G-TOR™: The Protocol

by Kantronics Staff
Mike Huslig, Phil Anderson, Karl Medcalf, and Glenn Prescott

Foreword

The G-TOR data communications protocol is an innovation of the technical staff of Kantronics Co., Inc. It was introduced as an inexpensive means of improving digital communications in the HF radio bands. A hybrid ARQ scheme, used in combination with an invertible, half-rate Golay forward error correcting code, is the single-most essential element in the protocol.

The purpose of this document is to present a detailed description of the G-TOR protocol. It is assumed that the reader is familiar with ARQ systems such as AMTOR, Pactor, and Packet; terms such as MASTER, SLAVE, ISS, IRS as they pertain to protocols; and binary, HEX and C-language number notations. Operation, performance objectives, and performance results of systems using this protocol are not discussed; these aspects of G-TOR have been covered widely in trade publications.

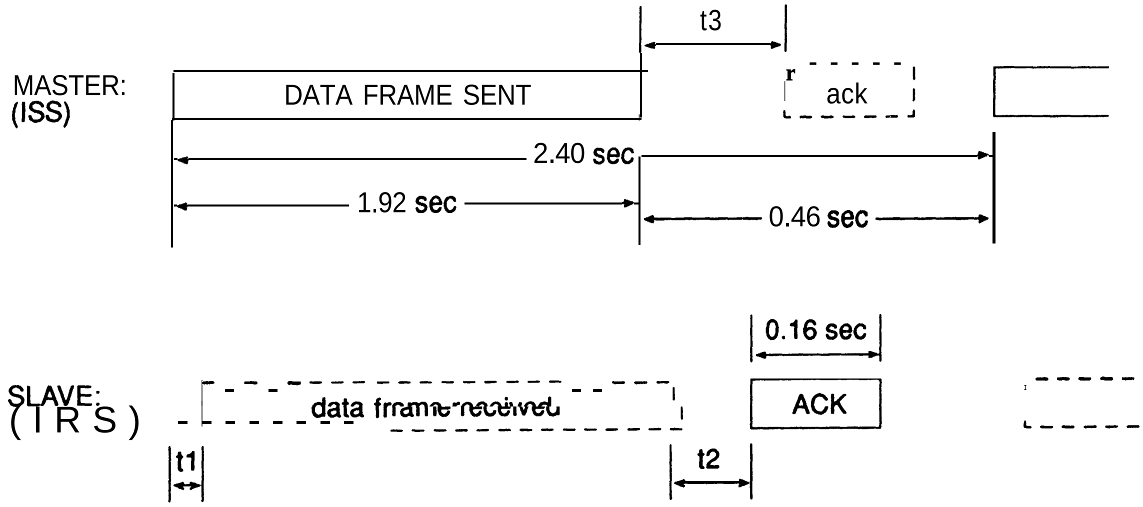
The description is organized in sections as follows: a general overview, including term definitions and initialization of parameters; timing; definition and usage of data, control, BK, and connect and disconnect frames; data formats; speed change procedures; the Huffman table; and Golay coding and data interleaving. Appendices containing flow charts, a Huffman decoding tree, and a C language routine for Golay encoding/decoding follow the protocol description.

General

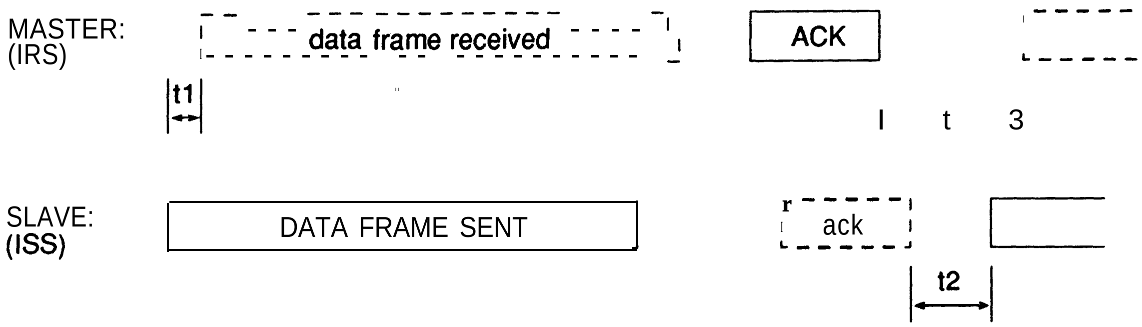
The system which originally transmits a G-TOR connect request is called the Master, and the system which responds to the transmitted connect request is called the Slave. The system currently sending data blocks is called the Information Sending Station (ISS), and the system receiving these data blocks is called the Information Receiving Station (IRS). During a connection, tlhe Master is always the Master and the Slave is always a slave, but either system may be the ISS while the other is the IRS. Immediately after a connection, the Master is the ISS while the Slave is the IRS. The IRS will send a Control Signal #1 (CS1) immediately after connection or turnaround to indicate it is ready for data; it sets an internal flag (Send_CS_flag) to CS1. The IRS also sets its internal error count to 0, its blocks_received count to 0, and its Last Block number to 0. When the ISS receives the CS1, it sets an internal flag (Expecting_CS_flag) to CS2 and Block_number to 1.

The Master and the Slave both have an internal flag (Golay_flag) which is complemented every 2.4 second cycle. During the connect process, this flag is set to be the same in both systems. Whenever the ISS receives a proper Acknowledgment (the CS1 the first time around), it forms a new frame of data (Real_Data). This new data frame is also fed through the Golay encoder to form a frame of parity bits (Golay_Data). The ISS sets an internal error count to 0. Depending on the state of the Golay_flag in the ISS, the ISS will choose the Real_Data frame or the Golay_Data frame to transmit. The Golay_flag in the ISS is then complemented for the next cycle. Whichever frame is chosen, that frame is then interleaved and transmitted.

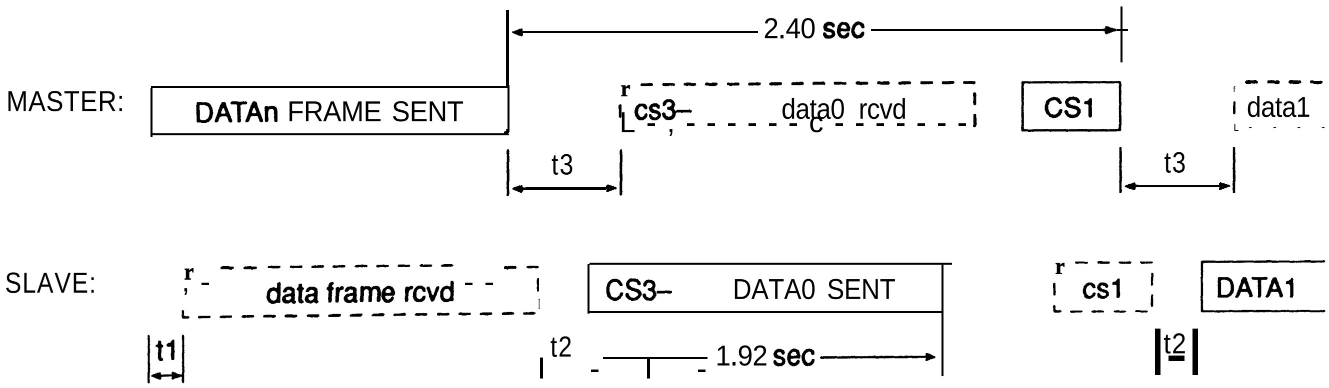
Figure 1



tl is radio wave propagation time
t2 is slave acknowledgment delay which includes processing time and transmitter turn-on delay; t2 is constant while connected, even when the Slave is the ISS
t3 is determined by the Master during initial synchronization and should vary only slightly during the connection



Changeover timing



The IRS is expecting to receive a frame during a certain time period in the 2.4 second cycle. When it has received the frame, the IRS then increments its Blocks_received count and de-interleaves the block. If the ISS Golay_flag is set, a copy of the block is saved as Golay_Data; the block is then fed through the Golay encoder to generate the original data. If the ISS Golay_flag is clear, a copy of the block is saved as Real_Data. The Golay_flag is then complemented. If the CRC of the block is correct, the IRS has received the data correctly. If the CRC is not correct, the IRS checks to see if the Blocks_received counter is greater than 1, indicating it has received a copy of both the Real_Data and the Golay_Data. If the IRS has a copy of both, it will try to regenerate the original data using Golay error correction. If the CRC is still incorrect, the IRS error count is incremented. If the error count is greater than a set maximum number of errors, the IRS will go back into a standby mode; otherwise, to indicate failure, the IRS will re-send the same Control Signal it sent in the last cycle. If the CRC of the received block or the error corrected block is correct, the IRS clears its Blocks_received count and compares the Block_number in the received frame with the Last_Block number it correctly received. If they are the same, then the received data frame is the same, indicating most likely that the ISS has not correctly received the last Control Signal sent by the IRS; the IRS then re-sends the last CS. If the Block_number is one greater than the Last_Block number received, then this block is the next data expected; the IRS now sets its Last_Block number to the Block number received and prints the data received; the IRS error count is set to 0; the Send_CS_flag is complemented and the appropriate Control Signal is transmitted. If the Block number is otherwise, then some protocol error has occurred, and data has been lost.

