BF80, BF240, and BFREL (buffer lists).
- BANM (banner message).
- LOGM (login message).

Following these tables, INITIAL builds all dynamic tables.
Since these tables depend on each other, their order should
be maintained:

SOCK socket table. A socket entry is established for each entry in the device table of type TTY, PALLS, PALHS, PR96, or PR64.

DVSK device to socket index. This table is big enough for the maximum device number in the device table. Hence it is not built until the device table is scanned in its entirety. Routine INISOCK then plugs the socket number in, indexed by the device number.

PORT port table. There is one port for each socket, plus one for each DT.PORT entry in the device table. This table follows DVSK.

PTBUF port buffers. All port circular lists are carved out of this table. It is allocated by INIPORT as each port is initialized, and immediately follows the port table.

MALC memory allocation table. All remaining memory after above tables is divided into allocatable blocks by INIMEM. The MALC table contains 1 halfword for each allocatable block, indicating the availability and ownership of that block.

ALLOC allocatable memory. This immediately follows the MALC table, and consists of all core left after the above tables have been built. Each allocatable block is L.BF80 (84) bytes long. Blocks are used for buffers and the timer queue.

4.8 Open processing.

Open consists of the following tasks:
1. OPENSP - open socket to port.
2. OPENPP - open port to port.
3. OPENPS - open port to socket (not implemented).
4. OPENSS - open socket to socket (not implemented).

These tasks are responsible for establishing all FREND connections.

4.8.1 OPENSP - open socket to port.

This routine opens a connection to the mainframe for all incoming phone calls. When a carrier is established, SKOPEN is initiated. It starts OPENSP, which finds an available port, and links the socket to the port. If no port is found, a message is returned to the socket, and CLOFSK is started to disconnect the socket. OPENSP also opens a connection to ARGUS for the NETCNT command. It then starts SKOPEN to open a second connection for the socket (already connected to MANAGER) to ARGUS. If this connection can be opened, the MANAGER connection is pushed down to the second connection position in the socket, and the ARGUS connection becomes the primary connection. If the connection cannot be opened, a message is returned directly to the socket.

4.8.2 OPENPP - open port to port.

This routine opens a port to port connection. It is initiated by ARGUS for incoming MERIT connections. (It is also used by the stimulator to initiate simulated interactive users). ARGUS sends an FP.OPEN request over its control port. This causes the CTLPT task to initiate OPENPP. OPENPP will find 2 free data ports, and connect them together. It then returns an FP.ORSP response to ARGUS, indicating the port number on the ARGUS side of the connection. It sends an FP.OPEN request to MANAGER, indicating the port number on the MANAGER side.

4.8.3 Open errors.

If an open cannot be made, the error is always returned to the open initiator. For OPENSP, the open comes from the socket. Therefore, any open error returns a message to the socket (assumed to be a user terminal) and no open takes place.

For OPENPP, the open comes from a control port. Therefore, any open error returns an FP.ORSP (open response) to the originating control port, indicating the reason for the reject. (note that a successful open also returns an FP.ORSP. with a success code)

4.9 Close processing.

The CLOSE routine contains all close processing for various connections. This consists of the following tasks:
1. CLOFPT - close from port.
2. CLOFSK - close from socket (disconnect).

Each close task is concerned only with the specific side from which it was called. Thus CLOFPT only closes out the port side, while CLOFSK only closes out the socket side. These tasks follow the same general outline:
1. Check the ID to be sure it matches.
2. Break all connections by clearing the connection numbers in the port/socket and all port/sockets to which it is connected.
3. Reset the port or socket.

Additionally, it is necessary to allow for the following conditions:

1. A disconnect, which is a CLOFSK, should also cause the port to be closed out, since the connection is broken. This is done by having CLOFSK send an FP.CLO protocol record to the control port of any port to which the socket was connected. This starts a chain of events whereby MANAGER

logs out the user, and returns an FP.CLO request back to the 7/32. This causes the 7/32 to start a CLOFPT. which will zero out the port (which is no-longer connected).

2. A FP.CLO from the mainframe, which causes a port to be closed out, can also ask for the line to be disconnected. In this case, CLOFPT will initiate a CLOFSK task. CLOFSK then will eventually disconnect the user.

Disconnecting the line:
The line (socket) is disconnected (CLOFSK is started) if the user logs out and there are no secondary connections from his socket. The line is not disconnected if a LOGIN or NETCNT is rejected, and there is no secondary connection from his socket. A line is never disconnected if there is a secondary connection--it simply becomes the primary connection.
The following is the chain of events:

User disconnect:

1. PALISR calls CLOFSK.
2. CLOFSK breaks all connections.
3. CLOFSK sends FP.CLO to control port for all connected ports.
4. CLOFSK resets the socket and calls SKINIT with a five second delay.
5. MANAGER receives the FP.CLO. This causes it to put the job into LOGOUT.
6. When the LOGOUT completes, MANAGER sends an FP.CLO to the 7/32.
7. The 7/32 starts a CLOFPT to close out the port.
8. CLOFPT resets the port. Since the port is not connected to anything, CLOFPT ends.

User-initiated LOGOUT:

1. When LOGOUT finishes, MANAGER sends FP.CLO to the 7/32.
2. The 7/32 starts CLOFPT.
3. CLOFPT breaks the port connection and resets the port.
4. If the port was connected to a socket, CLOFPT starts CLOFSK.
5. CLOFSK closes out any additional socket connections as described above. Then it disconnects the user and resets the socket.
6. CLOFSK then calls SKINIT with a five second delay.
7. If the port was connected to another port, CLOFPT sends an FP.CLO to the associated control port.

Note - CLOFPT will recall itself once with a sufficient delay to allow any outstanding output to be sent to the socket. The SKSWOT flag in the socket is set to ensure that the output gets printed. Any output left after the delay is lost.

CLOFSK will recall itself once with an 8 second delay if all output at the socket has not been sent. After that time, the line is disconnected and any remaining output is discarded. This is necessary to prevent the socket and line from being tied up indefinitely.

4.10 Command processing.

FECMD is the interface between a port or socket and the front-end command processor (COMMAND). FECMD sets up and calls COMMAND, and then generates an appropriate error response.

There are two tasks for front end commands:

1. FECPT - front end command from a port.
   This task expects the first character of the buffer to be the start of the command. It returns any error indication by an FP.FCRPY protocol record to the control port.

2. FECSK - front end command from a socket.
   This task expects the first character of the buffer to be the the front end command character.
   It returns any error indication as a message directly back to the originating socket.

COMMAND contains all the actual command processors for the FREND commands. PRFEC is the routine which is called to process the command. This is called by 2 tasks:
  FECSK - front-end command from socket, and
  FECPT - front-end command from port.
PRFEC processes the command and returns an error code to the calling task, which either return an error code or an error message.

COMMAND calls upon several utility routines located in PARSER.
These are: GNELM - get next element from the command line,
        SEARCH - search a special table.
Also used are several of the utility routines in MISS.

GNELM extracts the "next" syntactical unit from a command. This may be a number, an alphanumeric string, a delimited character, or some other single character. GNELM returns the element's position and length, its type, and its value (e.g. The binary representation of a number.)

SEARCH quick sequential search of the standard

command table format genrated by the TABLE macro.
Returns the value associated with a given table entry.

These routines are designed to be called from background tasks, are fully interruptable, and use register set F.

Typical use:
When SKINCL detects a front-end command, it initiates FECSK to process it. This task begins by locating the beginning of the command, then calls SEARCH to find out if the command is defined, and if so, where the command processor is located. The command processor is invoked directly. The command processor parses the command, making repeated calls to GNELM to extract the individual elements. When alphanumeric parameters are encountered, the processor may call SEARCH to find their meaning. Other types of elements are accompanied by their meanings (values) on the return from GNELM.

4.11 Answering a phone line.

Answering a phone line is handled by a set of tasks in SKOPEN. These tasks are SKCARR, SETBD, and SKOPEN.

Simplified process:

1. Socket in state IN.IDLE. Phone rings, and task SKCARR initiates automatic baud rate detection.
2. Socket in state IN.AUTO. User types a character. and I#PAL sets baud rate, terminal type.
3. Task SKOPEN initiates connection to mainframe port.
4. Socket in state IN.IO, accepting input through PALMIN.

Detailed description:

To enable a PAL device so that we can accept incoming calls, INITIAL calls SKINIT. SKINIT is also called to re-enable a line after we hang a user up. SKINIT resets the socket to the idle state, and sends commands to the PAL device. These commands set parity off (for auto-baud detect), drop the busy signal on the phone line, and enable interrupts.

Once SKINIT has been called, a user can dial up and get a ring. The ring is automatically answered by the PAL device, because SKINIT sends a command to set data terminal ready. When the PAL device answers the ring, it brings up a carrier, which is answered by the user's modem.

There may be a great many interrupts from the PAL device during this process, as the phone rings and various relays in the DAA bounce around. We are only interested in the interrupt that tells us a carrier from a user is up, so we ignore all others.

Socket byte SKISTA contains the state value for input interrupt processing. At the time the phone rings, it is state IN.IDLE. In this state, interrupts are enabled but ignored, unless the carrier-off bit in the PAL status becomes clear, indicating a call may have been answered.

When V.SPCOFF goes clear, we want to make sure this is not just a momentary carrier-detect. The code at IDLE in I#PAL then disables further interrupts, sets state IN.WAIT, and requests the task SKCARR, with a 1/4 second delay.

In state IN.WAIT, interrupts should stop coming - any that occur are ignored.

1/4 second later, when SKCARR runs, it first checks to see if V.SPCOFF is still clear in the PAL status. If it is not, the carrier-detect was only momentary, and there is no valid call. In this case, SKCARR just resets the socket state to IN.IDLE, re-enables interrupts, and drops out. Back to square one.

If the carrier is still up after 1/4 second, we think we may have a call. The next step is to establish the baud rate of the terminal. If it is not an auto-baud socket, this is easy - SKCARR just sets the proper baud rate and calls SKOPEN. For auto-baud lines, SKCARR checks the "reverse channel receive" status bit from the device, which has been wired to indicate 1200-baud operation. If this line is up, it works just like a 1200 non-auto baud line. If this line is not up, SKCARR sets the state to IN.AUTO, and requests task SETBD with a 10-second delay. This task will set the rate to the default (300 baud) if the user does not type a character to set it first. (For fixed-rate lines, SKCARR calls SKOPEN, which will open the connection.)

State IN.AUTO is ended either by the user typing a character which determines the baud rate, or by SETBD after the timeout. In IN.AUTO, interrupts are processed by the code at AUTO. If the carrier drops during this time, the state goes to IN.OFF, and CLOFSK is run to reset the socket and busy out the phone line. Eventually SKINIT will run and re-enable the line.

When the user types a character to set his baud rate, AUTO reads the character and looks it up in a table, to see what baud rate, if any, is indicated. If the character does not define the baud rate, it is ignored.

When AUTO knows the user's baud rate, it calls SKINTTY to set the terminal attributes in the socket, then SKINCD to send the baud rate to this PAL device, and then SKOPEN to request the connection to the mainframe.

If SETBD runs before the user has typed his baud-rate character, it does the same, but always at 300 baud.

After the baud rate is determined, we are ready to connect the socket to a port. SKOPEN takes care of this. SKOPEN first checks to make sure the line is still up, (and starts CLOFSK if it is not) then it sets state IN.IO and sets the interrupt service routine vector to PALMIN. This is the only way a device gets to the normal I/O state. Finally, SKOPEN requests OPENSP, which will attempt to make the port connection.

Auto-baud rate determination:

(See also task SKCARR, which sets this state and requests task SETBD.) For baud-rate detection, the PAL device is pre-set (by subroutine SKINIT) to operate at 110 baud. The user's terminal, however, may be set to 110, 150 or 300 baud. When it sends CTRL-X or carriage return, the data will be received as a character which will uniquely determine the baud rate his terminal is set at. What the CPU receives was determined by experimentation, with a terminal set at different baud rates, and the result is the table T.BAUD. When indexed by the received character, this table gives the code for the indicated baud rate. Blank table entries contain the code BL.AUTO (zero), which indicates the data does not determine the baud rate, and should be ignored.

If the data received is greater than F0, it always means 300 baud - so the entries for data greater than F0 have been omitted from the table to save space.

4.12 Socket output processing.

4.12.1 SKOTCL - socket output control.

SKOTCL initiates all output to a socket. This involves setting up the device CCB, processing carriage controls, and fetching the next buffer to be printed. Buffers are removed from all connections feeding the socket.

