Technical Report

UCAM-CL-TR-12
ISSN 1476-2986

Number 12

HASP "IBM 1130" multileaving remote job entry protocol with extensions as used on the University of Cambridge IBM 370/165

M.R.A. Oakley, P. Hazel

September 1979

ISSN 1476-2986

Summary

This document brings together most of the information required to design, write and operate a HASP Remote Job Entry Terminal program. Most of the document describes facilities available using any host computer supporting the HASP protocols. The remainder of the document describes improvements to these facilities which have been made on the University of Cambridge IBM 370/165 in order to enhance the reliability of the system, to make it easier to run, and to provide for a wider range of peripherals than the basic system.

M.R.A. Oakley
P. Hazel

September 1979

Contents

1.0 Introduction

This document describes the communications protocols used to connect an "intelligent" Remote Job Entry (RJE) workstation terminal to a mainframe host computer for the purpose of submitting jobs to the host's job stream and receiving output produced by such a job. This output is directed to one or more peripheral devices attached to the workstation.

The protocol described is the IBM/1130 workstation protocol, together with a number of extensions available in general only on the Cambridge 370/165. The original program is supplied by IBM as part of the Houston Automatic Spooling Priority (HASP) system which runs on many IBM 360 or 370 computers in conjunction with the OS/MVT or other IBM operating systems. Note that other protocols (e.g. IBM/2780) exist within HASP but are outside the scope of this document.

Other computers can be operated as workstations by emulating the IBM/1130 interface. Machines currently connected in this way to the Cambridge system include other IBM/370's, GEC 2050's and 4070's, INTERDATA-70's, PDP-11's, a CTL Modular One and a Honeywell (RXDS) Sigma-6.

Information in this document is derived from several sources which are listed in the reference section below. Some familiarity with communications protocols, possibly from such a reference, is assumed throughout this document.

2.0 General Overview

Communication between a workstation and the host mainframe is carried out by transmitting blocks consisting of control and possibly data information from one computer to the other. There is effectively a shuttle in which a block is transmitted in one direction, and an acknowledgement of successful or unsuccessful receipt of this block is transmitted back to the original sender. A negative acknowledgement causes the block to be retransmitted. A positive acknowledgement allows the sender to proceed to the next block, and in addition may contain data passing in the opposite direction. Thus the protocol can be used on either a full duplex or a half duplex communication line.

Blocks are sequence-checked and checksummed to avoid loss of data. If the sender does not receive any acknowledgement within a fixed time (usually 3-10 seconds) the block must be retransmitted. This is called a "timeout".

2.1 Signon

Before transmission of any data can commence the workstation must identify itself to the host computer. The workstation announces its presence to the host by sending an initialisation block. When a positive acknowledgement is received, a signon block identifying the workstation is sent and when this gets a positive acknowledgement, signon is complete.

2.2 Data Transmission

Once a workstation has signed on it can transmit operator console and reader input, and must be prepared to receive output. Before sending input from a reader however, it must request permission from HASP by sending the appropriate control record and waiting for the reply. This does not apply to operator console input which may be transmitted at any time. Once a permission-to-send control record has been received for this device, the reader concerned will be "ACTIVE" until an end-of-file (null) record has been sent, at which point it reverts to its original "INACTIVE" state. Permission-to-send must be requested again before transmitting further records. An end-of-file is normally sent at the end of each job.

Similarly, when the host wishes to send printer, punch or plotter output from a job to the workstation, it will send a control record asking permission to start sending to the device, will await a permission-to-send in reply before starting to send the output, and will eventually terminate this output with an end-of-file. Again, this does not apply to operator console output which may appear at any time and is never terminated by end-of-file.

Physical transmission blocks may contain records from more than one device and control records interleaved. Construction of such a block should be terminated however when a control record or operator console input record is encountered, to ensure this is transmitted and acted upon as soon as possible.

At present, blocks sent by the host to workstations contain records for one device only to facilitate stream control by the workstation. Most existing workstation programs utilise this fact, which may be a dangerous thing to do.

2.3 Stream Control

During transmission, data streams may be operating for several different devices at a time. Workstation programs typically do not have much available bufferage for data directed at each device and may find it necessary to suspend transmission of data for one or more devices for short periods until backlogs have been cleared. To facilitate this, a pair of bytes in the header of each block are designated as function control sequence (fcs) bytes.

One bit in this sequence is allocated to each device. The receiver of the stream may set this bit to zero to suspend transmission of data for or from this device, and set it on again later to allow continuation. One bit in the fcs (the WAIT-A-BIT bit) is used similarly to suspend transmission for all devices in cases of, for example, general buffer shortages.

Note that there are two logical sequences - that sent by the host to the workstation, and that sent by the workstation to the host. After receiving a block with WAIT-A-BIT set, only blocks containing no data records may be transmitted until a block without WAIT-A-BIT is received. Normal transmission may continue in the other direction.

2.4 Signing Off

There is no direct equivalent to the signing-on sequence. When a workstation wishes to terminate a connection to the host, it may either transmit a single record from the principal reader containing just '/*SIGNOFF' and await the completion of all current output documents before breaking the connection, or it may simply just stop transmitting, in which case the host will timeout the workstation and sign it off within a minute or so. [The workstation can adopt a similar attitude to a long host timeout, by assuming the connection broken and attempting to sign on again.]

[Cambridge only:] An alternative to the /*SIGNOFF card from the reader is the operator console $SIGNOFF command. This has the advantage that it is operative at all times; the /*SIGNOFF card will not be read if the reader concerned is drained or otherwise inoperative.