The ISS is expecting to receive an acknowledgment during a certain time period in the 2.4 second cycle. If the ISS receives a CS2 when it was expecting a CS2, or it receives a CS1 when it was expecting a CS1, the ISS considers the sent data to be properly acknowledged. The Expecting_CS_flag is complemented, the Block_number is incremented, and the ISS fetches new data to be transmitted. Otherwise, the data has not been acknowledged, or the ISS has not received the acknowledgment. The ISS then increments its error count, and if the error count is less than some set maximum, the ISS will try to send the data again.

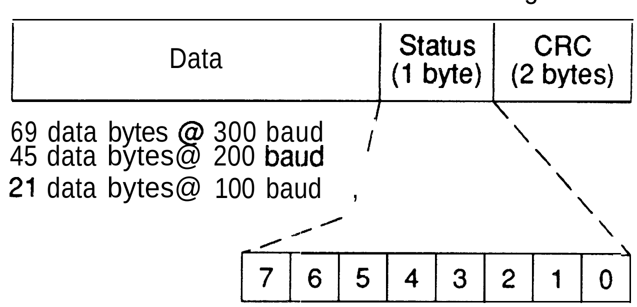
Timing

The basic G-TOR cycle is very similar to AMTOR and PACTOR. The ISS sends long data frames which are acknowledged by the IRS with shorter control signals (CS). The total cycle duration is 2.4 seconds. The data frames are 1.92 seconds long and the control signals are 0.16 seconds long. 0.32 seconds remain in the cycle for radio switching, wave propagation, and the necessary computing for both Master and Slave systems. The Master controls the total cycle time. The Slave adjusts its receive window to follow the Master's transmissions, but since the Slave's transmissions are always fixed in relation to its receive window, the Slave's transmissions follow the Master's transmissions. The Master only corrects its receive window. Refer to Figure 1.

Data Frame Structure

The frame structure for a typical G-TOR data frame (before interleaving) is shown in Figure 2. The data frame is 1.92 seconds in duration. Depending on channel conditions, data can be sent at 100,200, or 300 baud. Each data frame is composed of either 72 bytes (at 300 baud), 48 bytes (at 200 baud), or 24 bytes (at 100 baud).

Figure 2
G-TOR Frame Structure Before Interleaving



A single byte near the end of the frame is devoted to command and status functions. The status byte is interpreted as follows:

- **status bits 7 & 6**:
  Command
  - 00 – data
  - 01 – change-over request
  - 10 – disconnect
  - 11 – connect
- status bits **5 &** 4:
  Unused
  - 00 – reserved
- **status bits 3 & 2**:
  Compression
  - 00 – none
  - **01 – Huffman**
  - 10 – Swapped **Huffman**
  - 11-reserved
- status bits 1&0:
  Block Number modulo 4

The last 2 bytes of the frame contain the CRC. Like Packet and Pact-or, the CRC is computed using the same CCITT standard, starting at the first byte of a data, connect, or disconnect frame and starting at the third byte of the BK frame. However, the two bytes of the CRC are swapped before being put in the frame.

Control Signal Structure

The G-TOR Control Signals (CS) are 2 bytes (16 bits) long and are always sent at 100 baud. Each byte of the Control Signal is sent LSB first. Control Signals are used to acknowledge correct or incorrect receipt of frames from the information sending station. They are also used to request changes in transmission speed and to initiate a change-over in information flow direction. There are five different G-TOR Control Signals:

Signal-Function	Code	Bit pattern in time
CS1-Data ack/nack	**F11A**	1000111101011000
CS2-Data ack/nack	**6B62**	1101011001000110
CS3-Change-over command	**5E13**	0111101011001000
CS4-Speed change	**4D3C**	1011001000111100
CS5-Speed change	8957	1001000111101010

The CS codes are composed of multiple cyclic shifts of a single **15-bit** pseudo-random noise (PN) sequence (an extra '0' bit is appended to the sequence for balance, so the total CS word length is 16). A pseudo-random noise sequence is used because PN sequences have **powerful** mathematical correlation and distance properties which facilitate the identification of the appropriate CS code, even in the presence of noise and interference. Each CS has a mutual Hamming distance of 8.

BK Frame Structure

The change-over frame is shown in Figure 3. This frame is always transmitted at 100 baud and is never interleaved. It is essentially a combination of the CS3 Control Signal and a shortened data frame. Each byte of the BK frame is sent LSB **first.**

Figure 3
G-TOR Changeover Frame Structure

← 1.92 **sec** →

cs3 (2 bytes)	Data (19 bytes)	Status (1 byte)	CRC (2 bytes)

Formation of Connect and Disconnect Frames

Connect and Disconnect frames are always sent at 100 baud (24 bytes). The first 10 bytes contain the call/address of the destination and the second 10 bytes contain the call/address of the source. These 20 bytes use 7 bit ASCII. If the call/addresses are less than 10 bytes long, the fill character **0x0F** should be used to extend the addresses to 10 bytes.

The 21st byte is zero. Bytes 23 and 24 are the CRC. Byte 22 is the status byte and will be 11000000 for a connect frame or **100000xx** for a disconnect frame. Note that the Block Number for a connect frame is always 0.

The MSB of the first 20 bytes are originally zero because of the use of 7 bit ASCII. Bytes 2,

Figure 4

```
47 54 4F 52 54 4F 43 41 4C 4C 4D 59 43 41 4C 4C OF OF OF OF 00 CO ?? ??
   \/       \/       \/       \/       \/       \/       \/
                                    becomes
   /\       /\       /\       /\       /\       /\       /\
47 4D 4F 52 4D 4F 43 1C 4C 4C DC 59 43 1C 4C 4C F8 OF OF F8 00 CO F5 E4
```

5, 8, 11, 14, 17, and 20 should now have their MSB set to one; then the nibbles of these bytes should be swapped. For example, a connect frame to GTORTOCALL from MYCALL would form as shown in Figure 4.

The reason for this strange format is that when the frame is broken up into 12 bit tribbles and sent to the interleaver,

474 D4F 524 D4F 431 C4C 4CD C59
431 C4C 4CF 80F 0FF 800 C0F 5E4

the first 14 bits transmitted (the MSBs of the tribbles) will be alternating 0s and 1s. Note that this pattern is not present when the Golay form of the frame is being sent.

The Slave should also look for connect frames with mark and space inverted, and the Master should also look for inverted Control Signals. Once connected, each station should remember its received polarity.

When the Slave decodes a connect frame addressed to it, the Slave would normally answer with a CS1 control signal. If the Slave is busy, it would answer with a CS2. If the 21st byte is not zero, or the 6 lower bits of the status byte are not zero, the Slave should answer with a CS5; this is for future expansion – the Master indicating it has added capabilities, the Slave indicating it does not yet support those capabilities.

The Slave must be careful about 'when' it acks the Master. Like Amtor and Pactor, the Slave sets a fixed time after the Master's transmission for its own transmission. For maximum propagation, the Slave should set this time as short as possible. However, the time should be long enough so that it can not only decode and possibly correct a data frame before sending the ack as an IRS, but also long enough to form a data frame when an ack is received from the ISS. In other words, the Slave must be aware of the time needed for its own processing.

The connect and disconnect frames are always sent at 100 baud. If the ISS wants to disconnect but is transmitting at a higher baud rate, it should send an idle frame with a status byte 100000xx; when the IRS sees this frame, it should send a CS5 to downspeed the ISS but should stay connected until the ISS sends a true disconnect frame.

After the IRS acknowledges a disconnect frame, it should remember the time relationship between the disconnect frame and the IRSs ack. If the ISS did not copy the ack, it will keep sending disconnect frames until it times out. If the IRS copies a disconnect frame to it while in standby, it should re-send the last ack.

Data Format in Frames

The ISS can send data in three forms: straight ASCII, Huffman compressed, and swapped Huffman compressed. Swapped Huffman uses the same tables as Huffman compressed but swaps the upper case letters with the lower case. Since Huffman compressed favors lower case letters as in normal text, Swapped Huffman favors upper case letters in text which may be predominately upper case. The ISS must decide in which form to send the data in order to provide the greatest throughput; if there is no advantage in sending Huffman codes, the ISS should send in straight ASCII. All normal data frames and connect and disconnect frames are interleaved and, on alternate cycles, Golay encoded.

If there is not enough data to send in a data frame, IDLE codes are used to fill the frame. If the frame is sending straight ASCII, 0x1E is used as the IDLE code. In order to send a 0x1E data byte, a 0x1C pass code must be sent followed by 0x7E; in order to send a 0x1C data character, a 0x1C pass code must be sent followed by 0x7C. Only the ASCII data characters 0x1C and 0x1E need a pass code. The pass code should never be the last character in an

ASCII data frame; in other words, the combinations 0x1C 0x7E and 0x1C 0x7C should never be split between data frames. G-TOR Huffman compression uses a unique IDLE code; there is no pass code when sending a Huffman compressed frame.

The IDLE code also indicates the end of data in a data frame: straight ASCII or Huffman compressed. The IRS should stop decoding the data frame when it encounters an IDLE code, and the ISS should never send data after an IDLE code in a data frame. This function is reserved for possible expansion.