SKOTCL has total responsibility for supervising the output pathway to a socket. It is called by anyone who wants to ensure that output gets sent to a socket. The most important calls are:

1. By ISR65, the mainframe ISR , whenever a line is added to a port which is connected to a socket. SKOTCL is only called if output is not active at that socket.

2. By SOCMSG, which adds messages to the socket output stack. SKOTCL is called whenever a message is added to the stack, and output is not active at the socket.

3. By OUTISR, the CCB interrupt service routine for socket output. SKOTCL is called whenever a buffer empties, and there is no data in the alternate buffer. This call to SKOTCL sets up the next line for output. Since it is time critical (the user is waiting) this call goes on the high-priority stack. Because the PALS have a 2 character buffer, there are 2 character times for SKOTCL to start up the CCB again. Otherwise, a noticable pause will occur.

4. By SKINCL, the socket input control routine, whenever it adds a line to the port or socket connection. This restarts any output which had been waiting because the user was typing in a line.

Control port data: SKOTCL will initiate SENDCP to send an FP.OTBS (output buffer status) message to a control port, whenever it removes a line of data from a port buffer, and there are 0, 1, or 2 lines of output remaining. This signals MANAGER or ARGUS that it can send more output.

General output philosophy: is done via a CCB. This CCB is setup by SKOTCL. All CCB interrupts are processed by OUTISR. These include end-of-buffer interrupts and translation table routines. All user output is sent in buffer 0 of the CCB. Any carriage controls are placed directly in the text buffer, overwriting any header information. CCB buffer 1 is used only to send delays for CR, LF, HT, VT, and FF. This buffer is setup by OUTISR on a translate table interrupt for one of these characters. SKOTCL does not start output until it determines that the terminal is ready to receive output.
Output is active if:
  the CCB is active (V.CBSACT is set)
Output can't be sent if:
  output is suspended (SKOSUP is set)
  user input is present (SKINCC non-zero)
  echo lines override the above considerations.

SKSWOT is a special flag, set by CLOFSK and CLOFPT, which forces output to be sent regardless of any input data or the output suspend flag. This ensures that all output gets sent before the phone is disconnected, and prevents a user from tying up a line indefinitely by simply suspending output. An echo line always overrides output suspended and user input present. The bottom line on the socket output stack is always checked to see if it is an echo line. An echo line is any line with the V.DHCECH flag set. Echo lines are added to the bottom of the socket output circular stack by:

PALISR - for input echo while output is printing.
RETIN - for retrieve of typed-ahead input.

SKOTCL is responsible for deciding which translate table will be used by the CCB. Normally the "transparent" table OUTXLT, which does not change any code values, is used. When sending

AF or OM data to a terminal in an alternate character set, however, a special table which is not transparent must be used. The address of this table is kept in the socket for any terminal which has an alternate character set selected.

4.12.2 OUTISR - output CCB processing.

OUTISR contains all interrupt processing for the socket output CCBS. All output to sockets (except immediate echo output) is sent using the 7/32 CCB (channel command block) mechanism. The channel command block is basically a channel program consisting of a description of the operation to be performed. and a list of parameters associated with the operation. In this case, the operation is always a write to a specific PAL device, and the parameters specify the buffer to use and the number of characters to send. The 7/32 autodriver channel automatically transmitts each character until the buffer is empty (or until specific characters are encountered), at which point an interrupt is generated and control passes to I#CCB (end of buffer) or a specific character translate routine. For a full description of the CCB, see chapter 7 of the INTERDATA 32 bit series reference manual.
CCB output processing routines consist of:

1. CCB ISR, which handles
   end of CCB buffer
   bad CCB device status
   immediate transfer (execute bit clear)

2. Output translate table routines.
   These handle transmission of delays for
   CR, LF, HT, VT, and FF.
   They also handle special processing for right margin.

3. INIOCT.
   This routine initializes the output translation
   table by inserting the addresses to handle the
   special characters CR, LF, HT, VT, and FF.

4. SETRM.
   This routine resets the CCB to stop at the current
   right margin setting. VCOL, the virtual column,
   is also set to the theoretical position the termianl
   carriage will be at when the CCB terminates.

5. DELAY.
   This routine sets up CCB buffer 1 to send the correct
   number of nulls, to do the delay after a CR or LF is sent.

6. RMFAKE.
   This subroutine subtracts one from the virtual column
   number (the projected carriage position at end of buffer)
   when a character is sent that causes no carriage motion.

7. DONPC.
This is the main routine for interpretation of non-printing characters. It calls CKNPC to decide whether the character must be interpreted, then calls NPCSTR to generate the string to be substituted, and places the interpretation into buffer 0, in place of the control character that is being interpreted.

The output CCB is always setup and initiated by SKOTCL. The CCB interrupt routines are invoked only for control characters, which may require delays, interpretation, or adjustments to the right margin. By manipulating the buffer byte count, an end-of-buffer interrupt is programmed to occur when the carriage reaches right margin, and at that time a CCB interrupt routine sends the CR and LF necessary to continue on the next line.

Note - since these routines are all run in non-interruptable mode, the primary coding consideration was speed of execution.

4.12.3 Right margin processing.

Caution - right margin processing is a complex and highly interdependent process - be sure you understand it fully before modifying it.

Special fields:
C.SKRM - the current right margin setting.
C.SKVCOL - the virtual column, which is the theoretical column position of the terminal when the CCB interrupts at end of buffer.
J.SKATRM - a flag which is set when the terminal carriage is setting at the right margin.
W.CBLWAO - the actual LWA of the output buffer, as distinguished from W.CBBOAD, which is the LWA of the buffer that the CCB is to print.

General mechanism:
Whenever a new line is started by SKOTCL, and whenever a CR is transmitted, SETRM is called to reset the CCB for the right margin. SETRM will set the CCB to terminate printing just at the right margin, assuming that all the characters in the CCB are printing characters (I.e, they cause carriage movement). Furthermore, VCOL is set to the column number at which the terminal will be sitting when the CCB terminates printing. When the CCB terminates, the address just printed is compared against the LWA of the output buffer. If they differ, then we assume we are only at the right margin. In this case, the CR.LF for the right margin return is inserted into the buffer

immediately in front of the next character to print, and the CCB is reset to resume printing.
Complications:

1. Imbedded non-printables.
   Each non-printable leaves the carriage one position left of what the originally calculated right-margin thinks it should be. To account for this, whenever a non-printable is sent, VCOL is decremented by 1. Thus, when the CCB terminates on end-of-buffer, VCOL indicates the exact column position of the carriage. If VCOL is not equal to the right margin, then a new right margin stop is calculated.
   Printing resumes without sending the CR,LF for the margin.

2. Non-printables after the margin.
   Once right margin is hit, we do not want to force a CR,LF unless the next character would print beyond the margin. Thus, if the next character after the margin is hit is a non-printable, the CCB is reset to transmit that character. However, since we are already at right margin, we want to leave VCOL=RM, so the SKATRM flag is set.

3. Partial lines and the $ carriage control.
   Because the terminal carriage can be at any position when a new line is started. VCOL is maintained across line boundaries. When a new line is started, the new right margin stop is always calculated relative to the current value of VCOL.

4. Right margin before the first character in the line.
   If the last line ended just at the right margin, and the next line is a continuation line, the CR,LF must be sent before the first character of the new line. This is done by first sending a null character, and faking a right-margin condition.

Key components:
1. SETRM - this is the routine which calculates where the CCB should stop, based on VCOL and RM.
2. RMFAKE - this routine adjusts VCOL down by 1 whenver a non-printable character is sent.
3. HITRM - this section of outisr is entered whenever the CCB interrrupts on end-of-buffer at the right margin.

4.12.4 Non-Printing Character Processing.

NPC interpretation is done entirely in the output ISR, for both output and echoed input. Input control characters are always echoed via a buffer, which allows this to happen.

When a control character is found in the output buffer, the translation done by the auto-driver channel causes control to go to one of the output translation routines in OUTISR. (NONPT, BSRM, CRDELAY. etc.) Each of these routines first calls DONPC, which may substitute a string of printable characters into the output buffer, and bypass normal control-character processing.

DONPC calls CKNPC to make the decision to interpret a control character. This is a complex decision, based on the following things:

1. Whether NPC is on or part (socket flag SKNPC).

2. Whether the CCB is sending a true output buffer or an input echo buffer (this affects which characters will be interpreted).

3. Whether the buffer contains an internal system message (we don't interpret these).

4. Whether the control character is a CR or LF for carriage conrol or right margin generation (these are never interpreted).

5. Whether the parity bit is set in an input echo buffer (this prevents interpretation).

If the character is not to be interpreted, the appropriate processing will be done by the translate routine (adujusting the right margin, generating delays, etc.)

If the character must be interpreted, DONPC calls NPCSTR to generate the string to substitute for it. This will be a single character, or a mnemonic enclosed in brackets, or a string like "<CTRL-X>".

DONPC places the string into buffer zero, overwriting the character being interpreted, as well as up to 4 characters before it. If the character is the first one in the buffer, the entire 4-byte data header will be overwritten. DONPC next calls SETRM, which will recompute the position of the carriage at end-of-buffer. and set up the CCB to interrupt at the right margin.

In the "CTRL" interpretation mode, the string to be substituted is 8 characters long. This is too big to substitute for characters 1-3 of a buffer, since the data header is only 4 bytes long. Therefore these strings are done in 2 parts. The first time the character is encountered, DONPC puts only the first four characters of the interpretation string in the buffer, and leaves the interpreted character in the buffer. It sets the "SKNPP2" flag. so that when this character is encountered again, after the first 4 characters have been sent, the last 4 will be sent. The last 4 characters are placed in the buffer so as to overwrite the interpreted character, so no further interrupts will be generated for this character.

After a string is placed in the buffer, an interrupt must be generated to get the auto-driver channel going again. This is done by requesting task "SINT". which simply executes a "simulate interrupt" instruction for the requested device.

4.13 Socket input processing.

PALISR consists of the interrupt service routines for the PAL input side. There are two ISR's: I#PAL - processes all interrupts for lines that are not in the normal I/O state -- either turned off. waiting for a carrier to come up, doing automatic baud rate detection, or disconnecting. PALMIN - processes interrupts for lines in the normal I/O state.

4.13.1 PALMIN - normal input.

This interrupt service routine is called when the socket input state (C.SKISTA) is IN.IO. We do it this way to avoid an extra load during the character interrupt, since this interrupt will burn large amounts of our CPU time. Thus when a character arrives, we don't have to check the input state to figure out what the interrupt is about.
In the normal case. this routine does the following:

1. Check status for a character present.
2. Read the character.
3. Translate the character.
4. Put the character in the buffer.
5. Increase the input character count.
6. Echo the character.

Special conditions are:

- bad status (no character)
- literal-next flag set for character
- special action characters
- buffer becomes full
- echo disabled

The code is written to pass straight through in the minimum time for the normal case. Special cases are handled in separate bits of code, following the main line.

4.13.2 Echoing of input.

If output is not active, input is echoed directly by simply writing the input data directly to the PAL.

If output is active, the data must be saved temporarily in a buffer. The method is:

1. If there is no echo buffer, then
   get an echo buffer,
   insert the character into the buffer,
   set the buffer address in SKECBF and add it
     to the bottom of the socket output stack.
2. If there is an echo buffer, but the buffer is not currently being printed by the CCB, then
   add the character to the buffer.
   increment the byte count in the record.
3. If there is an echo buffer, and the buffer is currently being printed by the CCB, then
   add the character to the buffer,
   increment W.CBLWAO (buffer LWA) in the CCB.
4. If the echo buffer is full (240 characters), clear W.SKECBF. This will cause a new buffer to be established for the next character.

Notes:

1. The echo buffer address is pointed to by W.SKECBF, but the buffer address is always in the CCB or on the socket output stack. W.SKECBF should never be used as an address of a buffer to purge - the address should be obtained only via the CCB or output stack.
2. OUTISR will clear W.SKECBF when it finishes printing the echo buffer.
3. The echo line is binary. Hence, RM and CC are not processed. This is necessary to ensure that echoed data exactly matches the input data. It also prevents interlock problems when SKOTCL modifies the CCB for carriage controls.
4. Control characters are always echoed via a buffer, to allow NPC processing to occur. If control characters are interpreted, this is done by OUTISR for both output and echoed input. The parity bit is used as a flag bit, to prevent interpretation of certain special echo characters, such as the CR that is echoed for the end-of-line function, and the backspace echo. PALISR clears the parity bit from control characters that it echoes via a buffer, and sets it for these special echo characters.