As a result, an attempt to reconnect before the previous connection has timed out may not succeed on a dialled line as this will appear engaged (busy) with the workstation still active.

3. Transmission Blocks

Here follows a list of all the possible types of block which may be transmitted, with comments on their use. Blocks consist of a sequence of control characters intermixed with the actual data characters. The values of the principal control characters (in hex) are

SOH	01	STX	02	DLE	10
ETB	26	ENQ	2D	SYN	32
NAK	3D	PAD	FF		

All blocks start with two or more SYN characters and end with a PAD character (omitted below). In practice four or five SYN characters are recommended to give the host time to turn the transmission line round and start receiving. Blocks containing data are of two types - transparent and non-transparent. Transparent blocks begin with DLE STX and subsequent control characters must be preceded by DLE to be recognised. A data DLE is then transmitted as two successive DLE characters. Non-transparent blocks begin with SOH STX, and control characters are recognised anywhere. All data is transmitted in EBCDIC code. Redundant SYN characters (or, in the case of transparent blocks, DLE SYN pairs) may occur anywhere in the block, and should be ignored. They are often inserted by the communications multiplexor to fill in time. These characters can cause the number of bytes in a block to be greater than normal, and this should be allowed for by any workstation program.

3.1 Blocks Not Containing Data

a) Initialization Block

```
-----------
|SOH |ENQ |
-----------
```

This block is the first block sent by the workstation when it wishes to sign on to the central system. The only valid response is an ACKO block.

b) ACKO Block

```
-----------
|DLE | 70 |
-----------
```

This is a general positive acknowledgement, sent whenever there is no data block to send, or when the receiver has requested a temporary hold up of data blocks ("wait-a-bit"). Sending ACKO implies that the wait-a-bit state for the ACKO sender is reset.

c) NAK block

```
------
|NAK |
------
```

This is a negative acknowledgement. The last block sent that was not itself a NAK should be re-transmitted.

3.2 Blocks Containing Data

Blocks containing data consist of the data bytes themselves and a number of structured control bytes. These are as follows:

bcb -	block control byte. This is used to maintain a cyclic sequence count of blocks so that sequence errors, missed or repeated blocks can be easily detected.
crc -	cyclic redundancy check (two bytes). This is used to checksum each block transmitted so that transmission errors within a block can be detected.
fcs -	function control sequence (two bytes). This is used to control the flow of data to and from each device stream.

The most significant bit in each of these bytes is always set to one. Any control byte with a zero in this bit is due to an error - the block should be faulted (see 3.3 (c)).

Two formats are used for data blocks

i) non-transparent

```
-----------------------------------------------------------------------
|SOH |STX |bcb |fcs |fcs |data & control records| 00 |ETB |crc |crc |
-----------------------------------------------------------------------
```

ii) transparent

```
----------------------------------------------------------------------------
|DLE |STX |bcb |fcs |fcs |data & control records| 00 |DLE |ETB |crc |crc |
----------------------------------------------------------------------------
```

The non-transparent form is used by HASP only for blocks which do not contain data records. The current maximum length of data blocks is 400 bytes. This count includes all bytes from the bcb to the zero byte indicating end of block, but without any necessary DLE characters for purposes of transparency.

3.2.1 bcb

The bits in the block control byte (bcb) are used as follows

```
----------
|OXXXCCCC|
----------

   O   = 1 (must always be on)
  XXX  = control information
       = 000 normal block
       = 001 bypass sequence count validation
       = 010 reset expected sequence count to CCCC
CCCC   = modulo 16 block sequence count
```

The block sequence count starts at 0 at signon and is incremented (modulo 16) by one for each non-retransmitted block containing a bcb thereafter. All stations must be prepared to receive the same block twice (ignoring it the second time) to cope with the case of a garbled acknowledgement.

3.2.2 fcs

The bits in the function control sequence halfword (fcs) are used to control the flow of data streams from or to individual devices as follows:

```
-------------------
|OSRRABCD|OTRRWXYZ|
-------------------

O = 1 (must always be on)
S = 1 suspend ALL stream transmission (WAIT-A-BIT)
  = 0 normal state
T = remote console stream (input and output)
R = reserved
A,B,C... = normal print or input streams
Z,Y,X... = normal punch or plotter streams
```

A bit on => continue transmission; a bit off => suspend transmission.

The assignment of individual bits to device streams is by mutual arrangement between the workstation staff and the host staff.

Note that there are two logical sequences - that sent by the host to the workstation, and that sent by the workstation to the host. After receiving a block with the WAIT-A-BIT bit set, only blocks containing no data records may be transmitted until the WAIT-A-BIT bit is reset or an ACKO is received.

3.2.3 Data and Control Records

Records containing data consist of the data bytes themselves and a number of structured control bytes. These are as follows

rcb - record control byte. Describes the type of record.

srcb- sub-record control byte. Provides additional information particular to the individual record type.

scb - string control byte. Describes the internal structure of the data in the record.

The most significant bit in each of these bytes is always set to one (unless the whole byte is zero - see below). Any non-zero control byte with a zero in this bit is due to an error - the block should be faulted (see 3.3 (c)).

The format of data records is:

```
----------------------------------- - - -----
|rcb |srcb|scb |data|scb |data|scb |      | 00 |
----------------------------------- - - -----
```

Control records begin with an rcb in the same way, but the data format thereafter depends on the control record type (see sections 3.2.4 and 3.3). A control record must be the last record in the block.