BK Frames

If the IRS wants to send data to the ISS, it can seize the link and become the ISS by sending a BK frame. The BK frame is a special data frame which is always sent at 100 baud and is never interleaved nor Golay encoded. The first 16 bits of the BK frame comprise the CS3 control signal. The next 22 bytes are 19 bytes of data plus the status byte and 2 byte CRC formed over the data starting after the CS3. The Block Number in the status byte of the BK frame is always 0. Each byte is sent LSB first. If the ISS receives the BK frame correctly, it sends a CS1 and becomes the new IRS, expecting new data frames at the previous baud rate. If the ISS detects the CS3 but does not receive the data correctly, it sends a CS2 and becomes the new IRS, still expecting data at the previous baud rate. If the original sender of the BK frame received a CS2, it will re-form a data frame using the old data that was used in the BK frame plus any additional data available, but again at the baud rate in use before the BK frame was sent. If the sender of the BK frame receives neither a CS1, CS2, or CS3, it will re-send the original BK frame.

Since there is a possibility that the ISS does not receive the CS3 part of the BK frame and therefore will re-send a data frame or the Golay encoded form of the data frame, the ISS must ensure that any data frame or Golay encoded form of a data frame will not produce a waveform which would appear as a 100 baud CS1, CS2, or CS3 in the time slot where the IRS may be looking for an acknowledgment to its BK frame. The IRS should be sampling in the receive ack time slot at the previous baud rate to ensure that the ack received is truly a 100 baud signal and not an artifact of the ISS data frame at a higher speed.

The ISS can request a changeover by sending a data frame with bit 6 of the status byte (BK request bit) set to 1 (0100xxxx); the IRS would then send a BK frame. A BK frame can also be acknowledged with another BK frame, causing quick changeovers. The BK frame serves as a positive acknowledgment of the previously received data.

Changing Speed

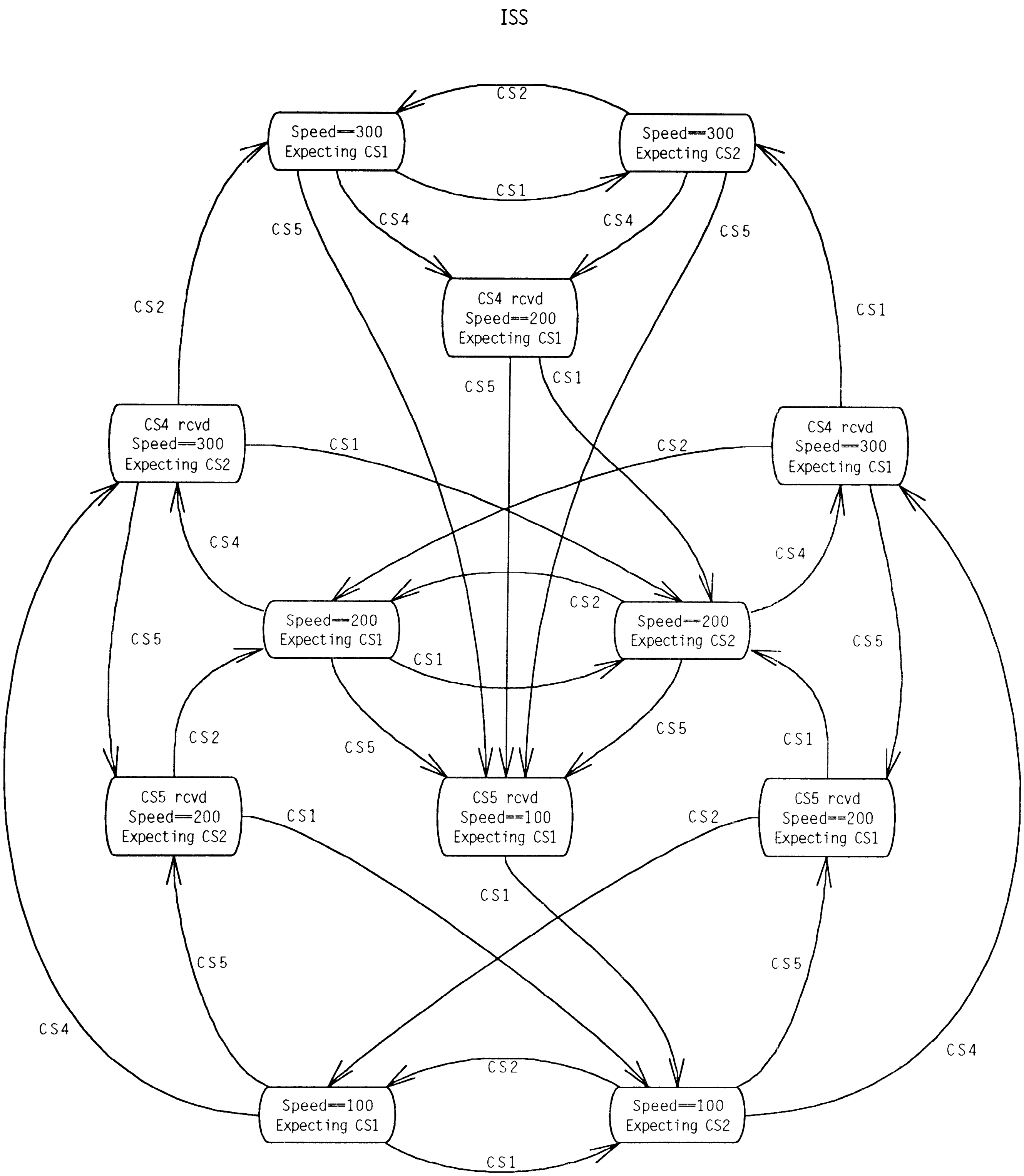
Data frames can be sent at 100,200, or 300 baud. CS4 and CS5 are the Control Signals that the IRS uses to change the sending speed of the ISS. Since the IRS can cause the ISS to change from any one speed to any other speed, the Control Signal used by the IRS depends on the states of the two systems. Refer to the Speed Transition Diagram in Figure 5. The algorithm used by the IRS to determine speed changes is not a part of this protocol. The algorithm used by the KAM, however, is shown in the flowcharts. A speed-up CS always acts as a positive acknowledgment of the previous data frame. A speed-down CS asks for the previous data to be re-sent at a slower speed.

Slowdowns and BK Frames

If the ISS receives a slowdown signal from the IRS, it has no way of knowing whether the data just sent was received correctly or not and therefore should re-send the data at the requested slower speed using the same block number. It is possible that the IRS could request a further slowdown in speed while the ISS is re-sending data. Any time the IRS receives valid data, it should keep a count of the characters in the frame. If the IRS slows the ISS down and the new data frame received has the same block count as the previous frame, the IRS knows the ISS is resending data and should throw away the appropriate number of characters. The ISS and IRS need to be careful with these character counts during double slowdowns (from 300 to 200 and then from 200 to 100 baud).

If the IRS tells the ISS to slow down after the ISS has sent a data frame with the BK request bit set, or if the ISS decides it wants to send a

Speed transition diagram
Figure5



BK request while re-sending data in response to a slowdown, the ISS should not set the BK request bit in the slower data frames until the data frame contains the last character sent in the original.

The IRS cannot send a BK frame until it receives a valid data frame since the CS3 of the BK frame is an acknowledgment of valid data. If the IRS is receiving duplicated data due to a slowdown, it should not send a BK frame until all the duplicated data is received.

RLEn Coding

An RLEn code is a 19 bit code made up of a unique 14 bit Huffman code followed by 5 bits which represent a number n, O-31. RLEn codes are found only in Huffman compressed data frames and can never be the first code in a data frame.

When an RLEn code is encountered in a data frame, the previous character decoded in the frame should be repeated an additional N times, where N is a number which depends on n and the number of bits used by the previous Huffman character according to the following table.

length of previous character	N
2 bits	n+10
3 bits	n+7
4 bits	n+5
5-6 bits	n+4
7-9 bits	n+3
10-16 bits	n+2

An RLEn code may follow another RLEn code immediately, indicating that the previous code, which was just repeated, should be repeated an additional N times.

Huffman codes are put into the data fields in the order shown in Appendix 11. For example, the first few bytes of "The quick brown fox" using Huffman compression would be formed as shown in Figure 6.

And before interleaving or Golay encoding, the bytes are grouped into tribbles

1A2 3BD 6FE A65 . . . .

Golay Coding and Interleaving

Before a data frame is transmitted, the data is regrouped into 12 bit tribbles. For example, a 100 baud frame of "The quick brown fox" using no compression would be formed like:

```
54 68 65 20 71 75 69 63 6B
20 62 72 6F 77 6E 20 66 6F
78 1E 1E 01 7E 64
```

And then grouped into tribbles

```
546 865 207 175  696 36B
206 272 6F7 76E  206 66F
781 E1E  017 E64.
```

The data is interleaved by sending in time the MSB of each tribble, and then the next MSB, etc. The bit sequence of the above data would start:

```
  time->
0100000000000101
1000100011011101
0010111111111101
1001010001001000
... 8 more groups of 12 bits
```

The ISS alternately sends frames of data and Golay encoded data. Golay codes are unique 1%bit codes derived from 12 bits of data. The C program in Appendix 10 shows how to generate the codes from the data and also how to regenerate the correct data from the 24 bits of data and Golay codes which have errors. The correction algorithm will correct up to 3 bits in error from the 24 bits of data and encoded data. The Golay codes are generated from the

Figure 6

```
   T       h     e        q         u     i      c       k
0001101 000100 011  10 1111010110 11111 1101 010011 0010101
00011010  00100011  10111101 01101111 11101010 01100101 01......
   1A        23        BD        6F        EA       65
```

tribbles of data before interleaving, so that "The quick brown fox"

```
546 865 207 175 696 36B 206 272
6F7 76E 206 66F 781 E1E 017 E64
```

becomes

```
083 092 57B 1A7 F88 C46 A85 AF1
9AE 342 A85 291 114 BAF 0B1 3F0.
```

The tribbles are then interleaved as before, starting with the MSB of the first tribble.