4.13.3 <u>Special input characters.</u>

Special action characters are indicated in the main translate table by setting bit 0 of the halfword entry. The rest of the entry contains the index into the socket translate table for the character. Loading the appropriate byte from the socket translate table gives the code for the control function for the special character. Finally, we index the SKXLATE table by the control function code, and load a halfword address of the control function processor to jump to.

"Break" is a framing error with a zero character. If the user is not in binary mode, we ignore it. For binary mode, we clear the binary input flag and simulate an escape, which will stop any tape input by MANAGER.

Note: when "break" is entered in binary input, the current line must be terminated normally. This line must be sent to the port before MANAGER receives the abort. Hence INESC is placed on the low priority stack, which causes it to run after SKINCL has placed the current input line on the port. The processing for CR is used to close off the current line.

4.13.4 SKINCL.

SKINCL processes a line once it is passed to it by the socket input ISR SKINCL is called only by the PAL's input ISR. It is called whenever the input ISR has received an end-of-line (CR), and has a fully assembled line. This line is passed to SKINCL, and is subsequently ignored by the ISR.. SKINCL decides how to process the line, based on its type. and the socket connection:

1. If the line begins with the front-end-command character (but the second character is not the same). SKINCL passes the line directly to FECSK, to process the command.

2. If the line had 2 consecutive command characters, the first one is deleted, and normal processing continues.

3. If the socket is connected to another socket, SKINCL passes the line to SOCMSG, which will send it directly to the connected socket.

4. If the line is connected to a port, SKINCL first checks for the special typeins *EOF, *EOP, *EOR, *EOS, *EORnn, and *EOSnn.

5. The line is then added to the port circular input list via the ADDPORT routine.

6. If the port is now full, "INPUT FULL" is returned to the socket via SOCMSG.
   If the port was full, the line is discarded, and "INPUT LINE DISCARDED" is returned to the socket.
   If the port is not full, but reader is on. only a DC3 is returned by sending it directly to the device.
   If the port is OK. SKOTCL is requested to wake up any output which was waiting for the input to finish.

Note - SKINCL is the only routine which handles complete input lines.

4.14 PPIOCL - port-to-port input/output.

Transfers of data on a port-to-port connection are needed whenever the mainframe sends a line to a port, or takes a line from one. PPIOCL is called in either event, and it always transfers data from the port it is called to to the other port of the connection.

Example: consider port A, which is connected to port B. When the mainframe sends a line to port A, it also sends a HEREIS command for that port, to the 7/32. When the 7/32 receives the HEREIS, it knows that port A has output data, and PPIOCL is requested at port A. PPIOCL will then transfer one line of data from port A to port B.

When the mainframe takes the line from port B, it sends an ITOOK command to the 7/32. The 7/32 then starts PPIOCL on port A. This call may result in no action, if port A is empty. However, it may be the case that port B has filled up with port A's output, and this task is needed to move more output from port A.

4.15 SKINIT - socket initialization.

SKINIT consists of a set of routines which are used to initialize various aspects of a socket. These routines are:

SKINIT: this is a task which is run at initialization time (by INITIAL), and 5 seconds after CLOFSK disconnects (and busies out) a line. It initializes the socket and PAL to answer a call (data terminal ready is raised) and sets up the socket default terminal type and associated defaults. The socket is placed into IN.IDLE, where it waits until carrier detect appears. Note: SKINIT¦ will ignore any socket which does not have its enable flag (SKSENB) set.

SKINTTY: this subroutine sets up the socket defaults for a specified terminal type. These defaults are currently parity, CR/LF/HT/VT/FF delays, and right margin.

SKINTLT: this subroutine sets up the socket translate table for special characters.

SKINCD: this subroutine sends commands to the PAL to setup the proper parity, character size, and baud rate, based on values in the socket.

SKINIT also contains 3 important tables:
1. SKTT - default values for each different terminal
   type.

2. TTABLS - the default terminal type for each terminal baud rate.
3. DVSK - the socket address for each PAL device, indexed by the device address.

It is started by CTLPT upon receipt of an FP.CPCLO protocol record.

4.16 Abort processing.

INESC contains 2 tasks which handle escape (abort) processing on a connection.

INESC is invoked by the socket input ISR (PALISR) when it receives an escape. INECSC will do the following:

1. Check the socket connection. If not connected to a port, it does nothing.
2. If connected to a port, all pending output in the port output stack is discarded unless the NTA (no-throw-away) flag is set. This makes ESC immediately abort all pending output.
3. An FP.ABT control port protocol record is sent to the associated control port. This tells MANAGER or ARGUS that the user has sent an ESC.

No input is discarded, since the ESC may have terminated autoline numbering or READPT, and the pending input is valid.

PPESC - port-to-port escape.
PPESC is invoked by the control port record precessor (CTLPT) when it receives an FP.ABT from ARGUS, which is an abort from a network user. PPESC does:
1. Check the ID of both ports.
2. Purge all NTA data in the port output stack.
3. Purge all NTA lines in the input side of the source port (which corresponds to the output side of the destination port)
4. Send an FP.ABT control port protocol record to the control port associated with the destination port.

4.17 Switchable Bus processing.

4.17.1 Initial set-up.

At initialization time, FREND grabs all the bus switch reservations it can. Busses which cannot be reserved are divided into two categories, switchable and non-switchable. Because a non-switchable bus which is not reserved at initialization time can never be assigned to this FREND, all devices connected through this bus are ignored. ("Ignoring" them, in this case, means setting their interrupt addresses to I#NODEV and their DVSK addresses to 0, and not creating any sockets for them.) A switchable bus, on the other hand, may later be assigned to this CPU even if it cannot be reserved now, so sockets are set up for devices connected through such a bus.

4.17.2 Turning a bus off.

To turn a bus off (using the BUS command), the bus is first marked as logically off by clearing C.BSON in the bus status table (BST). Then subroutine OFFDEV is called to deal with devices on the bus. For interactive sockets, a CLOFSK task is requested for printers, the FECSK task is used to simulate a "OFF,XX" command. The CLOFSK task will look at the bus and, seeing that it is off, will not start up a SKINIT task on the socket, so that phone lines will be left busy. Finally, task RELBUS is requested with a half-second delay.

RELBUS issues a non-fatal error message for all devices on this bus which still have the SKSACT (socket active) flag set. Then it drops the bus reservation and clears the "bus reserved" (C.BSRES) flag in the BST.

4.17.3 Turning a bus on.

Before doing anything else, the command processor checks C.BSRES to be sure that the bus reservation is not already held. Assuming the reservation is not held, subroutine GETBUS is called to try to get it. If the bus switch cannot be reserved (which probably means that another 7/32 holds the reservation) the command processor returns an error. If the bus is successfully reserved, GETBUS will set both the "bus reserved" and the "bus on" flags, C.BSRES and C.BSON.

Subroutine ONDEV requests tasks to initialize devices on the bus. SKINIT is requested for interactive sockets, and the FECSK task is used to simulate an "ON,XX" command for printers.

4.18 SENDCP - control port records to the mainframe.

SENDCP consists of 3 major portions:

1. SENDCP task, which issues a subset of FP. Messages to control ports.
   Allowable messages are INBS, OTBS, CLO, ABT, STAT.
   SENDCP constructs the FP.XXXX protocol record based on the current status of the port.

2. MSGCP task, which issues a specified buffer to the control port.
   The entire FP.XXXX protocol record, fully formatted, is supplied to MSGCP, who simply adds it to the control port.

3. ADDPORT, a subroutine for issuing any message to a port.
   ADDPORT is the only routine which should be used to add input lines to a port.
   It is used by SENDCP, MSGCP, SKINCL. PPIOCL.

4.18.1 FP.OTBS - need-data processing.

The "need-data" handshaking scheme between FREND and the mainframe is complex enough to warrant a separate explanation.

The FP.OTBS protocol record is used by FREND to tell MANAGER the state of an output buffer. It consists of a count of free buffer slots. When the number of <u>used</u> buffer slots goes below a certain level (the "threshold"). FREND will send the OTBS record to indicate to the mainframe that more data is needed. The mainframe will respond by sending the data via 1FP to the port.

It is important that this communication be synchronized--namely, that <u>only one</u> OTBS be sent per data transfer. If this is not the case, and an extra request for data is sent, the mainframe will fill the port in response to the first request, and then attempt to do it again in response to the second. In addition, no OTBS can be sent (or even formatted to be sent) while 1FP is actually filling the port, since the count of free entries is not going to be valid.

For these reasons, two flags were defined in the port:

- PTXFER - set when 1FP is transferring data into the port
- PTOTBS - set when a task has <u>requested</u> that an OTBS be sent

Let's look at an example of how this mess works. Suppose a job is printing on a front-end printer. Task PRINT, via subroutine NEXTLIN, is removing lines from the port and printing them. When the number of lines in the port is 2 or less (this is the threshold for printers), PRINT sets the PTOTBS flag (to indicate that a need-data is going to be sent for the port) and requests task SENDCP to send the OTBS. A short while later, PRINT again sees less than 2 lines in the port (it's printed another line), but sees the PTOTBS flag set so it does not request another SENDCP.

Meanwhile, MANAGER gets the OTBS, reads some more data from disk, and calls in 1FP to transfer it. 1FP, upon beginning the transfer, sets the PTXFER flag (to indicate transfer in progress) and clears the PTOTBS flag (to indicate that the OTBS has been responded to). If SENDCP runs during the data transfer, he will see the PTXFER flag set and will recall himself until the transfer is complete--this ensures that an invalid slot count is not sent.

There is one additional hitch to all of this--mainframe control port programs (such as MANAGER) have the ability to request an OTBS in order to determine the state of a port. These requests must be responded to, even if the port is above threshold. If the port is below threshold, a requested OTBS must be recognized as a need-data as well. All of this is handled in the following manner in SENDCP:

If the port is in a need-data condition, the OTBS is always sent (the PTOTBS flag has been set by the calling routine).

If the port is not in a need-data condition, the PTOTBS flag is cleared (since the OTBS will not result in a data transfer) and the OTBS is sent only if it was requested by the mainframe.

The following is a short summary of how this handshaking scheme is handled in each component of the system.

4.18.1.1 MANAGER.

MANAGER also knows the threshold levels for the various types of ports (connected to interactive sockets, printer sockets, and ports) and recognizes only OTBS's below threshold as requests for data.

4.18.1.2 1FP.

1FP clears PTOTBS and sets PTXFER when beginning a front-end transfer for write and rewrite orders and clears PTXFER upon completion of those orders. 1FP holds the PTNDIK interlock while manipulating these fields (as does FREND).

4.18.1.3 <u>SENDCP.</u>

SENDCP uses the following algorithm when called to send an OTBS record:

```
  if FREND low on buffers
    set PTWTBF in port
    clear PTOTBS
    drop out
  else (no low buffer)
    if PTXFER set
      recall
    else (PTXFER clear)
      if above threshold
        clear PTOTBS
        if mainframe request
          format/send OTBS
        else (not mainframe req)
          don't send OTBS
        endif
      else (below threshold)
        format/send OTBS
      endif
    endif
  endif
```

If the FREND system is low on allocatable buffers, SENDCP sets the PTWTBF flag (which causes routine BUFFER to send another OTBS when more buffers show up) and clears PTOTBS (so that SENDCP will run when called by BUFFER).

4.18.1.4 <u>SKOTCL/NEXTLIN (PRINT).</u>

When either of these routines sees that the port is low on data, they set PTOTBS, and then call SENDCP to send an OTBS only if PTOTBS was previously clear.

4.18.1.5 <u>CTLPT.</u>

CTLPT handles mainframe requests for OTBS's. When an OTBS request comes in from the mainframe, CTLPT will set the PTOTBS flag and request SENDCP to send the OTBS in any case, <u>except</u> when 1) the PTOTBS flag is clear, and 2) the port is above threshold. When these two conditions are true, we know that an OTBS has not yet been sent, but will be as soon as the output in the port is printed off.

CTLPT flags the OTBS request to SENDCP as a request from the mainframe by setting the V.EXTREQ flag in the request.

4.18.1.6 BUFFER.

In subroutine FREEPT, when BUFFER has determined that the system is no longer low on buffers and the PTWTBF flag was set in the port, the PTOTBS flag is set and SENDCP called to issue an OTBS when the port is below threshold and the PTOTBS flag was not previously set.