3.2.3.1 Record Control Byte

The bits in the record control byte (rcb) are used as follows:

```
----------
|OIIITTTT|
----------
```

A zero rcb indicates logical end of transmission block.

```
   O = 1 all other valid rcb's
 III = stream identifier if TTTT ~= 0 (starts at 1)
 III = control information if TTTT = 0
     = 001 request to initiate stream transmission (prototype rcb in srcb)
     = 010 permission to start stream transmission (prototype rcb in srcb)
     = 110 bcb error indication
     = 111 general control record
```

```
TTTT = record type identifier
     = 0000 control record
     = 0001 remote operator console output record
     = 0010 remote operator console input record
     = 0011 normal reader input record
     = 0100 print record
     = 0101 card punch record
     = 1101 papertape punch record [Cambridge only]
     = 1110 plotter record         [Cambridge only]
```

Thus the commonly encountered rcb's are:

93	input record from reader 1	
A3	input record from reader 2	
92	remote operator console input record	
91	remote operator console output record	
94	output record for printer 1	
95	output record for punch 1	
9D	output record for papertape punch 1	[Cambridge only]
9E	output record for plotter 1	[Cambridge only]
90	request to send (control record)	
A0	permission to send (control record)	

3.2.3.2 <u>srcb</u>

The sub-record control byte (srcb) is used as follows::

a) For request-to-send and permission-to-send control records, the srcb contains the rcb for the device for which permission is being requested or granted.

b) For a general control record, the srcb can only have the following value

C1	initial terminal signon record

No other general control records are currently implemented.

c) Format of print record srcb:

```
----------
|OMCCCCCC|
----------

     O = 1 normal print record
     M = 0 always
CCCCCC = carriage control information
       = 1000NN space (linefeed) immediately NN spaces
       = 11NNNN skip immediately to channel NNNN
                (channel 1 is top of page - channel 12 is bottom
                 of page and used for printing job trailers.)
       = 0000NN space NN lines after print (NN=00 for overprinting)
       = 01NNNN skip to channel NNNN after print.
```

d) Format of card punch and input record srcb:

```
----------
|OMMBRRRR|
----------

     O = 1 normal record
    MM = 0 always
     B = 0 record is translated
       = 1 record is column binary and not translated   [Cambridge only]
  RRRR = 0 reserved
```

e) Format of papertape punch record srcb: [Cambridge only]

```
----------
|OMMBRRCF|
----------

     O = 1 normal punch record
    MM = 0 always
     B = 0 Data part of the record requires code translation
            and may require termination.
       = 1 Data part of the record is binary.  It requires no
           translation or termination.
    RR = 0 reserved
     C = 0 The record is a data record.
       = 1 The record is a control record and restart marker.
     F = 0 Record is to be terminated with newline if not binary.
       = 1 Record is not to be terminated
```

f) Format of plotter record srcb: [Cambridge only]

```
----------
|OMMBRRCF|
----------

     O = 1 normal plotter record
    MM = 0 always
     B = 0 Data part of record requires translation (not used at present).
       = 1 Data part of the record is binary.  It does not require
           translation.
    RR = 0 reserved
     C = 0 The record is a data record.
       = 1 The record is a control record and restart marker.
     F = 0 Terminate current picture before processing record.
           The new picture has its origin beyond the highest
           point used in the current picture.
       = 1 Continue the existing picture.
```

3.2.3.3 Character Strings

Data records are compressed by detecting strings of duplicate characters, and splitting the record into a sequence of string control bytes (scb) and strings. The scb is used to indicate whether the string is a string of blanks, another duplicated character string, or a string of distinct characters. The bits in the scb are defined as follows

```
----------
|OKLJJJJJ|
----------
```

```
A zero scb indicates end of record.
     O = 1  all other valid scb's
     K = 0  duplicate character string
            L = 0 duplicate character is blank
              = 1 duplicate character is non-blank and follows scb
            JJJJ = duplication count
     K = 1  non-duplicate character string follows scb
            LJJJJJ = length of character string following scb
```

Input records must always contain at least one data character (i.e. a blank if the record is empty). A null record (a record with the first scb zero) indicates end of file on input or output. When reconstructed, input records must not be longer than 80 bytes.

3.2.4 Cyclic Redundancy Check

The two cyclic reduncancy check (crc) bytes at the end of each transmission block containing data act as a checksum for the block. An algorithm for computing the 16-bit crc word is discussed in appendix B. For non-transparent blocks, all bytes between STX and ETB inclusive (but excluding any SYN's) are included in the checksum; for transparent blocks all bytes after STX up to and including ETB are included, with the exception of non-data DLE's (i.e. the first DLE's in DLE/something pairs) and SYN's immediately following a DLE. The checksum is transmitted with the least significant byte first.

Note that values corresponding to DLE and SYN may occur in crc bytes, and that these are not escaped even in a transparent block. This implies that one may not pad with SYN characters when a crc is due to be sent, and that such pad characters should not be ignored when a crc is due to be received.

3.3 Some Special Control-only Blocks

a) Signon block

```
---------------------------------------- - - - - - ----------------------
|SOH |STX | 80 | 8F | CF | F0 | C1 |signon card| 00 |ETB |crc |crc |
---------------------------------------- - - - - - ----------------------
           bcb  fcs  fcs  rcb  srcb
```

This block is sent by a workstation after it has received an ACKO reply to an initial SOH ENQ. Either another ACKO or a data block may be received as positive replies to this block.