Note that the CRC of the original data is also Golay encoded; there is no CRC generated over the Golay encoded frame.

Note also that the inverse Golay function is identical to the Golay function; in other words, $x=g(g(x))$.

FEC Transmissions

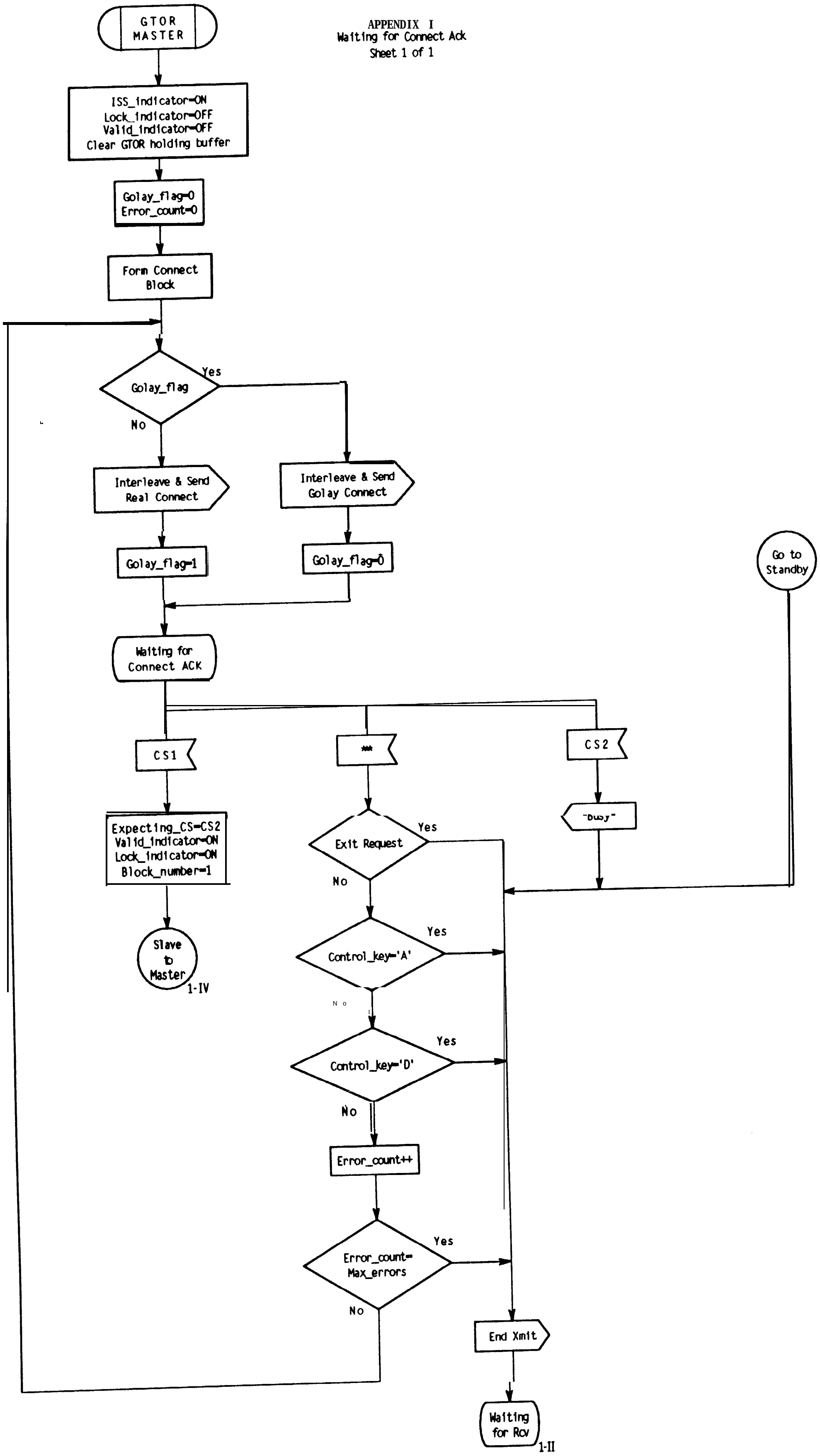
At this time there is no special G-TOR broadcasting mode. AMTOR mode B is used for calling CQ. A G-TOR unit in standby should be able to receive AMTOR mode B FEC signals.

Monitoring G-TOR

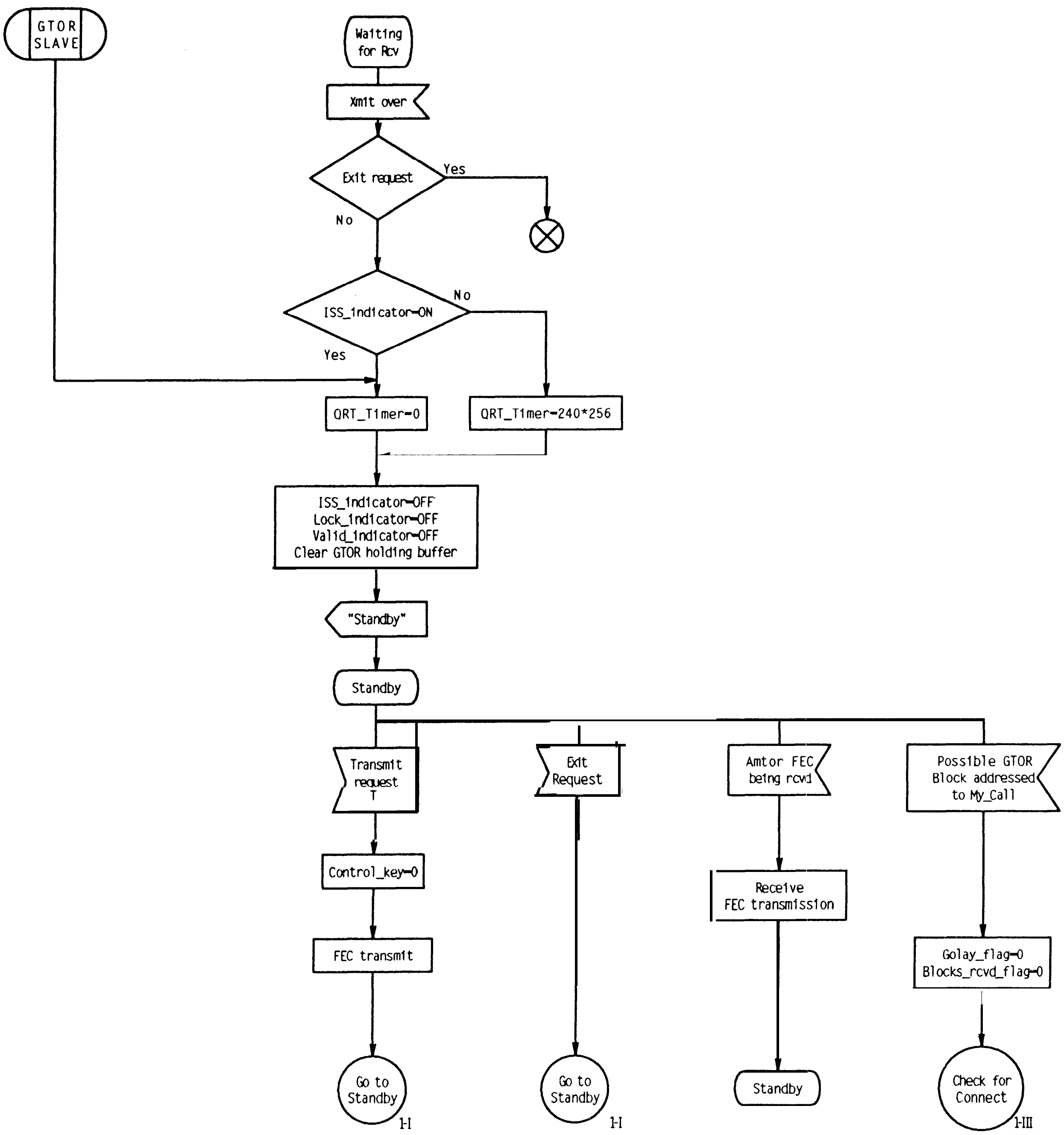
Third party monitoring of G-TOR connects can be very difficult due to the nature of the G-TOR protocol. Although a data frame is always 1.92 seconds long, it may have been sent at 100,200, or 300 baud. The frame received may be the Golay encoded form of a data frame. The BK frame is different in that it is not interleaved, and its CRC is calculated over shortened data. The frame received could also be inverted polarity; however, the inversion would stay the same during any particular connection. Since the Golay error correction allows the IRS to copy data without ever getting a proper CRC in one frame, a monitor program should also go back 2.4 seconds to form a correct frame if it is to be thorough. Again, because of the nature of the G-TOR signal, Carrier Detect or a PLL on bit transitions cannot be used reliably, but a brute force algorithm can be used. It would sample the data stream at twice the baud rate for 100,200, and 300 baud. Sampling at twice the baud rate will take care of problems caused by sampling near the edge of a bit. A program was written to do a brute force algorithm using the fastest assembly language techniques to check for all possible G-TOR frames; the program ended up using about 1/3 of the available cycles of a 50 MHz 486DX.