4.18.1.7 INESC.

In subroutine ZAPPTO, when INESC has emptied the port of output in response to an escape, the PTOTBS flag is set and SENDCP called to issue an OTBS if PTOTBS was not previously set.

4.18.1.8 PRINT (backspace processing).

When backspacing a printer, it is desired to finish printing the data in the port before actually beginning to backspace. FREND must wait for the port to empty, and he wants it to stay empty, so PRINT sets the PTOTBS flag without sending an OTBS--this causes all other routines which might request more data to cease and desist. After the port empties, PRINT does the backspace and then requests SENDCP to send the OTBS.

4.18.1.9 PPIOCL.

PPIOCL sets the PTOTBS flag and calls SENDCP to send an OTBS only if the port is below threshold and PTOTBS was not previously set.

4.19 Control port messages from the mainframe.

CTLPT processes all inbound (from the mainframe) messages on control ports. Each control port record is acted upon in an appropriate fashion.

This task is invoked by the device 5 (mainframe) ISR whenever output is received on a control port.

See ISR65 for a full description of the 1FP/FREND protocol.

Control port messages are sent from the mainframe routines MANAGER, ARGUS, and the stimulator. They are automatically transferred by 1FP.
ISR65 receives these protocol records and adds them to the output stack on the appropriate control port on the 7/32.
It then calls CTLPT for that control port.
CTLPT will process all records on the port output stack, and then exit.

4.20 Control port open/close.

CPCLO performs three distinct functions for control ports.

CPCLO - control port close.
This task closes each socket and port connected to a control port. For each connection, it sends a termination message, and then invokes CLOFPT to do the actual close.

CPACT - control port activity.
This task executes once each minute. If no activity occurred on the control port within the last minute, it marks the port as inactive, and sends a message to each socket specifying service interrupted. This task is initially started by CTLPT upon receipt of the first FP.CPOPN (control port open) for a control port.

CPOPEN - control port open. Whenever a control port is opened (after having been closed) CPOPEN will send a SERVICE RESUMED message to all sockets whose port is connected to the control port. This task is started by CTLPT upon receipt of an FP.CPOPN when the control port does not have the S65 (connected to mainframe) flag set. (CPACT is started at the same time, and S65 is set)

4.21 Sending messages to sockets.

SOCMSG: send a message to a socket. SOCMSG adds a message to the socket output stack. It is used by the 7/32 for sending all internal (i.e., originating within the 7/32) messages to the socket. Generally, these are front-end command error messages, and messages from OPEN (service not available), CPCLO (service terminated) and CPACT (service interrupted).

SOCMSG can be called as a task, or as a subroutine. The task call is preferred when only 1 or 2 messages are to be sent. The subroutine call is preferred when messages are to be sent to all lines (to prevent placing 100 SOCMSG tasks into the task stack).

Note that if the socket output stack is full, SOCMSG will throw the message away. This prevents a large number of SOCMSG tasks always in recall on the timer queue. The assumption here is that internally generated messages are so few that 5 positions on the output stack should be sufficient except in pathological conditions.

The one exception to this is socket 1 (the console) since this is a 110 baud device, and many messages go to it (all trace and error messages), SOCMSG will add messages to a 20 slot message stack. This stack is serviced by SKOTCL.

CONMSG: send a message to a connection.

CONMSG is similar to SOCMSG, but is used when the message is to be sent to the other end of a port connection. CONMSG is called specifying the port number. If the port is connected to a socket, CONMSG will simply call SOCMSG. If the port is connected to another port, CONMSG will try to add the message to the destination data port, and will recall itself indefinitely until the message is accepted.

4.22 General 7/32 interrupt processing.

ISRROUT contains all interrupt service routines for interrupts which indicate a hardware or software malfunction.

1. Arithmetic fault interrupt.
2. Machine malfunction interrupt (memory parity error, power failure, or 1FP killing the 7/32.)
3. Protect mode violation.
4. System queue interrupt.
5. Console interrupt. (this resets W#DISP, the front panel display address, and is the only interrupt which is not an error)
6. Illegal instruction. This generally is caused by a purposeful software crash.
7. Interrupt on a non-existant I/O device.

System crashes.

7/32 software crashes are caused by the CRASH macro, which simply generates an illegal instruction. This causes control to pass to the illegal instruction interrupt processor. which saves register set 0 at REG0SV and register set F at REG1SV. The PSW which was active when the crash occurred is then printed at the console teletype, and FREND hangs in a loop flashing DEAD on the front panel.

4.23 General 1FP/FREND protocol.

All interrupts from the 6500 are expected to have set FCMDTY in the FPCOM table to a command for ISR65 to process. Almost all 1FP/732 interchange is based on the FPCOM table. There are two basic sequences: 1. Read (1FP reads data from 7/32) 2. Write (1FP writes data to the 7/32)

4.23.1 Read (7/32 to mainframe).

On a read, the 7/32 places the address of a buffer containing data, into word W.PTIN of the associated data port. When 1FP sees this word non-zero, it requests the PTINIK interlock, which tells the 7/32 that it is using this word. It then reads the data from this buffer. When it is done, it sends an ITOOK command to the 7/32, specifying the port number. The 7/32 (ISR65) then releases the buffer pointed to by W.PTIN, and attempts to refill the PTIN word from the circular input list of the port. The last thing the 7/32 does is release the PTINIK interlock, which tells 1FP that it can look at W.PTIN again. If the buffer was read from a data port, task SENDCP is started to return a new FP.INBS (input buffer status) to the mainframe. This way the mainframe is always informed of the number of input lines waiting in the 7/32. This action is not necessary for control ports, since they are always automatically serviced by 1FP.

4.23.2 Write (mainframe to 7/32).

On a write, 1FP first interlocks the output side of the port by getting the PTOTIK interlock. It then checks H.PTOTNE, which is the count of available cells on the output circular stack. If non-zero, 1FP gets a 7/32 buffer by reading a buffer address from W.NBF80 (for an 80 character buffer), or W.NBF240 (for a 240 character buffer) in FPCOM. 1FP selects the smallest buffer which will hold the entire record to be written. 1FP then writes data to this buffer. Finally, it sends a HEREIS command to the 7/32 specifying the port number. This tells the 7/32 to refill NBF80/NBF240 with a new buffer, and to move the buffer whose address was in W.NBUF to the port circular output stack. H.PTOTIK is then released.

4.23.3 Interlocks.

There are 4 interlocks, all set by 1FP and cleared by the 7/32. These interlocks all have the same meaning:

1FP is altering or has altered the fields, and the only 7/32 routine which may process these fields is the mainframe ISR.

Thus, 1FP wil not process a field until it can get the interlock, and the 7/32 mainframe ISR ensures that all such interlocks have been set.

Interlocks:

PTINIK	port input interlock. Interlocks W.PTIN
PTOTIK	port output interlock. Interlocks W.PTOT, W.PTOTNE
NBUFIK	FPCOM next buffer interlock. Interlocks W.NBUF
FCMDIK	FPCOM command interlock. Interlocks W.FPCMD

4.23.4 Commands.

1FP can send 3 commands to the 7/32:
1. ITOOK - 1FP has read a buffer full of data
2. HI80 - 1FP wrote to the 80 character buffer (W.BF80)
3. HI240- 1FP wrote to the 240 character buffer (W.BF240)

All commands consist of the command ordinal, and the port number to which the command applies.

1FP will set the FCMDIK interlock before writing a command to W.FPCMD. This interlock is cleared by the 7/32 only when it has finished the command, and is ready for another one. Interlock clear = ready for command.

4.23.5 Buffers.

W.NBF80 is the address of an 80 character buffer for 1FP to write in. W.NBF240 is the address of a 240 character buffer for 1FP to write in.
1FP always sets the NBUFIK interlock before using 1 of these buffers. The 7/32 refils the buffer cell (W.NBUF) and clears the interlock.
Interlock clear = W.NBF80 and W.NBF240 each contain an available buffer.

Two different buffer sizes are provided for 1FP:
80 characters of data (W.NBF80)

240 characters of data (W.NBF240)
1FP always chooses the smallest buffer which will hold a complete line. It then writes to that buffer, and sends the appropriate hereis command to tell the 7/32 which buffer it used:

FC.HI80 - 1FP filled 80 character buffer
FC.HI240 - 1FP filled 240 character buffer.

In each case, the 7/32 replaces the buffer just filled with a new buffer of the same size.

4.24 The FREND MONITOR.

MONITOR is the main loop for any 7/32 CPU running in the front end system. It continually scans the task request stacks for any requests made by interrupt routines (or SVC routines), other tasks, or other CPU'S. When MONITOR finds a task to run, it sets up the task parameters and begins execution of that task. When the task completes, MONITOR begins again at the top of its loop.

MONITOR also performs other important functions each time through its main loop:

1. The console display panel is updated with the value of the word pointed to by W#DISP. This provides a dynamic core display.
2. Routine BUFFER is called to manage the buffer stacks.

If there are no tasks to execute, MONITOR enters a wait state by setting the wait bit in its PSW. Any interrupt will exit by clearing this wait flag, causing control to resume at the top of the MONITOR main loop.

4.25 Supervisor call routines.

There are 3 SVC (supervisor call) routines defined for FREND:

REQTSK - task request with no delay.
DLYREQ - task request with delay.
GETID - return a new ID number.

SVCROUT contains REQTSK and GETID.

REQTSK: this SVC is invoked by the REQTASK macro when no delay parameter is specified. The SVC instruction is followed by the task request block, as described in section 4.3. REQTSK moves this task block onto the proper task request stack, based on the requested priority (STK=HI, STK=MED, or STK=LOW).

GETID: this SVC simply returns the next ID number. The ID is a 15 bit number which increments sequentially on each GETID call. No attempt is made to handle the rollover condition in any special manner. Through this may result in duplicate ID numbers (after 32,762 calls), this should have little effect, since the ID is simply a double check for each task to ensure it is working on the correct user.

4.26 The timed request processor.

4.26.1 Delayed task requests.

Certain components of the FREND system need the ability to request background tasks which will not be executed until a specified interval has elapsed. To accomplish this, the DELAY parameter of the REQTASK macro is used, causing REQTASK to make a special supervisor call for a delayed task request.

This supervisor call routine (DLYREQ) computes the absolute time that the delayed task should be executed, and requests a task (TIMR) which will put the delayed request on the timer queue.

The timer queue is a list of delayed task requests, ordered by expiration time, so that the next delay to expire is the first entry on the list. TIMR simply sorts the new delayed request into the list.

The "precision interval clock" (PIC) is a device which can be ordered to interrupt the CPU after a given interval. The PIC is set to interrupt at or before the expiration time of the first delayed request on the list.

When the clock interrupts, a task called CKTIMR is run. This task removes any expired entries from the list, and enters them as immediate task requests on the request stacks.

4.26.2 PIC operation.

The precision interval clock (PIC) works as follows:

16 bits of data are sent to the clock's input register. Of these, the top 4 bits specify the resolution, or the rate at which the clock is to count down its interval. The resolution may be milliseconds, tenths of milliseconds, hundredths of milleseconds, or microseconds. The lower 12 bits sent to the

clock specify the interval to count. If the resolution is milliseconds, the clock can be set to count anything from 1 millisecond to 4.095 seconds.

At the command to start, the clock transfers the data which has been sent to its input register into its counting register. The counting register is then decremented at the resolution frequency.

When the clock's count reaches zero, the clock reads its input register (which may be unchanged from the last time), and begins counting a new interval. Whenever the clock reads its input register, it interrupts the CPU.

The clock's input register may be set at any time, without disturbing the on-going count. Thus the clock may be prepared for its next interval before the present one is complete - the clock may be set to interrupt at irregular intervals without ever having to be restarted.

At any time, the CPU can read the contents of the PIC's counting register, giving the count remaining until the next interrupt. Thus if the time of the next interrupt is known, the present time may be found by subtracting the value read from the clock.

FREND restarts the PIC whenever it interrupts. The clock resolution is always set at 1 millisecond, and the maximum interval is one quarter second. Thus if the next expiration time is 4.5 seconds away, the one quarter second interval will be sent to the clock. Eventually, there will be an interrupt when the next expiration is .5 seconds away. After this interrupt, the interval of .5 second is sent to the clock. The clock begins counting this interval immediately.

4.26.3 Delayed request queue.

The delayed request queue, or timer queue, is a linked list, with fixed-length entries, and forward links only. The list is terminated by an entry with a zero link field. The first word of an entry contains the link, which is the absolute address of the next entry. The second word is the expiration time - the value of the millisecond counter at the end of the delay. The actual task request begins in the third word. There is a maximum size for a task request, and a timer queue entry is that many words plus two.