The format of the "signon card" is as follows; where nn denotes the remote number assigned together with the workstation identifier by the host computer administration. Data compression scb's are not used and 80 bytes of data must be sent.

```
COL
1              16       25
/*SIGNON       REMOTEnn identifier
```

b) Wait-a-bit or fcs change block (transparent form shown)

```
----------------------------------------------------------
|DLE |STX |bcb |fcs |fcs | 00 |DLE |ETB |crc |crc |
----------------------------------------------------------
```

This block is sent

i) to set the wait-a-bit bit in the fcs when the sender has no buffers left to receive data blocks, or
ii) to change bits in the fcs when the sender has no data buffers to send.

One can of course do either of these activities as part of a normal data block if there is actual data to send.

c) Bcb error indicator block (transparent form shown)

```
--------------------------------------------------------------------
|DLE |STX | An |fcs |fcs | E0 | 8m | 00 | 00 |DLE |ETB |crc |crc |
--------------------------------------------------------------------
           bcb             rcb  srcb scb  rcb
```

This block is sent when an error in the bcb of a received block is detected. The bcb setting An says "reset sequence number to n", while the control record E08m indicates a bcb error where m was the expected sequence number.

The probable outcome of this is that the host will start ignoring the workstation, which will eventually time out. After signing on again, printer streams will restart backspaced a couple of pages, other output streams will be restarted from the beginning, and any jobs on readers will have been cancelled and must be resubmitted.

3.4 Data Records for Plotters and Paper-tape Punches [Cambridge only]

Data records for special devices can be either normal records (C=0 in srcb) or additional control records (C=1 in srcb). The format of these is:

3.4.1 Control Records for Paper-tape Punch

0	Offset of start of data.	
1	Flags xxxxxxx.	
	X'80'	Restart request
	X'40'	End of phase
	X'20'	End of document
	X'10'	Start of document
	X'08'	Record to be followed by runout
	X'04'	Runout, if present, is short
	X'02'	Data has special status - normally data to be ignored by simple stations.
2	Copy of SRCB bits B,C and F.	
3-5	Restart Marker number	
6-7	Job number in binary	
8	Width mask X'FF'=8, X'7F'=7 and X'1F'=5 track	
9	Code 0=ASCII, 4=Flexowriter	
10-end	Reserved fields and data (may be null)	

The total length of a control record will not exceed the total length of the largest data record.

3.4.2 Control Records for Plotter

0	Offset of start of data.	
1	Flags xxxxxxx.	
	X'80'	Restart request
	X'40'	End of phase.
	X'20'	End of document.
	X'10'	Start of document.
	X'08'	Any picture termination to be followed by gap.
	X'04'	Gap, if present, to be short.
	X'02'	Data has special status - normally data to be ignored by simple stations.
2	Copy of SRCB bits B,C and F.	
3-5	Restart marker number	
6-7	Job number in binary.	
8-10	Picture width in increments.	
11-13	Picture height (length) in increments.	
14-end	Reserved fields and data (may be null)	

3.4.3 Control Records and Restarting

The continuous period during which a job is active on a particular device is called a phase. Within a single phase, a number of distinct documents may be transferred, possibly corresponding to the use of different DD statements in the parent job. Documents within a phase will normally have similar operator setup requirements (e.g. paper tape width). A phase is terminated by either an end-of-phase control marker or a change of job number.

The first record of the phase is a control record and is numbered 1. For each subsequent record the number is incremented by 1, and inserted in control records as they occur. The numbers of the data records between control records should be maintained. If the transmission is restarted or interrupted then some or all of the records will be retransmitted, starting with a control record. Within a phase control records of number lower than the highest already processed should therefore be discarded, together with any following data records. The receipt of control records of sequence number exceeding the number of the current record by more than 1 represents an error. On receipt of a control record with number equal to that of the highest already processed, the appropriate number of data records should be counted off and then output resumed.

Notwithstanding the above, if a control record is marked as a restart request, then its restart marker should be taken as current and output commenced immediately.

The structure of data records for a plotter is described in Appendix C.

4.0 Design of Workstation Programs

Workstation programs are often based on the original IBM/1130 program, and consist of a number of semi-independent processes scheduled by a simple round-robin coordinator. In such a program the following processes may exist:

4.1.1 Line Driving Process

Transmits and receives blocks from the datalink, according to the protocols described in section 3, maintaining the block sequence count and setting the checksum and function control bytes for each block. Blocks for transmission are taken from a first-in first-out chain constructed by the block-building process (4.1.2). Received blocks are placed on a chain ready for the unpacking process.

Recovers from transmission errors in both directions

a) If a read timeout occurs, transmits a negative acknowledgment (NAK). Many successive timeouts usually mean that the host computer itself has stopped. In this case the workstation may have to sign on again, and reread from the beginning any uncompleted input jobs.

b) NAK received - retransmit the most recent block that was not NAK.

c) checksum error, incomplete block read, overlength block read, general garbage - transmit NAK.

In the event of the host sending WAIT-A-BIT, this process should immediately stop sending data blocks, and send only fcs or WAIT-A-BIT change blocks or ACKO. If this process runs out of receive buffers, it should send WAIT-A-BIT to the host, but nevertheless be prepared to accept null data blocks with fcs or WAIT-A-BIT changes. Any other data in these circumstances should be rejected with NAK.

4.1.2 Block Packing Process

Collects buffers chained by the device-input processes and packs these into 400-byte blocks for transmission to the host, in the format described in section 3.2. A block is terminated and put on the chain when

a) it is full (the next record will not fit), or
b) an operator console input record has just been inserted, or
c) an end-of-file has just been inserted, or
d) a control record has just been inserted.

Completed blocks are queued for the line driving process. The received fcs bits are examined by this process, and records are not accepted from a device with fcs bit zero. The number of blocks queued for transmission is usually limited by this process to prevent buffer shortages and also prevent excessive amounts of data from being queued ahead, which can make it difficult to decide which jobs to reread after a restart.