G-TOR is a trademark of Kantronics Co., Inc.

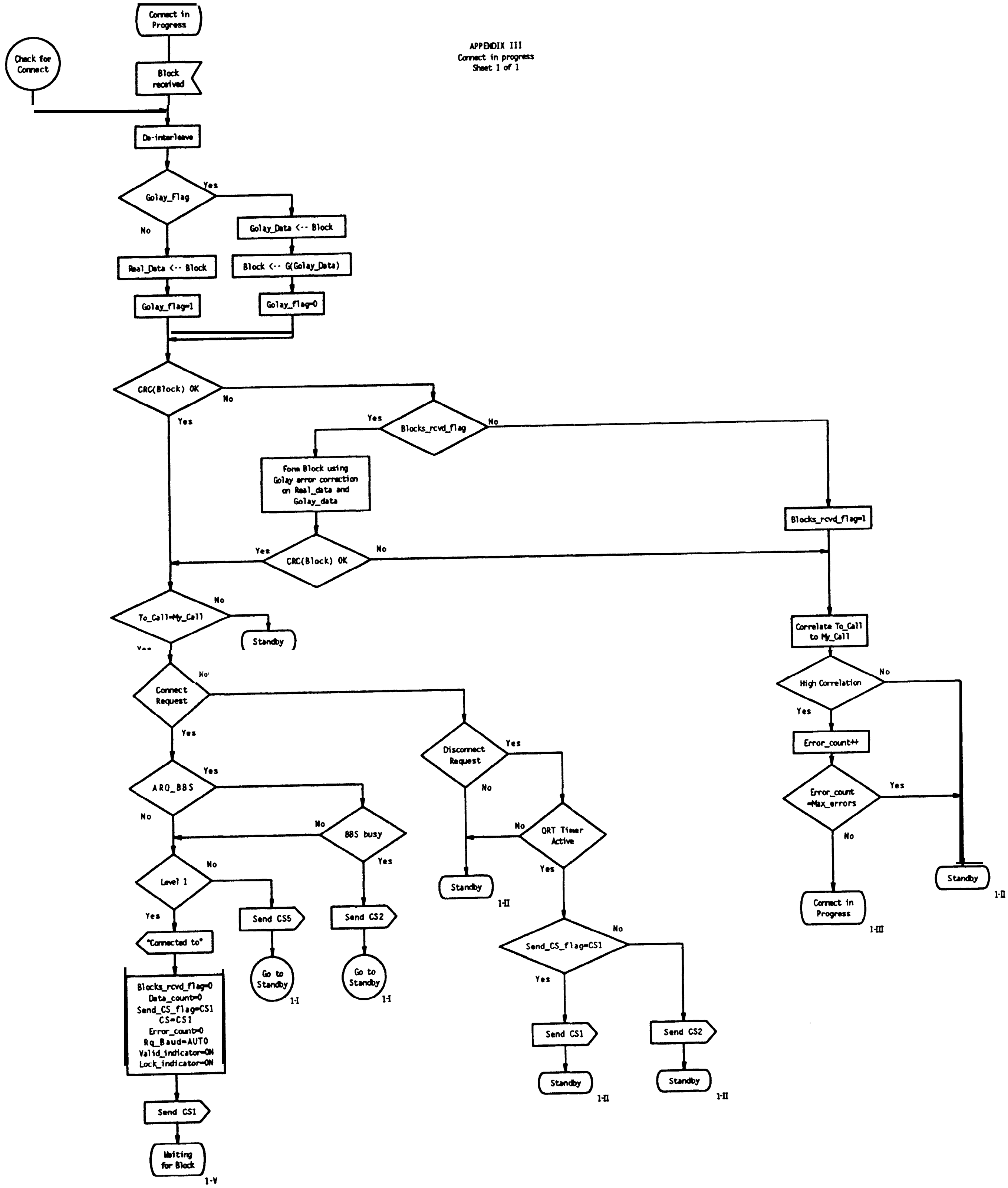
APPENDIX I
Waiting for Connect Ack
Sheet 1 of 1



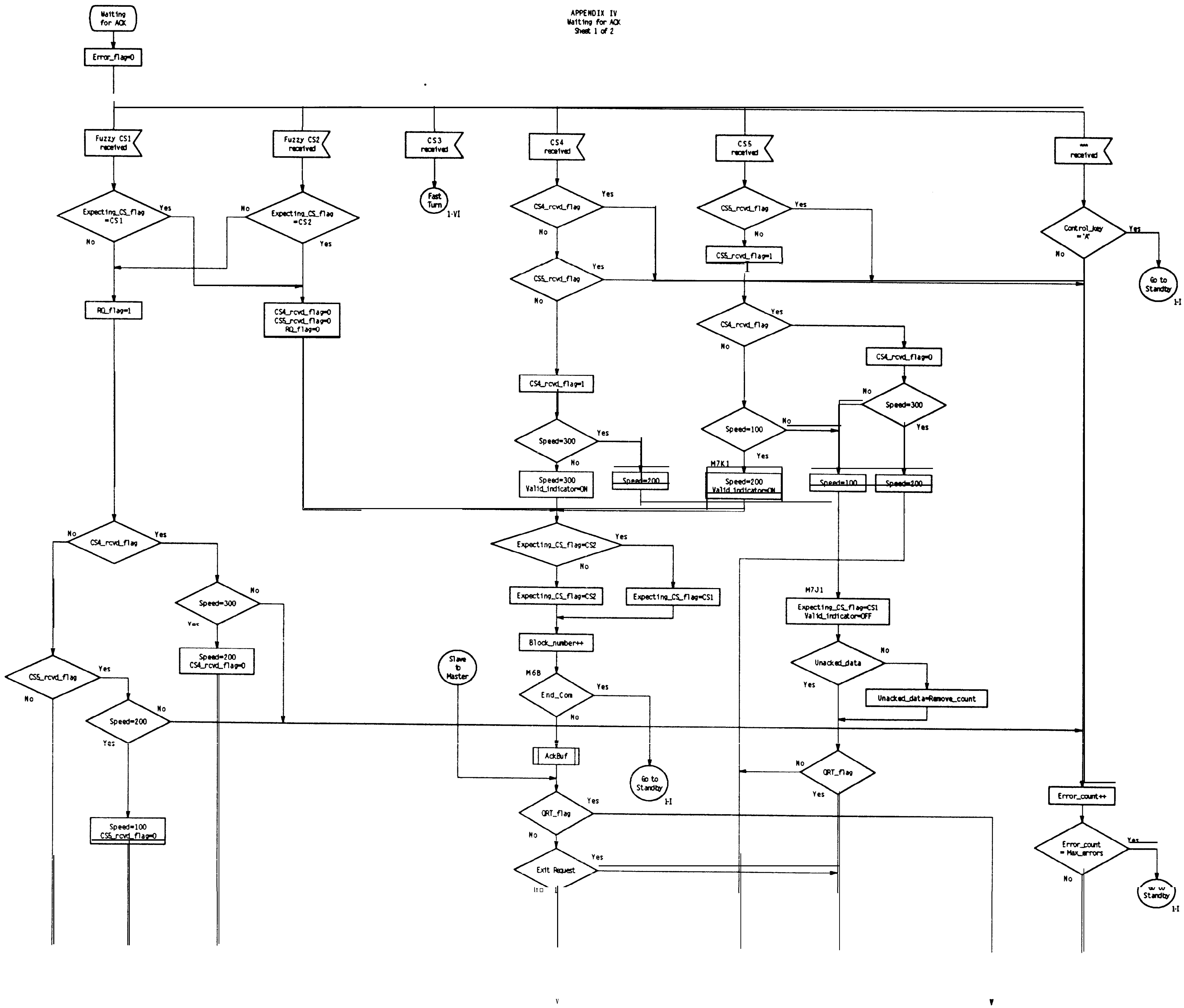
APPENDIX I I
Standby/Waiting for receive
Sheet 1 of 1