The timer queue also has an empty chain, from which new entries are taken. When the length of this chain falls below a threshold level, a task is initiated (TIMMOR) which calls the memory MANAGER to assign a block of core, and extends the empty chain into it. The clock interrupt service routine (I#PIC) maintains a millisecond counter, in a sense. Since the clock does not interrupt each millisecond. but at irregular intervals, the millisecond counter (NEXTINT) is always set to the time it will be at the next clock interrupt. The present time, to the nearest millisecond, can always be obtained by reading the clock. and subtracting that from NEXTINT.

4.26.4 Unsolved problems.

1. Nothing has been done to prevent overflow of NEXTINT. If the clock resolution is 1 msec, NEXTINT will overflow after approximately 25 days of continuous front-end operation. This would seem to be enough - but if the resolution is set to .1 msec, NEXTINT will overflow after 2 1/2 days, which is conceivable.

2. When the empty chain gets down to a certain length, the TIMMOR task gets a block of core to expand it in. No provision has been made to return these blocks when the empty chain gets big enough.

4.26.5 FREND timer components.

The FREND timer mechanism consists of one interrupt service routine, one SVC routine, and a number of background tasks:

I#PIC interrupt service routine for the PIC. Updates the millisecond clock, and requests CKTIMR.

DLYREQ supervisor call routine to make a delayed request entry. Computes the absolute expiration time, moves the request to a new entry, and calls TIMR to sort the new entry in.

TIMR task to sort a new entry into the queue. After doing so, it causes a PIC interrupt if the new task was linked onto the front of the queue.

CKTIMR task to check and clean up the timer queue. Removes all expired entries and puts them on the task request stacks.

TIMMOR task to augment the empty chain. Gets a block of core, and links it in.

TIFIND subroutine to search the timer queue for the first entry expiring after a given time.

POPEXP subroutine to separate expired entries from the timer queue.

The following variables are maintained in core:

TIMHEAD points to the current timer queue head entry

TIEHEAD points to the current empty chain head

TINEMPT count of empty chain entries

NEXTINT the millisecond clock. This is not incremented every millisecond, since the clock does not interrupt that often. This cell contains the time it will be at the next interrupt. To get the real present time, you have to read the clock and subtract it from NEXTINT.

LASTCLK the interval last sent to the clock, without the resolution bits. This is the value in the PIC's input register at any time.

4.27 Trace.

Trace is a routine which can be called to issue a message to the console TTY tracing a specific event. The message is under control of the %SET,TRACE,{0|1} command.

Each trace call specifies a 4 character parameter, and 2 32 bit values. The final trace message issued is:

```
  HH:MM:SS: DD/MM/YY XXXX  AAAAAAAA  BBBBBBBB
         XXXX = 4 character parameter (event)
         AAAAAAAA = first value, in hex
         BBBBBBBB = second value, in hex
```

The following trace messages are issued by FREND:

```
BL      0000AAAA BBBBCCCC
issued when a low buffer condition is detected.
AAAA = number of free blocks in malc table.
BBBB = number of 80 character buffers.
CCCC = number of 240 character buffers.

BH      0000AAAA BBBBCCCC
issued when the low buffer condition clears, and
normal operation is resumed.
AAAA| = number of free blocks in malc table.
BBBB = number of 80 character buffers.
CCCC = number of 240 character buffers.

CFPT AAAAAAAA BBBBBBBB
close from port
AAAAAAAA = port number
BBBBBBBB = ID
```

CFSK AAAAAAAA BBBBCCCC
close from socket
AAAAAAAA = socket number
BBBB = device number
CCCC = ID

CPCL 0000AAAA 00000000
control port close.
AAAA = control port number.

CPOP 0000AAAA 00000000
control port open (FP.CPOPN from the mainframe)
AAAA = control port number.

OPSP AAAABBBB CCCCDDDD
OPSP = open socket to port.
~Aaaa = originating socket number
~bbbb = destination port number
~cccc = socket device number (corresponds to phone line)
~dddd = ID number

OPPP AAAABBBB CCCCDDDD
OPPP = open port to port.
~Aaaa = orininating data port number
~bbbb = destination data port number
~cccc = destination control port number
~dddd = ID number

4.28 Buffer management.

BUFFER manages the free buffer list and free buffer chain.

Buffer management must satisfy the following 2 needs:

1. Obtaining a new buffer must be very fast, since it is done by interrupt service routines.
2. Managed memory must be used for buffers. This gives the flexibility of using the managed memory area to allocate any number of buffers which are necessary.

The only way to satisfy requirement 1 is to maintain buffer addresses in a 7/32 circular list (delinking buffers from a chain is too time consuming).
The easiest way to satisfy requirement 2 is to maintain each managed memory block as a single buffer.

4.28.1 Free buffer lists.

There are three different free buffer lists: the free buffer circular list, the buffer release list, and managed memory.

The free buffer circular lists:

There are 2 lists - BF80 and BF240.
BF80 - this contains buffers which will hold up to 80 characters of data (they are actually L.BF80 characters long, which is 80+L.DTAHDR = 84).
These buffers are used for everything except:

- user input data
- mainframe output lines longer than 80 characters
- the register save area from ERROR

BF240 - this contains buffers which will hold up to 240 characters of data. (they are actually 3*L.BF80 characters long, which is 252. This is done so that they are an even multiple of smaller buffer blocks.)
240 character buffers are used so that user input and long output lines need not be split within the 7/32.

Each free buffer list is maintained at least 3/4 full by the BUFFER routine, which is called by MONITOR main loop.

Buffers are removed from these lists by the GET80 and GET240 macros. Buffers are never returned directly to this list. The PUTBUF macro returns buffers to the buffer release list.
Only the BUFFER routine adds buffers to these lists.

The buffer release list:

There is one buffer release list, PW.BFREL, for all buffers returned via the PUTBUF macro.
Buffers of both 80 and 240 characters are intermixed on this list. Every time through the MONITOR main loop, BUFFER is called. It removes buffers from this list and restores them to the proper free buffer list (BF80 or BF240). If the free buffer list is full, the buffer is returned to managed memory.

The managed memory area:

Each managed memory block is L.BF80 characters long. A small buffer is thus one block. A large buffer is three blocks. This gives the flexibility of keeping different size buffers without requiring a fixed number of each size.

Buffers are requested from managed memory to fill the free buffer lists to 3/4 full. Managed memory is never used to fill these lists more than 3/4 full. since additional buffers may be released from the BFREL list. Buffers are only returned to managed memory when the appropriate free buffer list is full.

80 character buffers have a memory ID of ID.BF80. 240 character buffers have an ID of ID.BF240. The ID is kept in the MALC table. This is the only indication that a buffer is 80 or 240 characters.

Why there are 2 different size buffers:

1. It is advantageous to have the maximum number of buffers possible on the 7/32 since this reduces overhead at the host machine, and provides maximal data throughput with no interruptions.
2. Each buffer holds only 1 line of data. This is done for ease of implementation of many features on the 7/32. 90 percent of all data can fit in 80 characters. Thus, it is desirable to have a large pool of small buffers.
3. The maximum input line is 240 characters, and it is much cleaner to have only one buffer for long input (and output) lines. Thus, buffers are needed to hold 240 characters.

4.28.2 Buffer full condition.

Since there are a finite number of buffers on the 7/32 it is necessary to handle the condition where almost all the buffers are in use. The 7/32 handles this by setting a buffer threshold flag. When 1FP sees this flag set, it will stop filling the 7/32 with output from the mainframe. Note that user input, and control-port stuff, will still fill buffers. Hopefully, the threshold will be sufficient to allow the 7/32 to function without running out of buffers completely.

1. Flag H.NOBUF in FPCOM is set non-zero when there are less than L.BUFOK free memory blocks. This will only be the case if all memory is already parcelled up into buffers.

2. When this flag is set, 1FP will not write any output to data ports on the 7/32, but will set flag H.PTWTBF in each port for which it has data. It will then suspend the output operation. (swapout)

3. When the number of free memory blocks rises above L.BUFOK, routine FREEPT will cause an FP.OTBS control port record to be sent for each port

which had the H.PTWTBF flag set (it also clears this flag). This protocol record will cause MANAGER/ARGUS to wake up any job waiting for output.

4.28.3 Error detection.

Each buffer which is on a circular list, but not in use, has its entire first word zero. Each buffer which is in use has its first word non-zero. Based on this, this following checks are made:

1. In RELBUF, each buffer removed from the BUFREL list must have its first word zero.
2. Each buffer added to the BF80 and BF240 lists has its first word zeroed. The GET80 and GET240 macros check this first word to ensure that it is zero.

Each 80 character buffer has an ID in the MALC table of ID.BF80. Each 240 character buffer has 3 consecutive blocks in MALC whose ID is ID.BF240. Based on this, the following checks are made:

1. In RELBUF, the buffer address must be a multiple of L.BF80.
2. In RELBUF, if the ID of the first block of a buffer is ID.BF240, then the ID of the next 2 blocks must also be ID.BF240.

4.29 Management of allocatable memory.

Overview:
In the LMBI of the front-end, there exists a portion of core memory which can be assigned, in small chunks of uniform size, to any logical entity (hereafter referred to as "caller",) who needs memory. A caller might be a connection, port, socket, or some such a caller is identified by a unique number. All changes in core assignment are accomplished by subroutines in this ident. MANAGE does the bookkeeping, maintaining a table which shows the current assignment of any block of allocatable memory.

Subroutines:

AQMEMRY this subroutine is called on behalf of a caller who needs memory. The caller's number and the number of bytes it requires must be specified. AQMEMRY finds a free area at least as big as the requirement, assigns it to the caller, and tells the caller where it is. If sufficient core is unavailable, AQMEMRY informs the caller, who must wait.

This routine is called by the MEMORY macro.

RQMEMRY this subroutine is called when a caller no longer needs its blocks of core, and it may be called in one of three ways:

1. With a specific FWA and length of an area to release - RQMEMRY will expect to release a contiguous area of core assigned to this caller.
2. With an unspecified FWA and length - RQMEMRY will release all memory belonging to the caller.
3. With an unspecified length, but specific FWA - RQMEMRY will return all core assigned to the caller following the FWA.

RQALL this routine releases all memory assignments, regardless of ownership. It should probably be called only during initialization.

Error checking done by these subroutines:

AQMEMRY - assign memory.

1. Ensures that the caller has a non-zero ID number, less than 10000.
2. Ensures that the amount of core required is greater than zero.

RQMEMRY - release memory.

1. Ensures that the ID number is non-zero and less than
SKOBIN Binary output mode. Set whenever the line currently printing in the CCB (buffer zero) is in the BI or AS character set. This bit inhibits right margin processing in the output ISR.
2. Ensures that the byte count is greater than zero.
3. Ensures that the FWA to release is positive.
4. If FWA=0, the byte count must also be zero.
5. If FWA is non-zero, the byte count must be an exact multiple of the block length, and FWA+byte count must be in core.

4.30 <u>FREND core layout.</u>

7/32 core is arranged in two separate segments: low core, and the LMBI (shared memory).

4.30.1 Low core.

Low core is memory exclusive to one CPU. It can be read and written directly by the Cyber channel interface. However, the interface cannot test-set any cells in low core.

The first 500 bytes of low core are reserved for certain hardware status words, as described in the 7/32 reference manaual. These include PSW storage areas and SVC address vector. Locations D0 through 2CF contain the ISR address for devices connected to the 7/32 I/O bus. Each consecutive halfword contains the ISR address for the next device address. Space is allocated for 256 devices. These cells are setup by INITIAL.

Locations 900 - 940 contain the powerfail - restart save area. Whenever the INI button on the 7/32 console is pressed, or when the mainframe halt-loads the 7/32, the current PSW is stored at 900, and the current registers are stored starting at 940. These addresses are defined as PFPSW and PFREG in FESYM.

The FREND core image begins at 0#988. (This is an arbitrary address - it must simply not overlap the ISR vectors and the powerfail save area). FREND is a contiguous core image, containing all executable code and local storage areas. It extends from 0#988 through about 0#12000.

Early in FREND is the set of channel command blocks (CCB). There is one CCB for each PAL and printer defined in the device definition deck. The CCB is used to send output to a device, and is described in the 7/32 reference manual and in section 4.24. CCBS are built by INITIAL. Because the CCB address is storred in the ISR vector at 0#D0, it is limited to 16 bits. This means that all CCB's must begin before 0#FFFF. Each CCB is 0#24 bytes long. Note that FREND maintains certain information at the end of each CCB which is not part of the standard 7/32 hardware supported CCB.