Note that blocks already queued for transmission can hardly be unpicked of records for such a device - the host will allow enough overrun in such cases for this not to be a serious problem.

4.1.3 Block Unpacking Process

There is an unpacking process for each output device, and one for control records (with most of the code usually as a subroutine common to all such processes). This enables the system to cope with interleaved data records in one block - if one process is halted for lack of output buffers the rest can proceed.

Each process scans blocks in the received chain for records with the correct address (rcb). As these are found they are unpacked into output buffers and queued for the relevant device process. The control record process interprets control records and sets various flag bits to be interpreted by other processes. When each process has finished with a block, it sets a marker in the block to indicate this, and when all the markers are set, unchains the block and returns it to the free chain.

If a process must halt waiting for output buffers, it clears the relevant bit in the outgoing fcs before halting in order to suspend transmission of further data records for this device. When it subsequently gets buffers again, it turns the bit back on. The outgoing fcs is inserted into transmission blocks by the line driving process.

4.1.4 Operator Console Input Process

Reads input lines from the operator console (teleprinter); and, if the line starts with '$' or ';', queues the buffer for transmission to HASP. If not, it interprets the command as a workstation local command and acts on it (used for controlling papertape readers etc.). This process must be interlocked with the operator console output process as far as control of the teleprinter is concerned.

A brief description of current HASP operator commands appears as appendix D.

4.1.5 Card Reader Process

Watches its card reader until it becomes ready; queues a control record containing request-to-send; waits on a flag until it is set by the control record unpacking process to signify permission-to-send has been received for the device; reads cards and queues the data buffers for packing (handling any device errors appropriately); detects end-of-file on the device and indicates this to the packing process. (This may be effected by an EOF button on some card readers, or by the reader running out of cards with the last card read /*anything or //<78 blanks> on others.)

4.1.6 Paper Tape Reader Process [Cambridge only]

Basically the same as the card reader process, with some extra logic for handling warning sequences, binary tapes, etc (see appendix A). In order to permit arbitrary binary data to be input, the HASP operator command $T<reader name>,D (e.g. $TRM3.RD2,D) is provided. This causes all records following a /*DATA card until end-of-file to be treated as data, i.e. cards starting with /* and // are not recognized. This facility is clearly dangerous for real card readers, hence the requirement to authorise it each time by operator command. In the absence of authorization, /*DATA records are ignored. In the case of paper tape readers, however, jobs cannot be stacked up, and there is no danger of one job swallowing another as long as an end-of-file is sent at the end of each job. When a remote workstation is generated in HASP, it is possible to specify that certain of its logical readers are to be permanently authorized for binary data -- these should correspond to paper tape readers. The paper tape reader process must insert /*DATA when it switches to binary input.

4.1.7 Operator Console Output Process

Types lines of output from the queue set up by the console output unpacking process and by any other process that wants to output an operator message. The use of the teleprinter is interlocked with the console input process.

4.1.8 Line Printer Process

Prints lines of output from the queue set up by the line-printer record unpacking process; provides suitable operator error messages.

4.1.9 Card Punch Process

Punches cards from the queue set up by the card punch record unpacking process and provides operator error messages and job identification.

4.1.10 Plotter/Paper-tape Punch Stream Control Process [Cambridge only]

This process dequeues records for the special output devices, recognizes and records restart markers, and counts records between restart markers. If it detects a restart request, it skips records in an immediately retransmitted output stream and continues from the first unrepeated record. Records containing data are queued for the relevant special device process.

4.1.11 Plotter Process [Cambridge only]

Dequeues records from the queue set up the the stream control process. Converts these into appropriate plotter movement and control commands and uses these to drive the plotter. Provides suitable operator error messages.

4.1.12 Paper-tape punch process [Cambridge only]

Dequeues records from the queue set up by the stream control process, and punches the data from these on the papertape punch, performing any necessary code conversion and end of record processing (e.g. adding newline characters). Provides suitable operator error messages.

4.2 Implementation Notes

4.2.1 Buffer Numbers

Each device is observed to work fairly well with two buffers (double buffering). Slow devices can manage with one with an occasional delay between records. Fast devices such as lineprinters may run better with three buffers in a system with many devices. The operator console requires more buffers - five are often provided - otherwise error messages from local devices can be delayed.

The number of transmission buffers of 400 bytes each depends on the number of devices. One is generally needed for each output device, plus one being filled by the packing process, plus one being transmitted or receiving data, plus one queued for transmission. More are advisable if space permits, although in fact most workstation programs will run with fewer. The effect in such cases is for devices to run alternately in 'bursts' of a few records at a time at periods when several devices are active.

If the process which receives data from the line does not remove extraneous SYN or DLE-SYN characters, it must be prepared to receive more than 400 characters in a block. It can either cope correctly, or send NAK when 400 is reached to force a (hopefully shorter) retransmission.

4.2.2 Currently Known Problems

a) There is a bug in the standard version of HASP which causes it not to deal with a signon correctly if the workstation is already signed on. In this case the host should reset all remote devices to INACTIVE, and cancel any partially read jobs; in practice neither of these things occurs. Thus if a workstation re-starts itself without previously having signed off (e.g. due to a crash in the workstation program) it may find itself out of step with regard to the state of its peripherals, and the following problems can arise

i) If a job was being read, HASP will have the reader ACTIVE, and when a request to send (for another job) is received in this state, it simply starts ignoring the workstation! The remedy is to re-start a few times until the reader becomes INACTIVE.

ii) If a job was printing/punching, HASP will continue to send the output without first asking permission. The workstation program should be capable of dealing with this state. Many existing workstation programs restart with their output processes stopped, waiting for a request to send, which aggravates this problem.

b) Error recovery in HASP is minimal, and if it receives something it does not understand its response is either

i) to start ignoring the workstation altogether, or
ii) to send WAIT-A-BIT for ever more.