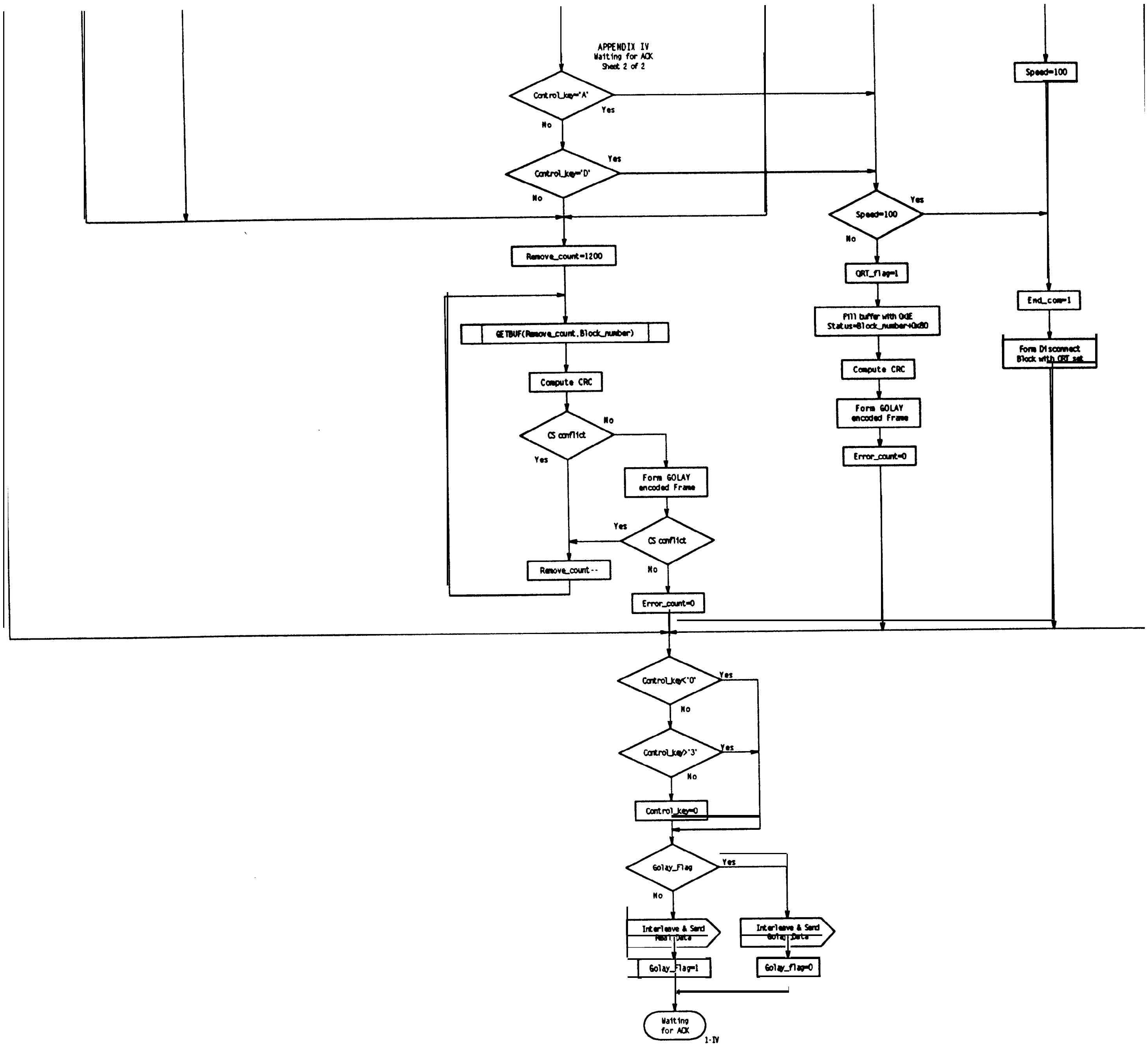


APPENDIX III
Connect in progress
Sheet 1 of 1

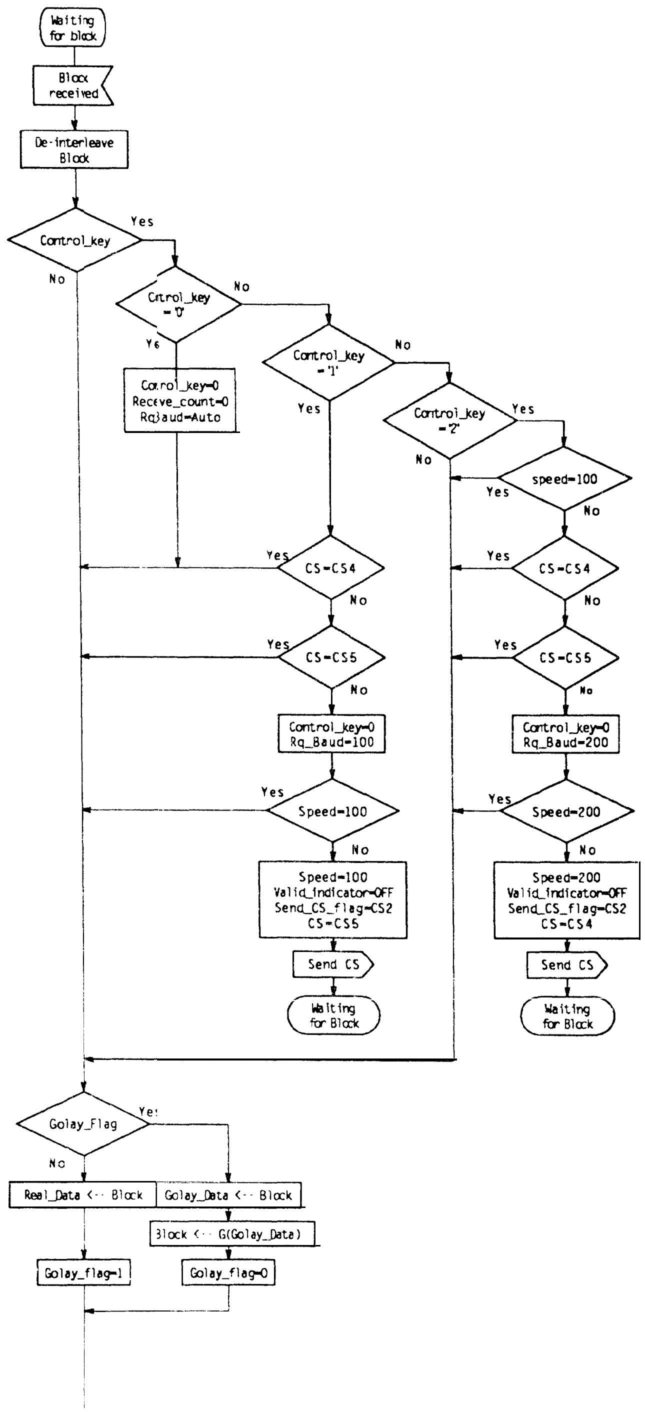


APPENDIX IV
Waiting for ACK
Sheet 1 of 2

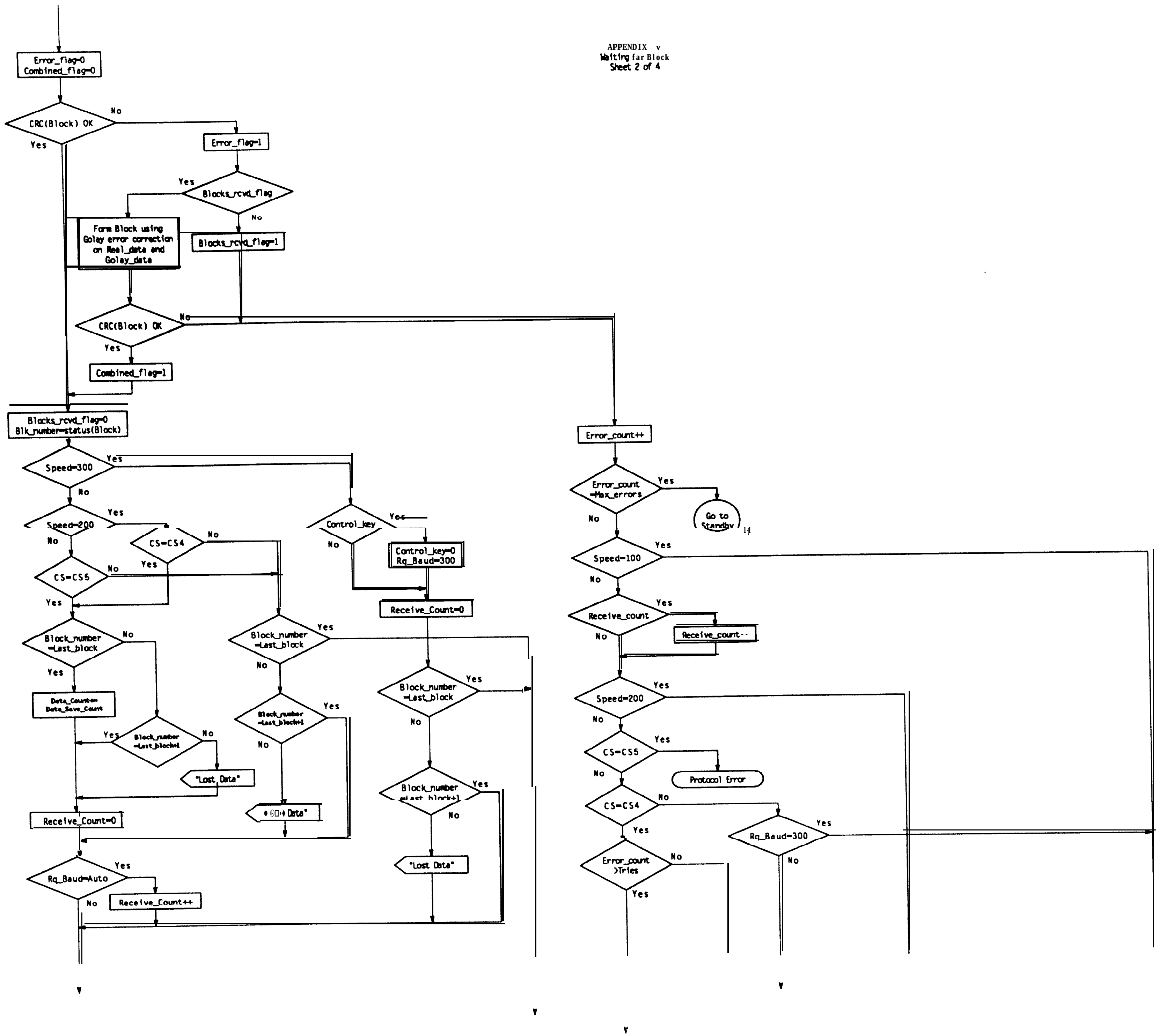


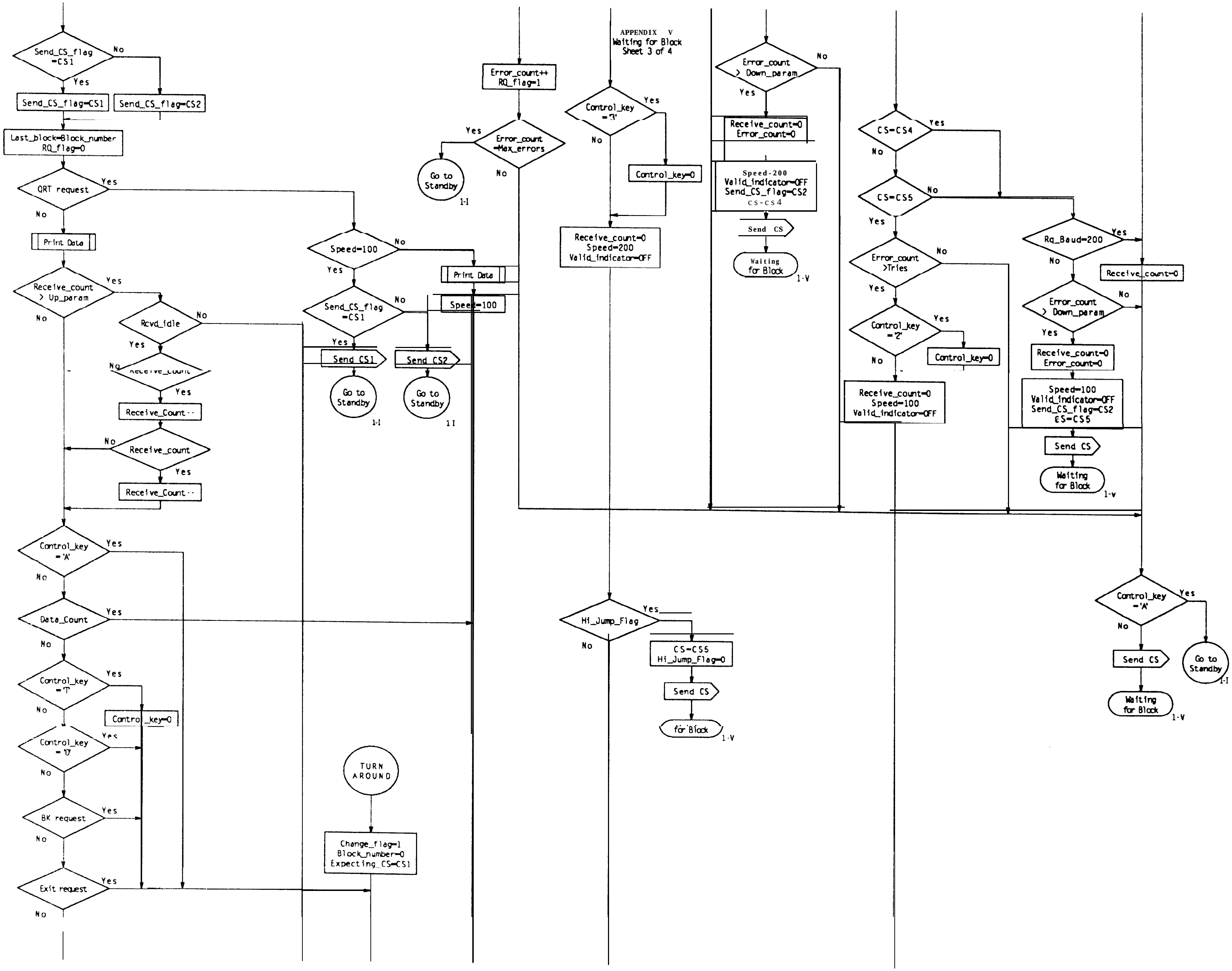

APPENDIX IV
Waiting for ACK
Sheet 2 of 2
Control_key='A'
Yes
No
Control_key='D'
Yes
No
Speed=100
Speed=100
Yes
No
Remove_count=1200
QRT_flag=1
End_com=1
Fill buffer with 0x0E
Status=Block_number+0x80
Form Disconnect
Block with QRT set
GETBUF(Remove_count.Block_number)
Compute CRC
Compute CRC
Form GOLAY
encoded Frame
CS conflict
No
Yes
Form GOLAY
encoded Frame
Error_count=0
Yes
CS conflict
Remove_count--
No
Error_count=0
Control_key<'0'
Yes
No
Control_key>'3'
Yes
No
Control_key=0
Golay_Flag
Yes
No
Interleave & Send
Interleave & Send
Golay_Flag=1
Golay_flag=0
Waiting
for ACK
1-IV