The Bus Status Table (BST) follows the CCBs, although it doesn't need to and could as well be in the LMBI. The table, which is also built by INITIAL, contains one entry for each bus. These entries can be changed as a result of commands which affect the bus.

Between the end of FREND and the last 2000 hex bytes of memory are the task request stacks (circular lists). There are 3 stacks: high priority, medium priority, and low priority. 25 percent of the task queue area is allocated to the high priority stack, 50 percent to the medium priority, and 25 percent to the low priority stack. Each stack is really a standard 7/32 circular list. Requests are added to the top of the list (by SVCROUT) and removed from the bottom of the list (by MONITOR). The lists are built by INITIAL.

The last 2000 hex bytes of low core are reserved as the 1FP dump area. Whenever 1FP finds the 7/32 has died, or when 1FP purposely kills the 7/32, it dumps itself to the 7/32 starting at the memory LWA - 2000. The dump is 6 PP bits into every 8 7/32 bits - one PPU word for each 7/32 halfword (2 bytes).

FREND is designed to work with at least four 32KB memory boards in low core. If additional boards are added, the 1FP dump address will automatically be readjusted.

4.30.2 LMBI (shared memory).

The LMBI can be read and written by two CPUS, and by the mainframe interface. The interface can also test-set cells in the LMBI. In the current FREND configuration, there is only one CPU, hence the LMBI is not shared. Almost all FREND tables are kept in the LMBI. The entire LMBI is built at initialization time by INITIAL. INITIAL determines the size of the LMBI - hence boards can be added or deleted as necessary.

The first area of the LMBI is the pointer area. This consists of a set of pointers to all other LMBI tables. The pointers are found in fixed areas assembled into FESYM. Each pointer has a symbol of the form PW.XXXXX. The pointer is 12 bytes long. The first 4 bytes (word) W.PWFWA, is the actual first-word address of the table. The next halfword, H.PWNE, is the number of entries in the table. The next halfword, H.PWLE, is the length of an entry. The remaining two halfwords have various meanings depending on the table involved.

All tables within the LMBI are located by a PW. Pointer. The PW table is setup by INITIAL. The /LMBI deck defines how many of the tables are allocated. Others are built by INITIAL based on available core and the device definition deck.
PW pointer table descriptions.

MISC — the miscellaneous table, containing the date, time, and version number.
LE is the total table length in bytes.
M1 is the H.INICMP flag. This flag is zero during initialization, and is set to 1 as soon as initialization completes, telling MANAGER that FREND is now ready to run. It is set to 2 when FREND crashes, telling 1FP that the 7/32 is dead.

FPCOM — the intercommunication area between FREND and 1FP.

BF80 — a circular list containing addresses of available 80 character data buffers. This list is maintained 3/4 full by the BUFFER routine.

The GET80 macro gets a buffer from this list.

BF240 a circular list containing addresses of available 240 character data buffers. This list is maintained 3/4 full by the BUFFER routine. The GET240 macro gets a buffer from this list.

BFREL a circular list containing the addresses of all buffers to be released. Addresses of both size buffers appear on this list. The BUFFER routine empties the list and returns the buffer addresses to BF80, BF240, or allocatable memory. The PUTBUF macro adds buffers to this list.

BANM the banner message. NE is the number of lines in the message. LE is the length of a single line. BANMES, the banner message set-up program, writes directly into this table, and its contents are copied onto each banner page.

LOGM the login message. NE.LOGM is the number of lines in the message, and LE.LOGM is the length of each line. FELOGM, a comdeck used by both LOGMES (the login message set-up program) and MANAGER, writes directly into this table. The login message is issued to each user immediately after the FREND system header.

SOCK the socket tables. NE is the total number of sockets. The first socket is referred to as socket 1. LE is the length of each socket. This table is built by INITIAL after it determines the number of sockets from the DEVICE deck.

DVSK each halfword in DVSK indicates the socket number for the corresponding device number.

PORT the port tables. NE is the total number of ports, LE is the length of each port table. The first port is referred to as port 1. This table is built by INITIAL based on the number of sockets and extra ports in the DEVICE deck.

PTBUF the circular lists for the port input and output stacks. Each list in this table is pointed to by PTINCL or PTOTCL in the port tables. LE is the total length of the table in bytes. This table is built by INITIAL, which sets up each circular list as empty.

MALC the memory allocation table. Each halfword corresponds to a memory block in ALLOC. If the halfword is zero, the corresponding memory block

is free. If non-zero, the memory block is in use, and the halfword contains an ID value indicating the owner of the block. This table is maintained by the MANAGE routine.
NE is the total number of blocks.
M1 is the total number of available blocks.

ALLOC allocatable memory. This table is divided into blocks of size LE (currently LE.BF80). The MALC table indicates whether a block in ALLOC is in use or available. NE indicates the total number of blocks. Allocatable memory is only used for the timer chain, and for buffers. One block is a small buffer, 3 consecutive blocks is a large buffer.
M1 is used by the MANAGE routine to aid in finding a free block.

4.31 Major data structures.

4.31.1 SOCKET.

A socket is a table associated with a PAL (or printer). Each PAL is associated with a dial-up or hard-wired communications line. The association between a specific socket and PAL is permanent. The socket table contains all information necessary for FREND to communicate with the device. All terminal attributes, such as parity, right margin, etc., are stored in the socket. In fact, most front-end commands alter fields within the socket.

Sockets are numbered sequentially beginning from one. Socket one is always the FREND console teletype. Socket two is always the operator teletype. The first phone line is socket three. Sockets are fully described in section 4.37, and in FESYM.

4.31.2 PORT.

A port is a table associated with a connection to the mainframe. A socket can be thought of as the description of the phone-line side of the 7/32, while the port describes the mainframe side of the 7/32. All data transfer between the mainframe and the 7/32 takes place using information contained in the port table.

The first three ports are known as control ports. They are permanently assigned to MANAGER, ARGUS, and the stimulator, respectively. Control ports are used to transfer protocol records between the 7/32 and the mainframe. These records indicate the state of various users on the mainframe and the 7/32. For example, a protocol record would say that a user just hung up, or just entered a line of data. Protocol records are fully described in section 4.40. A more up-to-date description will be found in FESYM.

All ports after the first three are known as data ports. Data ports are held in a pool. When a user dials in, a connection must be established between the user's socket and a data port. The first unassigned data port is linked to the socket by inserting the socket and port numbers in the port and socket. This establishes a "connection". Because the user will be under control of MANAGER or ARGUS, the data port is linked to a specific control port by inserting the control port number into the data port.

Data or protocol records are represented in the ports as buffer addresses kept on 7/32 circular lists, which are pointed to by fields in the port. Each port has a separate input and output circular list. These lists are manipulated only by the 7/32. When 1FP reads data for a data port, a field in the port gives 1FP the buffer address from which to read. When 1FP writes data for a port, it writes it to a free buffer on the 7/32. The 7/32 then adds this buffer address to the output circular list for the appropriate port.

4.31.3 Buffers.

All data and protocol records in the 7/32 are kept in buffers. There are two different size buffers, 84 characters (80 data characters plus 4 header bytes) and 252 bytes (248 data characters plus 4 header bytes). The official maximum line size is really 240 characters - the 252 byte size is greater than this, but is used because it is a multiple of 84.

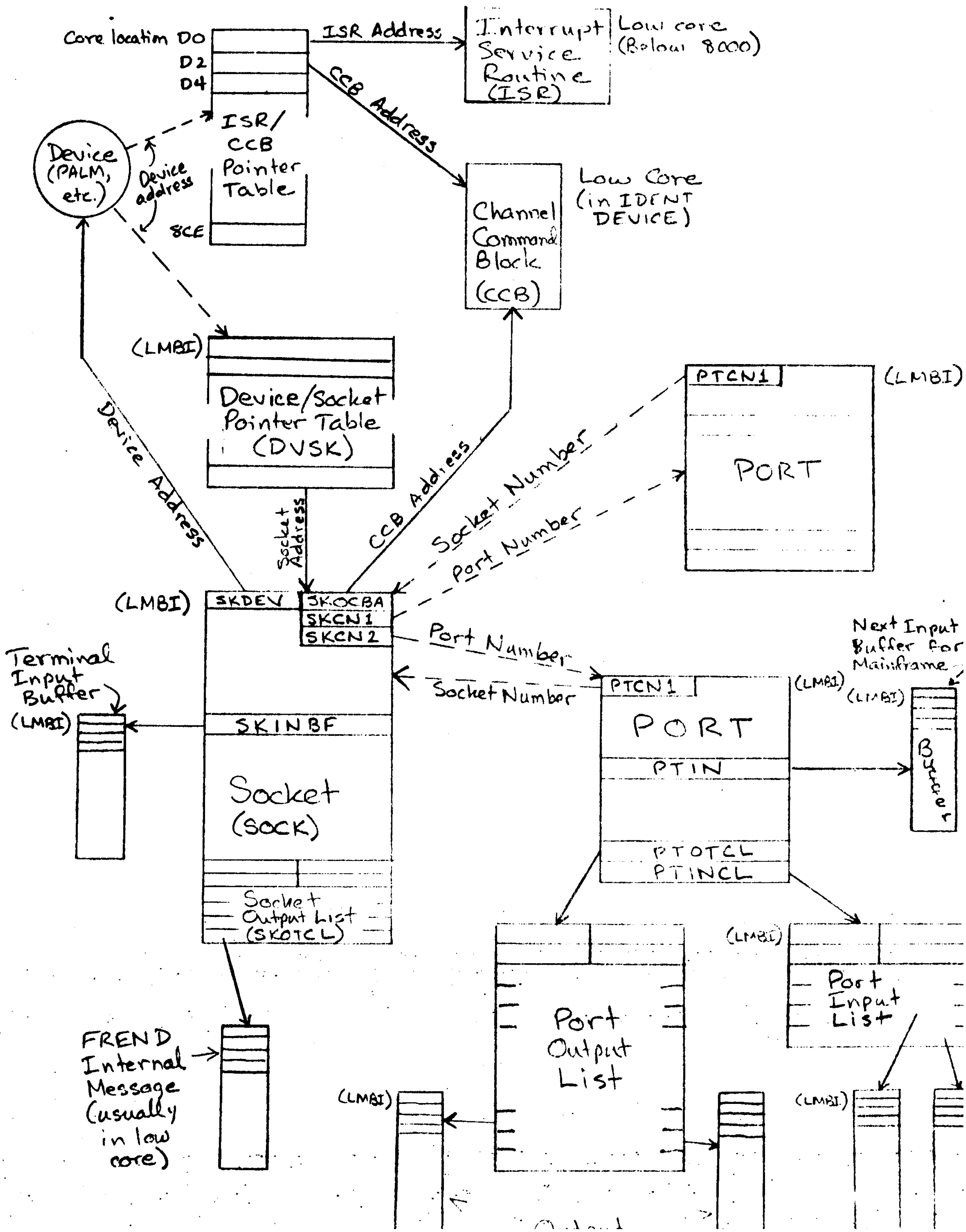
The two different size buffers are provided to allow maximal core utilization, while not requiring lines to be split over buffer boundaries. There are 3 circular lists used for buffers: BF80 is a list of 84 byte buffers. The GET80 macro is used to get a buffer from this list. BF240 is a list of 252 byte buffers. The GET240 macro is used to get a buffer from this list. BFREL is a list of released buffers. The PUTBUF macro is used to release a buffer to this list. Buffers are maintained on circular lists and managed memory, as explained in section 4.27.

The first 4 bytes of each buffer is a buffer header. This is explained in section 4.32.

4.32 FREND Table Relationships

The illustration accompanying this section shows the "pointer" relationships between the various tables used by FREND when servicing a connection. A solid line pointing from table A to table B indicates that table A has a field containing the absolute address of a table B entry. A dashed line indicates that the absolute table B entry address is found via an index, which is multiplied by the entry length and added to the base address of the table. For all LMBI tables, the base address, number of entries, and entry length are found in the pointer table (PW. symbols) at the beginning of the LMBI.

FREND Table Relationships



When a device interrupts the 7/32, it puts its address on the I/O bus. The 7/32 micro-code uses this address to index into the interrupt service vector, always located at 0#D0 in low core. This vector contains either the address of an interrupt service routine (ISR) or a channel command block (CCB). If it is an ISR, the 7/32 puts the device address in register 2, and starts the processor at the ISR address. The ISR uses the device address to index into the DVSK table in the LMBI, and loads the absolute address of the SOCKET table for that device.