For example, sending a request to send when permission has already been granted causes (i), while sending data before asking permission causes (ii).

4.2.3 HASPGEN

For a workstation to be able to connect to a host computer, the workstation must be configured into the host system as part of a HASPGEN process performed by staff at the host site. This defines the workstation together with which peripherals it may use and certain of their characteristics (e.g. printer width etc.). Certain options, such as plotters and papertape punches, are unlikely to be found other than on the Cambridge 370/165.

4.2.4 Design Bugs Observed in Workstation Programs

a) Failure to cope with interleaved output blocks (noted above).

b) Failure to respond immediately to an fcs bit being reset (there are usually buffers already on the output queue). This is normally not a serious problem as HASP can cope with one or two extra blocks.

c) Failure to cope with receiving data before a request to send (noted above).

d) Assumption that any response other than NAK (including garbage!) indicates block correctly received. (However, this is probably a fairly safe assumption).

4.2.5 Implementation Bugs Observed in Workstation Programs

a) Incorrect action when NAK received. It should retransmit the previous block that was not itself a NAK.

b) Failing to transmit NAK on a read timeout (usually retransmits the previous block instead).

4.2.6 General Remarks

The following features have been found convenient in the operation and debugging of some of the local workstations

i) The coordinator causes the machine's console display lights to count continuously, thus indicating at a glance that the system is running.

ii) Logging of line errors on the operator's console is useful for debugging but a nuisance in general operation -- it could be switched off by a handkey. The errors that need logging are NAK in, NAK out, crc check, ENQ in (should never occur), bcb check, timeout, etc. Counts of NAK's in and out can also be displayed on (another set of) console lights, so that even if the hardcopy logging is off, a great burst of errors is noticeable.

iii) The message **WAIT typed after receiving a number of successive wait-a-bit fcs's (currently 3), followed by **CONT when the wait-a-bit clears enables a permanent wait state to be easily recognized. (These do occur at times -- see 4.2.2b.)

iv) *TIMEOUT can be typed for each timeout -- after a certain number in a row the system restarts itself.

v) If it is possible to switch the operator's console output to the printer this is very useful for listing all the jobs belonging to a workstation, which can take a long time on a teleprinter.

vi) A testing version of the system in which all bytes sent and received are stored in a large circular buffer is invaluable for tracing obscure bugs.

vii) A facility for running out large reels of paper tape which stop in the middle (because of fault characters, for example) makes life easier.

5.0 References, Acknowledgements, etc.

1) General Information - Binary Synchronous Communications.
IBM Form Number GA27-3004

2) Houston Automatic Spooling Priority System (HASP) - II manual.
IBM Form number 360D-05.1.014 is the Version 2 manual.

3) Many parts of this document are based on internal Computing Service documents produced by Dr P. Hazel, Dr P.F. Linington, M.J.T. Guy and possibly others.

APPENDIX A -- Detailed Specification of Paper-tape Input Logic

This section applies to the Cambridge 370/165 only.

This logic implements the Cambridge paper tape input facilities; for a user's view, see the Cambridge IBM370 User's Reference Manual.

Variables:

S	state switch for input program	
B	binary switch	) logical variables,
P	fault character don't fail switch (+++P)	) taking only values
E	translated binary switch (+++E)	) 0 or 1 (i.e. true
ST	stream mode switch (+++S)	) or false)
T	translation switched off switch	)
D	'data only' mode (/*DATA)	)

States are:

0	ignoring rest of line (after warning sequence or @L)
1	after carriage return
2	after linefeed, formfeed, or vt
3	after buffer overflow
4	normal
5	after @
6	after '+' at start line
7	after '++' at start line
8	after '+++' at start line
9	after @X

Translated characters that are specially treated:

unassigned (fault) characters (includes wrong parity chars)
linefeed, formfeed, vertical tab (treated as synonymous)
significant shifts (flexowriter only)
backspace (bs)
carriage return (cr)
tab1 (ascii tab, giving at least one space)
tab2 (flexowriter tab, giving at least two spaces)
nul (runout and non significant shifts)
@
erase (rubout)

All others are 'normal' characters, as are all characters read when translation is switched off (T=1).

Actions:

At the start of a job, S=2 and B=P=E=ST=T=D=0. When each character is read, it is translated if T=0, and the action taken is determined by the current state and the character, according to the following table

state\char	normal	fault	lf etc	shift	bs	cr	tab1	tab2	nul	@	erase
0	0	0	L0	S0	0	C0	0	0	0	0	0
1	N1	ER1	L0	S0	B1	C0	T1	T2	0	A1	E1
2	N1	ER1	L2	S0	B1	0	T1	T2	0	A1	E1
3	N3	ER1	L0	S0	B1	C0	T1	T2	0	A1	E1
4	N4	ER1	L2	S0	B1	C4	T1	T2	0	A1	E1
5	N5	ER1	L5	S0	B1	C0	E1	E1	0	A5	E1
6	N6	ER1	L2	S0	B1	C4	T1	T2	0	A1	E1
7	N7	ER1	L2	S0	B1	C4	T1	T2	0	A1	E1
8	N8	ER2	ER2	S0	ER2	ER2	ER2	ER2	0	ER2	0
9	N9	ER3	ER3	S0	ER3	ER3	ER3	ER3	0	ER3	0

Details of actions:

'RTI' means 'return from interrupt routine to await next character'

```
0       RTI

N1      if char='+' then S:=6 else S:=4; N4

N3      S:=4; N4

N4      store char in buffer; if buffer full then (transmit it; S:=3); RTI

N5      S:=4; if ~(known escape combination) then ER3;
        if @C then (backspace one character in buffer; RTI)
        if @L then (S:=0; if there is one, throw away current buffer; RTI)
        if @X then (S:=9; RTI)
        if @N then do nothing
        else char:=escape replacement; N4

N6      if char='+' then S:=7 else S:=4; N4

N7      if char='+' then S:=8 else S:=4; N4

N8      remove '+++' from current buffer; if buffer empty then throw away else
        (if ST then pad it with X'00' else pad it with X'40'; transmit buffer);
        if char='+' then treat as '+++Z'
        if char~=(letter or digit) then (alter '+' to char, S:=0; RTI);
        if warning sequence unknown then ER2;
        switch on ws:-
        +++B    B:=1; D:=1; S:=0; RTI
        +++E    E:=1; D:=1; then as +++T
        +++F    T:=1; B:=1; D:=1; then as +++T
        +++P    P:=1; S:=0; RTI
        +++S    ST:=1; D:=1; S:=0; RTI
        +++T    S:=2; await start of next tape
        +++Z    terminate job
```

```
N9      if char~=hexdigit then ER3; if first hexdigit then save in buffer,
        else (compute new char value; N3)

ER1     if P then (char:=X'FF'; N3) else 'FAULT CHAR'; kill job

ER2     'BAD WS'; kill job

ER3     'BAD ESCAPE'; kill job

L0      S:=2; if B then T:=1; RTI

L2      S:=2; L2A

L2A     if ST then (char:=X'15'; N4) else (fill out line with blanks; transmit);
        RTI

L5      S:=2; RTI

S0      flip translation tables; RTI

B1      if ST then (char:=X'16'; N3) else ER1

C0      S:=1; RTI

C4      S:=1; L2A

T1      if ST then (char:=X'05'; N3) else (S:=4; insert relevant number
        of spaces; if buffer overflow then (transmit buffer; get new one)); RTI

T2      as T1, but at least two spaces

A1      if (ST or E) then N3 else S:=5; RTI

A5      char:='@'; N3

E1      if ST then (char:=X'07'; N3) else RTI
```

APPENDIX B -- Calculation of Block Check Characters

This is an extract from a note by M.J.T. Guy entitled "Rapid calculation of block check characters".

The block check character for a message is calculated as the remainder on dividing the message (considered as a polynomial with coefficients modulo 2) by the constant polynomial

$$x^{16} + x^{15} + x^{2} + 1.$$

This division is conventionally performed as for integer division (but in the opposite direction!), proceeding along the bits, and subtracting the divisor whenever a one is encountered in the dividend, i.e. as an alternating sequence of right shifts and conditional subtracts (where of course subtract equals add equals exclusive-or, since there are no carries). However, we can proceed equally well by shifting n bits at a time and then cancelling out all the bits which have fallen off the end. Since cancelling a bit only affects more significant bits, and since the decision to subtract or not at each step depends on one of the overflow bits, we can compute the total amount to be added into the block check character as a function of the n-bit overflow, by simulating the bit-by-bit process. If this function is precalculated and stored, we can update the block check character very rapidly; for example, for n=8, with a 256 entry table, we use the sequence

```
add new character into bcc
shift bcc right eight places; let x=8 bit overflow
add f(x) into bcc
```

where "add" always denotes exclusive-or!

If a 256 entry table is too bulky, a 16 entry table can be used, based on n=4. Two shift/add operations are then required per character. We illustrate the calculation of the tables using n=4. Since there are no carries, the various bits do not interact, and we have the relationship

$$f(x+y) = f(x) + f(y) \qquad \text{('+' denotes exclusive-or)}$$

We also have the relationship obtained by interchanging a right shift and an add

```
        f(x/2) = f(x)/2                 if LS bit of f(x)=0
               = f(x)/2 + 'A001'        if LS bit of f(x)=1
```

(constants are in hexadecimal). The second relation can be used to compute successively

```
        f(8) = 'A001'
        f(4) = 'F001'
        f(2) = 'D801'
        f(1) = 'CC01'
```

and we can complete the table by addition, using the first relation:

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	CC01	D801	1400	F001	3C00	2800	E401	A001	6C00	7800	B401	5000	9C01	8801	4400

where the first row is 'x' and the second is f(x).

APPENDIX C -- Detailed Specification of Plotter Data Records

This section applies to the Cambridge 370/165 only.

The data part of plotter records is a sequence of control characters and associated data. The data length is in each case implied by the control character. Each control character is divided into two four bit digits XY.

If X = Y = 0	The character is a pad and should be ignored.
If X < 10 and Y < 10	The character represents a position.
If X < 10 and Y > 9	The character is undefined and should be ignored.
If X > 9	The character is to be interpreted as follows

X = 10	Change state.
	Y = 0 Move
	Y = 1 Draw
	The state is reset to move at the start of any control record.

X = 11	Set pen colour to Y
X > 11	Extended operation, followed by X'XY'-X'C0' bytes of data.