The SOCKET may contain two "connection numbers", which are used as indices on the PORT table. The PORT entry contains absolute entries to the input and output circular lists, which contain addresses of buffers. (The circular list format is defined by the hardware - see the 32-Bit Series Reference Manual.) The output circular list queues output lines that have been sent from the mainframe, and are waiting to print. The input list is smaller, and queues typed-ahead input lines from a terminal.

The SOCKET also contains a pointer to the CCB, which is used to control the Auto-Driver Channel for output to terminals and printers. It also contains a pointer to the current input buffer for a terminal, as well as the Socket Output circular list, which queues messages for the terrminal generated internally by FREND. Note that unlike the other circular lists, the Socket Output list is contained within the Socket entry.

4.33 <u>FREND protocol records.</u>

Each protocol record to/from the 7/32 performs a specified function, as defined. The parameters associated with each record type are also described below.

Command-type protocol records are transferred only over control ports (ARGUS, MANAGER, stimulator).
Data-type protocol records are transferred only over data ports.
Formats of multiple-parameter protocol records.

FP.OPEN 8/PN, 8/OT, 8/OID, 8/DCP, 8/DID, 8/NH1, 8/NH2, 8/NCT1, 8/NCT2

- PN 7/32 data port number
- OT open originator type (OT.XX)
- OID ID supplied by open originator, (meant to be returned in ORSP)
- DCP destination control port (CTL.X)
- DID destination type (OT.X)
- NH1 1st character of network host name
- NH2 2nd character of network host name
- NCT1 1st character of connection type
- NCT2 2nd character of connection type

NH1,2, NCT1,2 are only present for FP.OPENS sent to ARGUS for NETCNT.

Sent to MANAGER/ARGUS by FREND to indicate a new user has just requested a connection to open. Sent from ARGUS to FREND to open a connection for an inbound network user. In this case, the data port number has no significance. FREND returns the data port assigned to the user in the FP.ORSP. In all cases, an FP.OPEN is acknowledged by an FP.ORSP with the same OID code.

FP.FCRP 8/PN, 8/code
PN = 7/32 data port number.
Code = front-end command reply code:
0 =ok
other values are EC.XXX error codes

returned by FREND to MANAGER to acknowledge a front end command. The command must have been sent to FREND over a data port from the mainframe.

FP.ORSP 8/PN, 8/ID, 8/OKR
PN = 7/32 data port number.
ID = ID from oid in open request.
OKR = 0 if open accepted,OPRJ.XX if rejected.

This acknowledges receipt of an open request, and indicates whether the open was successful or not. On a reject, the OPRJ.XX code indicates the reason for the reject. Sent by MANAGER and ARGUS to FREND to acknowledge an open request. Sent by FREND to ARGUS to acknowledge an open for an inbound network user.

FP.CLO 8/PN, 8/DIS
PN = 7/32 data port number.
DIS = 0 for no disconnect, 2 for a disconnect.

Requests a connection to be closed out. Sent by FREND to ARGUS or MANAGER to indicate a user disconnect. (ARGUS and MANAGER must send a CLO back to FREND to fully close out the connection.) Sent by ARGUS or MANAGER to FREND to request a connection be closed out. If DIS =2, and the user has no other connections, he will be hung up. Note that when sent from FREND, FP.CLO is informative, indicating a disconnect. When sent to FREND, FP.CLO is imperative, requesting the connection be closed.

FP.INBS 8/PN, 8/NUM
PN = 7/32 data port number.
NUM = number of input lines for this port in 7/32.

When sent from the Cyber mainframe to the 7/32,
"num" = 0 this causes an INBS to be returned from the 7/32 with "num" set correctly.
An INBS is also sent to the mainframe whenever a user enters a line, or the mainframe reads a line.

FP.OTBS 8/PN, 8/NUM
PN = 7/32 data port number.
NUM = number of empty output buffers in data port.

When sent from the Cyber mainframe to the 7/32,
"num" = 0 this causes an OTBS to be returned from the 7/32 with "num" set correctly.
An OTBS is also sent to the Cyber whenever the 7/32 prints a line, and there are 2 or less lines remaining in the port output stack.

FP.CPOPN

when sent to the 7/32 from the Cyber, indicates that the mainframe wishes to activate this control port. The record is returned verbatim to the Cyber as an acknowledgment.
A CPOPEN should be sent from MANAGER and ARGUS to FREND at least every 30 seconds to prevent FREND from declaring MANAGER or ARGUS as dead.

FP.CPCLO

when sent to the 7/32 from the Cyber, indicates that the mainframe wishes to close down this control port. The record is returned verbatim as an acknowledgment. This causes all ports connected to this control port to be disconnected, as if an FP.CLO had been sent out for all ports connected to the control port.

FP.TIME 8/0, 8/H,8/H,8/M,8/M,8/S,8/S,8/M,8/M,8/D,8/D,8/y,8/y
HHMMSS = hours, minutes, seconds.
Mmddyy = month, day, year.
Each character is in ASCII.

When sent to the 7/32 from the Cyber, the 7/32 moves the data into the current date and time which it maintains in the LMBI table area.

FP.CAN 8/PN
PN = 7/32 data port number.

Sent by MANAGER on an input-timeout. It causes the 7/32 to cancel the current input line and send backslashes to the terminal. Not effective on a port-to-port connection.

FP.INST 8/0, 8/TYPE, 32/VALUE
TYPE = instrumentation type (IT.XXX)

VALUE = instrumentation value

Sent by MANAGER to initiate and terminate the transfer of instrumentation data.
Sent by the 7/32 with instrumentation data for MANAGER.

FP.GETO 8/PN, 8/SOURCE, 16/PRU1, 16/PRU2
SOURCE = display code source
PRU1. PRU2 = primary/secondary PRU limits

Sent by the 7/32 to request a job to print from the specified source. MANAGER will respond with FP.NEWPR.

FP.NEWPR 8/PN, 8/DFP, 16/PRUS, 7*8/N, 8/COPIES, 16/PAGELIM
DFP = non-zero if dayfile present
PRUS = PRU size of job
NNNNNNN = job name
COPIES = copies count
PAGELIM = page limit

sent by MANAGER in response to FP.GETO. IF A JOB fitting the description in the FP.GETO is found, all fields are filled in. If there is no such job, all fields are returned zero.

FP.ENDJ 8/PN
Sent by FREND to MANAGER for the "END" command, and for jobs that exceed page limit. Returned by MANAGER in response.

FP.EOI 8/PN
Sent by MANAGER at end-of-information on a print file. Causes FREND to set the PTPEOI flag.

FP.SKB 8/PN, 16/COUNT
Sent by FREND to MANAGER to skip a print file backwards COUNT prus.

FP.SKIP 8/PN, 16/COUNT
Sent by FREND to MANAGER to skip a print file forward COUNT prus.

FP.ACCT 8/PN, 8/OWN, 16/PAGES, 32/LINES
OWN non-zero if user supplied own forms
PAGES = pages print count
LINES = lines print count

Sent by FREND to MANAGER at the end of a print. Causes MANAGER to dayfile the print charges and return an FP.ACCT.

FP.ACCT 8/PN, 8/RG, 24/AMOUNT
RG = rate group of job just printed
AMOUNT = print charge, in pennies

Sent by MANAGER to FREND in response to an FP.ACCT. Causes FREND to print the print cost line on the end of the job.

FP.COPY 8/PN
Sent by FREND to MANAGER to make MANAGER restart the print job from the beginning (a new copy).

Formats of data-port protocol records:

FP.EOR text =level number as entered by user, either null, or 1 or 2 level numbers in ASCII (not binary). Also sent from 1FP to FREND to indicate end-of-record on block-transfer files.

FP.EOF No text. Sent from FREND to the mainframe when the user enters "*EOF". SENT FROM 1FP TO FREND TO INDICATE end-of-file on block-transfer files.

FP.UNLK text =normal prompt, in ASCII.

FP.FEC text =front end command, in ASCII.

FP.BLDAT Sent from 1FP to FREND to indicate a block data buffer. The data is packed into 240 character buffers with appropriately imbedded end-of-line bytes.

4.34 PALS Test

The PALS test is an on-line test which can run on any asynchronous telephone line that is not otherwise in use. Only one line can be tested at a time. The PALS test is invoked by the %PALTEST,socknum command. Through the %AUTOTEST command, it can be programmed to sequence through all idle phone lines at regular intervals. In automatic mode, only lines with nonzero phone numbers (the SKPNUM field) in the socket are tested.

The test works by raising the BSY command line to the PALS device. If the device is connected to a VADIC modem, the modem will go into analog loop-back mode, and echo on the input side any data that is transmitted to it. For this reason, PALTEST will fail if the device is not connected to a modem.

The test is driven by two interrupt service routines - one for input, and one for output. The input ISR begins by sending 8 synchronization characters (FF), followed by 255 data characters: 01, 02, 03, . . . ,FD, FE, FF. The input ISR ignores any bad-status or busy-status interrupts that may occur as the test begins. It waits for at least one synch character (FF) with a status of 00 or 10. After that, the first non-synch character must be 01, 02, 03, etc.

The test will fail if nonzero status appears on the transmit side, or if any status other than 00 or 10 appears on the receive side. Some modems which operate satisfactorily in production generate a large number of receive-side interrupts with a status of 08 or 18 (busy). A test option causes the test to ignore these interrupts. The command, "%SET,INPINT.1" will cause these modems to pass. "%SET,INPINT,0" does the reverse.

4.35 Printer Format Tape Specifications

The standard format tape at MSU is punched as follows:

Channel 1: Top of form

Channel 2: Next 1/2 page

Channel 3: Next 1/3 page (6 lpi only)

Channel 4: Next 1/4 page

Channel 5: Bottom of physical page

Channel 6: Next 1/6 page (6 lpi only)

Channel 7: Next 1/3 page (8 lpi only)

Channel 8: next 1/6 page (8 lpi only)

Channel 9: Unused

Channel 10: Unused

Channel 11: Unused

Channel 12: Last line of form

At 6 lines per inch density, there are 66 lines on an 11-inch page. At 8 lines per inch, there are 88 lines. Because of the difference in spacing, there is not a perfect correspondence of page position for each line in the two densities. A "point of coincidence" is a vertical position on the page where a line occurs at both densities - these points are spaced at 1/2-inch intervals. Line 4 at 6 lpi is a point of coincidence, which corresponds to line 5 at 8 lpi. The next point is line 7 (6 lpi) or line 9 (lpi). In order for a punched hole on a format tape to work in both densities, it must be punched on a point of coincidence.

There are two peculiarities in the standard format tape that arise from the "point of coincidence" problem:

1. Channel 5 (Bottom of physical page) should, strictly speaking, be punched in line 66 (6 lpi), but since this is

not a point of coincidence, the hole is punched in the nearest one, which is line 1. Thus channel 5 is not actually the top of the physical page, not the bottom. For this reason, the "5" carriage control does not perform as advertised, but sends the printer to line one of the next page, instead of the bottom. The printer can be aligned so that line one is on the page break.

2. The 1/6, 1/3, 2/3, and 5/6 points on the 11-inch page do not occur at points of coincidence. For this reason, the standard tape uses two channels each for next-1/3 and next-1/6 carriage controls. Channels 3 and 6 are used if the density is 6 lpi, and channels 7 and 8 are used at 8 lpi. The printer software keeps track of the density as it changes, and uses the appropriate tape channel for the 3, 6, I and F carriage controls.

By channels, the standard tape is punched as follows (the line numbers are 6 lpi numbers, unless otherwise shown):

Channel	Meaning	Lines Punched
1	Top of form	4
2	Next 1/2 page	4, 34
3	Next 1/3 page (6 lpi)	4, 24, 44
4	Next 1/4 page	4, 19, 34, 49
5	Bottom of physical page	1
6	Next 1/6 page (6 lpi)	4, 14, 24, 34, 44. 54
7	Next 1/3 page (8 lpi)	5, 32, 59 (8 lpi)
8	Next 1/6 page (8 lpi)	5. 18. 32, 45, 59, 72 (8 lpi)
9	Unused	4
10	Unused	4
11	Unused	4
12	Last line of form	64

Non-standard carriage control tapes may be punched in any fashion, as long as a few rules are observed:

1. Every channel must be punched at least once, at a point of coincidence, to prevent paper from running away when a carriage control character references an unused channel.