Positioning information is interpreted in conjunction with a current position and a pair of stored current line magnitudes, which are all reset to zero (lower left hand corner) at the start of any control record. Each piece of positioning information is relative to the current position. The X and Y descriptors have similar form, each with bits RNNS. If R is nonzero, then the magnitude of the relative coordinate (line length) is given by the stored value. If R and NN are both zero then the component is zero. If R is zero but NN is not zero then a magnitude of length NN bytes follows the control character, X before Y. This magnitude then becomes the appropriate stored value. In all cases S gives the sign of the coordinate. An implementation should suppress any occurances of pen raised followed by pen lower without any intervening move, and must window all positioning requests to the current picture size.

APPENDIX D -- Details of HASP Operator Commands Available at Workstations

Commands that display information

$DA	display first 30 or so active jobs in host queue. [Standard HASP displays entire queue.]
$DF	display number of jobs with each forms type queued at the remote issuing the command.
$DF,r	display number of jobs with each forms type queued at the specified remote.
$DJn	display status of job n.
$DJn-m	display status of jobs with numbers in range n to m inclusive.
$DJn1[-n2][,n3[-n4][,...]]	display status of list of job numbers or job ranges (up to a maximum of 5 arguments).
$D'USER	display all jobs belonging to user. [Cambridge only]
$DQ,r	display the number of jobs waiting on the queue for remote station r.
$DQ,r,PRT	display the number of jobs waiting to print at remote r.
$DN	displays first 30 or so jobs in the host job queue. [Standard HASP displays entire queue.]
$DN,r	displays first 30 or so jobs with any connection to remote r. [Standard HASP displays entire queue.]
$DN,r,PRT	display by user identifier all jobs waiting to print at remote r.
$DLNEn	display status of RJE line n and devices on the associated remote if any.
$DRMr	display status of remote station r and all its peripherals.
$DMr,'m'	display the message 'm' on the operators console at remote station r. $DMO sends the message to the central operators.

Alternatives to PRT for printer in above commands are:

PUN	for card punch	
PLT	for plotter	[Cambridge only]
PTP	for papertape punch	[Cambridge only]
HOLD	for jobs in 'HOLD' state	
XEQn	for jobs awaiting execution in class n	
XEQ	for all jobs awaiting execution	

$$- $ -$$

Commands that alter the status of jobs

$CJn	cancel job n - this will not take immediate effect if the job is actually executing.
$HJn	'hold' job n - do not start any new activity for job n, i.e. keep it on the relevant HASP queue. Jobs should not be 'held' for long periods without informing the 370 operators of the reason.
$AJn	'allow' job n - remove it from held status.
$PJn	'drain' job (cancel after current activity ends).

In the above four commands, the 'n' may be replaced by a general job list as in the $DJ command.

$Rdev,Jn,RMr routes job 'n' output for the specified device to the specified remote. The device may be PRT, PUN[, PTP, PLT Cambridge only], or ALL meaning all device streams. [When routing to the Cambridge central printer RM255 should be used instead of RMO.]

$TJn,SAVE	puts job n into 'SAVE' state [Cambridge only]
$TJn,FREE	releases job n from 'SAVE' state [Cambridge only]

The above commands only work on jobs which 'belong' to the remote station which issues them. The owner of a job is determined by the current printer routing of the job. The central operators have control over all jobs in the host machine.

Commands that control the peripherals

Remote peripheral identifiers are of the form:-

RMr.RDn	for readers;
RMr.PRn	for printers;
RMr.PUn	for card punches;
RMr.PPn	for papertape punches [Cambridge only];
RMr.PLn	for plotters, [Cambridge only]

where 'r' is the remote workstation number, and 'n' is the logical number of the device starting from 1. In what follows, 'p' stands for such a remote peripheral identifier.

a. All devices

$Pp	'drain' peripheral p, that is, do not start anymore activity after the current job has finished with the peripheral.
$Sp	start peripheral p (the opposite of $P) - allow further jobs to use the peripheral.
$Ep	restart the current activity on peripheral p. In the case of a printer the job is returned to the print queue hence another job may be printed before the restarted one is repeated.
$Cp	cancel the job currently using peripheral p. The job is actually cancelled when the next record is read or printed.
$Np	repeat the current activity on output peripheral p by placing the output back on the queue when activity completes.
$Zp	stop the current activity and do not start anything else. $Sp may be used to continue from the point of suspension.

$$- $ -$$

b. Printers Only

$Ip	interrupt the current activity - the rest of the print output for the current job is returned to the print queue.
$Fp,n	forward space the printer by n pages, or to the end of the job if that is sooner. If n=DS the printer is forward spaced to the next dataset in the current job if any.
$Bp,n	backspace the printer with the same values of n as above.

Note that it is necessary to issue $P before $I or $E if you do not want the printer to start printing again after interrupting the current job.

c. Transfer of state

```
$Tp,C=1     where 'p' is a printer; causes all carriage control characters to be
            replaced by a single newline for the duration of this job only.
$Tp,DATA    where 'p' is a reader; causes all subsequent records in this job to be
            treated as data (including those beginning /* and //).
            [Cambridge only]
$Tp,F=nnnn set forms type for device 'p' to 'nnnn'.
            [The following forms abbreviations are also available at Cambridge:
            R reset        S standard     A auto
            N narrow plot W wide plot     O other (special) plot
            Q wide other   5 5-trk ptp    7 7-trk ptp
            8 8-trk ptp]
$Tp,S=Y     set full headers and trailers for a printer.  [Cambridge only].
      S     set full headers and single line trailers.    [Cambridge only]
      H     set headers only, no trailers.                [Cambridge only]
      N     set no headers or trailers.                   [Cambridge only]
```

Miscellaneous commands

```
$SIGNOFF    drains all active devices and terminates the datalink connection.
            [Cambridge only]
$$- $ -$$
```