2. Channels 1 and 12 must always be punched for top- and bottom-of-form, respecively, since the printer hardware uses these channels for its auto-page-eject function.

For any format tape, standard or non-standard, the proper carriage control character for a given channel may be determined by the following table:

Pre-print Character	VFU Channel	Post-print Character
1	1	H
2	2	I
3	3 (6 lpi)	J
3	7 (8 lpi)	J
4	4	D
5	5	E
6	6 (6 lpi)	F
6	8 (8 lpi)	F
7	12	G

Pre-print characters cause the page to skip to the indicated channel before the line is printed. Post-print characters cause the skip to be one after the line is printed.

Lower-case alphabetic carriage-controls are recognized the same as their upper-case equivalents.

5.0 Operator communications and procedures.

A full description of all applicable operator communications and procedures can be found in the FREND SYSTEMS OPERATOR GUIDE.

5.1 FREND loading.

FREND is loaded automatically by MANAGER. No operator intervention is required. MANAGER automatically selects the proper version of FREND, as explained in 6SM 135.

5.2 FREND dumping.

FREND is automatically dumped by MANAGER whenever 1FP detects that FREND is no longer running. This causes MANAGER to abort, after which DUMPFE is automatically run. FREND is also automatically dumped at each deadstart. This is done in case the previous crash was caused by 1FP, in which case no 7/32 dump is possible. In the event of a parity error while dumping, DUMPFE will pause and request the operator to disable parity checking on the 7/32 interface. After this is done, the operator should give DUMPFE a GO, at which time the dump will be retried.

5.3 SENDALL.

A message may be sent to all front-end users even though the mainframe is not operational. This is generally used to give users information on mainframe problems. The SENDALL command is used for this. The format is:

%SENDALL, message.

5.4 SENDBUS and BUSIDLE.

A message may be sent to all front-end users connected through a particular bus. This is generally used to warn users that the bus is about to be turned off. The format of the SENDBUS command is:

%SENDBUS,X,message.

Where X is the identifier of the switchable bus.

The BUSIDLE command will automatically send a pre-formatted series of messages to warn users that the bus is being shut off, and it will follow these with the command to turn the bus off. The format for this command is:

%BUSIDLE,X.

Where, as before, X is the identifier of the switchable bus.

5.5 FREND console commands.

Several FREND commands are valid only from the 7/32 console teletype. In general, these are of use only to systems programmers. They are described in detail in the FREND OPERATOR GUIDE. There are also a set of commands for detecting and dealing with line problems, also described in the operator guide.

6.0 User aspects.

The major user benefits from the front-end system are the increased flexibility and capability for support of various terminal types and attributes. Also of major importance is the support of long input lines, and of full binary input/output. These features are enumerated in sections 2.2 through 2.7.

7.0 System file changes.

Because FREND is a separate entity from the mainframe, there are no changes to the system files on the mainframe.

8.0 References.

SMP 28 - MSU front-end.
SMP 49 - MSU front-end, phase 1.
SMP 60 - Front-end command and control.
6SM 94.1 - banner messages.
6SM 131 - MERIT interactive support.
6SM 134 - 1FP/CP2TT.
6SM 135 - MANAGER and frends.
FREND OPERATOR GUIDE.
INTERDATA 32-BIT SERIES REFERENCE MANUAL.
MODEL 7/32 REFERENCE MANUAL.
FREND BATCH PRINTER OPERATOR GUIDE.
BATCH PRINTER HARDWARE GUIDE.

This document was originally written by Robert F. Bedoll, and updated by David M. Katz and Kenneth R. Josenhans.

Written by: John K. Renwick
John K. Renwick

Approved by: Richard R. Moore
Richard R. Moore

Appendix A - FREND routines.

A list of all FREND decks, and a brief description, follows. Note that each IDENT contains routines which are functionally similar. When adding new routines, the general scheme illustrated below should be preserved.

FREND	FREND version number and global comments. Also contains the FREND correction history.
INITIAL	all initialization processing. This is the first routine to execute after FREND is loaded. It sets up all tables.
BANNER	formats the banner pages for the printers.
BUFFER	manages the FREND buffer circular stacks. Fills BF80 and BF240, and empties BFREL.
BUS	contains subroutines to use the bus switch and the bus status table.
CLOCK	maintains the FREND time-of-day clock and the current date. Driven by the hardware line-frequency clock.
CLOSE	all close processing for ports and sockets.
COMMAND	contains processing for all the FREND commands.
CPCLO	processing for control port close, control port open, and control port activity (the 60 second activity timeout on a control port)
CTLPT	processes all control port messages from the Cyber mainframe.
DEVICE	contains the definitions of all hardware devices connected to FREND. Used only at initialization time.
ECOBUF	processes the CTRL-Q function, which echos the current input buffer back to the user.
ERROR	issues an error message and dumps registers to the console TTY for FREND-detected non-fatal errors.
FECMD	does initial processing for front-end commands,

calling upon COMMAND to do the actual processing.

GETPRT	asks the mainframe for new print jobs.
INESC	handles the input abort (escape) process.
INST	a collection of all instrumentation counters maintained by FREND. This contains data cells only, no executable code.
INST65	sends instrumentation data to the mainframe.
IOMSG	issues I/O device diagnostics ("paper out").
ISRROUT	general interrupt service routines, containing processing for all illegal interrupts, including FREND crashes.
ISR65	processes all interrups from the mainframe. An interrupt is generated whenever 1FP reads a record from, or writes a record to, the 7/32.
LOGMSG	updates and delivers the login message.
MANAGE	the MANAGER for FREND allocatable memory. Handles all requests to get and return memory blocks.
MISS	a collection of miscellaneous subroutines.
MONITOR	the FREND MONITOR. It initiates all FREND tasks.
NEXTLIN	unpacks block-data buffers into line buffers (currently called only by PRINT).
OPEN	all open processing for socket-to-port and port-to-port connections.
OUTISR	the interrupt service routine for output to the terminal (driven by the CCB)
PALISR	the interrupt service routine for all input from the terminal. Interrups on each input character.
PARSER	general command line parser for FREND command processing.
PALTEST	the PAL test routine.
PPIOCL	processing for all port-to-port connections.
PREPRT	performs pre-print setup functions.

PRINT	the batch printer driver. Processes every output line to printers. Initiates transmission of data.
PRSTAT	formats the response for the "PRNTSTAT" command.
PRTISR	the interrupt service routines for printers.
PRTMISS	contains miscellaneous subroutines for the printers.
PRTTST	performs on-line printer diagnostics.
SENDCP	sends various protocol records to mainframe control ports.
SKINCL	socket input control. Processes every input line from a socket (terminal).
SKINIT	socket initialization. Resets a socket for a new user.
SKOPEN	contains the various tasks associated with answering a phone and establishing an initial connection.
SKOTCL	socket output control. Processes every output line to a socket, and initiates transmission of the data to the terminal.
SKXL	the socket input translation tables.
SOCMSG	sends messages from FREND directly to a socket.
SVCROUT	supervisor routine processor. Processes all task requests.
TIMER	manages the timer queue.
TRACE	issues the FREND trace messages.

Each FREND routine is relocatable. All linkage is done through entry points, using the CYBER loader for linkage and relocation. A full load map should be consulted to locate all entry points and their appropriate ident.

Appendix B - FREND tasks.

The following is a list of all tasks, with a brief description. The IDENT containing the task is given in parenthesis.

CKTIMR	(TIMER) remove expired entires from the timer chain, and request the task.
CLOCK	(CLOCK) increment the time-of-day clock by 1 second, adjusting the date if necessary.
CLOFPT	(CLOSE) close a connection from the data port side.
CLOFSK	(CLOSE) close a connection from the socket side.
CONMSG	(SOCMSG) send a message to a connection.
CPACT	(CPCLO) control port activity 60 second timeout.
CPCLO	(CPCLO) control port close processing.
CPOPEN	(CPCLO) control port open processing.
CTLPT	(CTLPT) processes all control port messages from the mainframe.
ECOBUF	(ECOBUF) processes the CTRL-Q - echo current buffer.
ERRMSG	(ERROR) issues the message and register dump generated by the ERROR macro.
FECPT	(FECMD) process a front-end command from a data port.
FECSK	(FECMD) process a front-end command from a socket.
GETPRT	(GETPRT) get a print file from the mainframe.
INESC	(INESC) process the escape (abort) function.
INST65	(INST65) sends instrumentation data to the mainframe.
IOMSG	(IOMSG) issue an I/O device diagnostic.
MSGCP	(SENDCP) send a pre-formatted message to a control port.
OPENPP	(OPEN) open a port-to-port connection.

OPENSP	(OPEN) open a socket-to-port connection.
PALATO	(PALTST) processes the autotest sequence.
PALTDY	(PALTST) termination processing for the PAL test.
PALTST	(PALTST) initiates the PAL test.
PMMSG	(IOMSG) issue a message for the "PM" carriage control.
PPESC	(INESC) process an escape (FP.ABT) on a port-to-port connection.
PPIOCL	(PPIOCL) port-to-port input/output control.
PREPRT	(PREPRT) pre-print processing for printers.
PRINT	(PRINT) batch printer driver task.
PRTTST	(PRTTST) batch printer on-line diagnostic test.
PULSE	(TIMER) runs every 1/4 second and clears the H.FEDEAD flag.
RELBUS	(BUS) releases a bus switch reservation.
RTRV	(RTRV) processes the CTRL-U - retrive current line.
SENDCP	(SENDCP) constructs certain messages for control ports.
SENDIO	(IOMSG) send a message to I/O terminals.
SETBD	(SKINIT) autobaud timeout task.
SINT	(PRTMISS) simulate an interrupt on a device.
SKCARR	(SKINIT) ensures that a carrier is present 1/4 second after a user connects.
SKINCL	(SKINCL) socket input control - processes every input line.
SKINIT	(SKINIT) initializes a socket to answer a call.
SKOPEN	(SKOPEN) returns the header message and opens a connection to MISTIC.
SKOTCL	(SKOTCL) socket output control. Processes every output line.
SOCMSG	(SOCMSG) send a message to a socket.

TIMMOR	(TIMER) increase the size of the timer chain.
TIMR	(TIMER) sort a new entry onto the timer chain.

Appendix C - FREND interrupt service routines.

The following is a brief description of all the ISR's in FREND.

I#ARITH (ISRROUT) arithmetic fault.

I#MMALF (ISRROUT) machine malfunction (parity error). Also entered when 1FP kills the 7/32.

I#PROT (ISRROUT) protect mode violation.

I#QUEUE (ISRROUT) system queue interrupt

I#DISP (ISRROUT) console display interrupt. Resets W#DISP, the address of the word to display on the front panel display.

I#ILLEG (ISRROUT) illegal instruction interrupt. This is the FREND intentional crash processor.

I#NODEV (ISRROUT) immediate interrupts for which there is no device are trapped by this.

I#LFC (CLOCK) processes the LFC (line frequency clock) interrupts which are used to maintain the time-of-day clock.

I#PIC (TIMER) processes the PIC (programmable interrupt clock) interrupts. The PIC is used to maintain the timer queue, holding all delay task requests.

I#DEV5 (ISR65) processes all interrupts from the mainframe interface. These are generated by 1FP whenever it reads a buffer from, or writes a buffer to, the 7/32.

I#PAL (PALISR) processes the interrupts from the input side of the PAL when it is idle, ringing, or in autobaud detect phase. It does not process PAL interrupts for normal PAL input.

PALMIN (PALISR) processes the character-by-character interrupts for each input character when the PAL is in normal input state. This is the routine which accepts user input.

I#CCB (OUTISR) processes the interrupt when the PAL output CCB exhausts a buffer, or encounters a bad status. This routine switches output buffers and processes the right margin for all output to a terminal.

I#BUSSW (ISRROUT) Bus switch interrupt routine.

I#PRINT (PRTISR) processes the interrupts from the printers. Bad status, end-of-buffer (with buffer switching), and unconditional transfer (ADC switched off) are handled here.

Translate table routines (OUTISR) These routines are entered when the output CCB is about to send certain characters. The specific characters and the associated routines are defined in the CCB output translate table, established by INIOCT (OUTISR).

All ISR addresses are set by INITIAL.

For debugging purposes, a circular list of the last 20 interrupts is maintained at entry point INTSTK. This list is maintained by the EXITINT macro and it may be read by BUG. (PIC and LFC interrupts are not stored.)

SECTION MAP

SECTION MAP

SECTION MAP

SECTION MAP

SECTION MAP

SECTION MAP

SECTION **PAGE**