APPENDIX V
Waiting for Block
Sheet 1 of 4

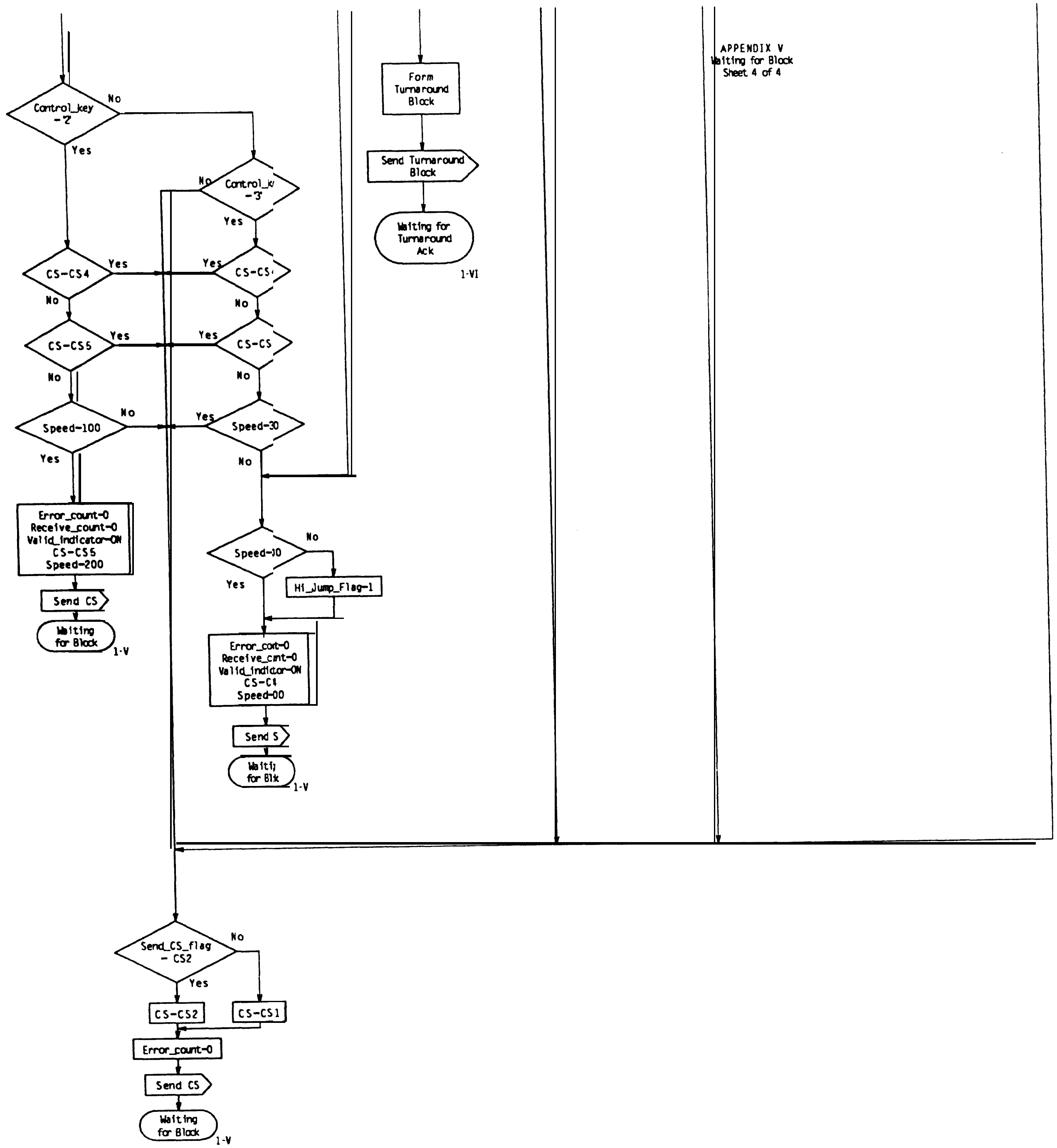


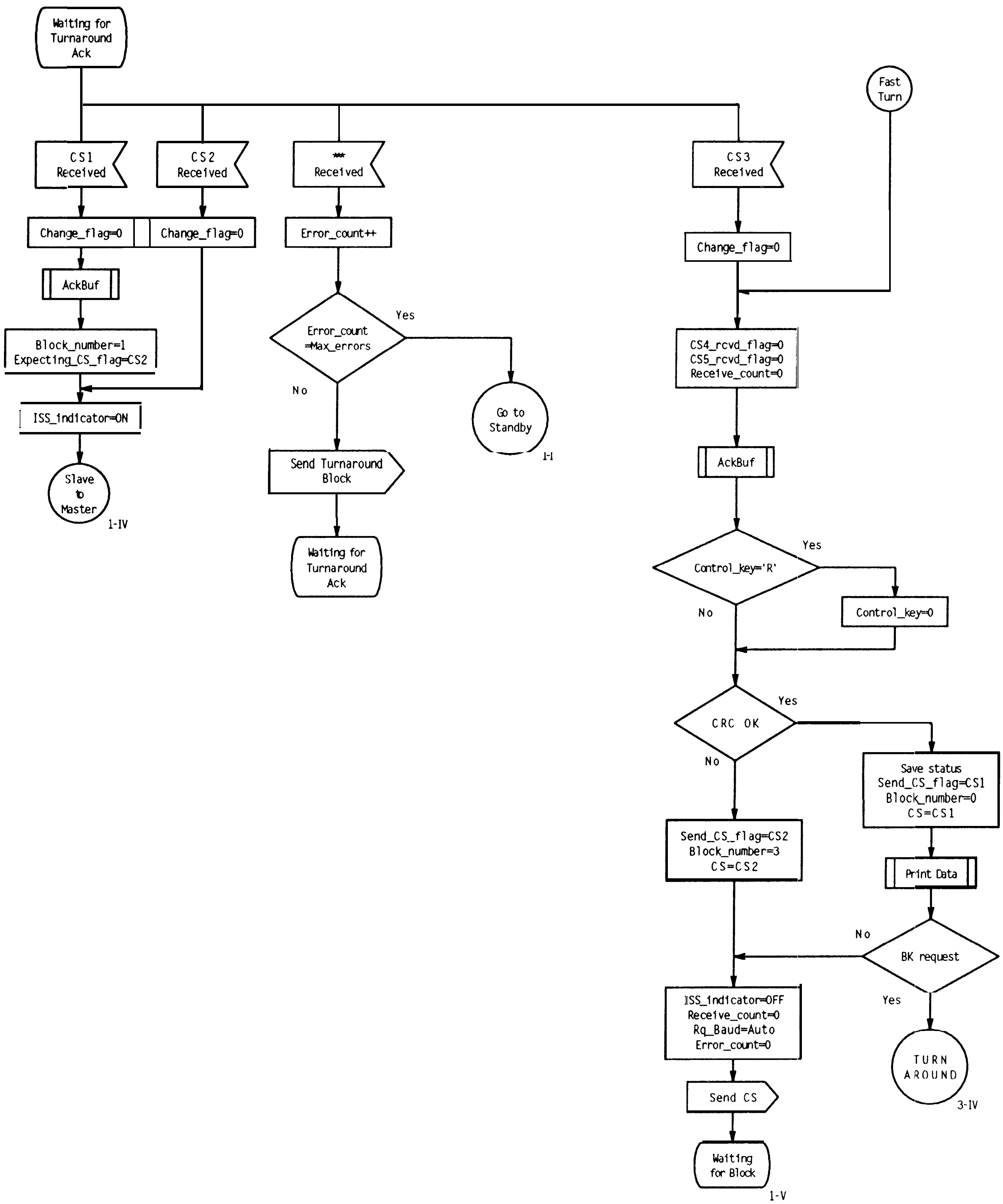
APPENDIX v
Waiting far Block
Sheet 2 of 4



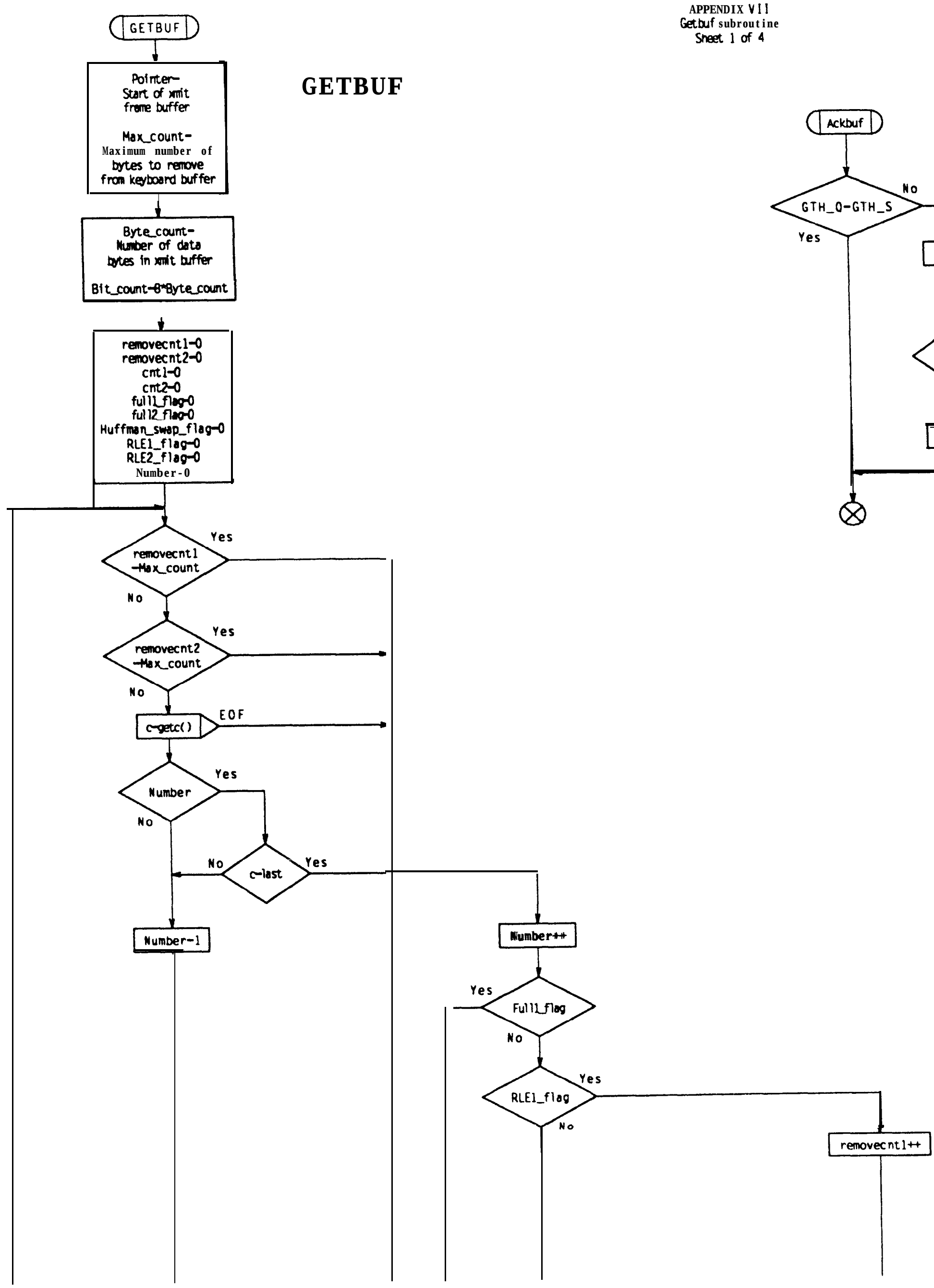

APPENDIX V
Waiting for Block
Sheet 3 of 4
Send_CS_flag =CS1
Yes
No
Send_CS_flag=CS1
Send_CS_flag=CS2
Last_block=Block_number
RQ_flag=0
QRT request
Print Data
Receive_count > Up_param
Rcvd_idle
Receive_Count--
Receive_count
Control_key ='A'
Data_Count
Control_key ='T'
Control_key=0
Control_key ='U'
BK request
Exit request
Speed=100
Send_CS_flag =CS1
Send CS1
Send CS2
Go to Standby
1-I
TURN AROUND
Change_flag=1
Block_number=0
Expecting_CS=CS1
Error_count++
RQ_flag=1
Error_count =Max_errors
Speed=100
Control_key ='3'
Receive_count=0
Speed=200
Valid_indicator=OFF
Hi_Jump_Flag
CS=CS5
Hi_Jump_Flag=0
Send CS
for Block
1-V
Error_count > Down_param
Receive_count=0
Error_count=0
Speed-200
Valid_indicator=OFF
Send_CS_flag=CS2
CS-CS4
Waiting for Block
CS=CS4
CS=CS5
Error_count >Tries
Control_key ='2'
Speed=100
Rq_Baud=200
Receive_count=0
Error_count > Down_param
Send_CS_flag=CS2
CS=CS5

APPENDIX V
Waiting for Block
Sheet 4 of 4

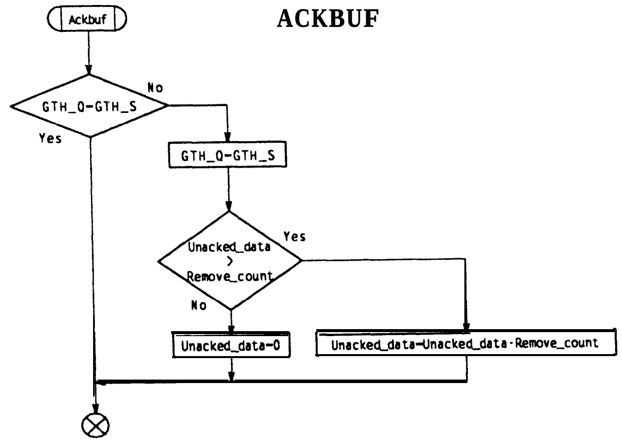


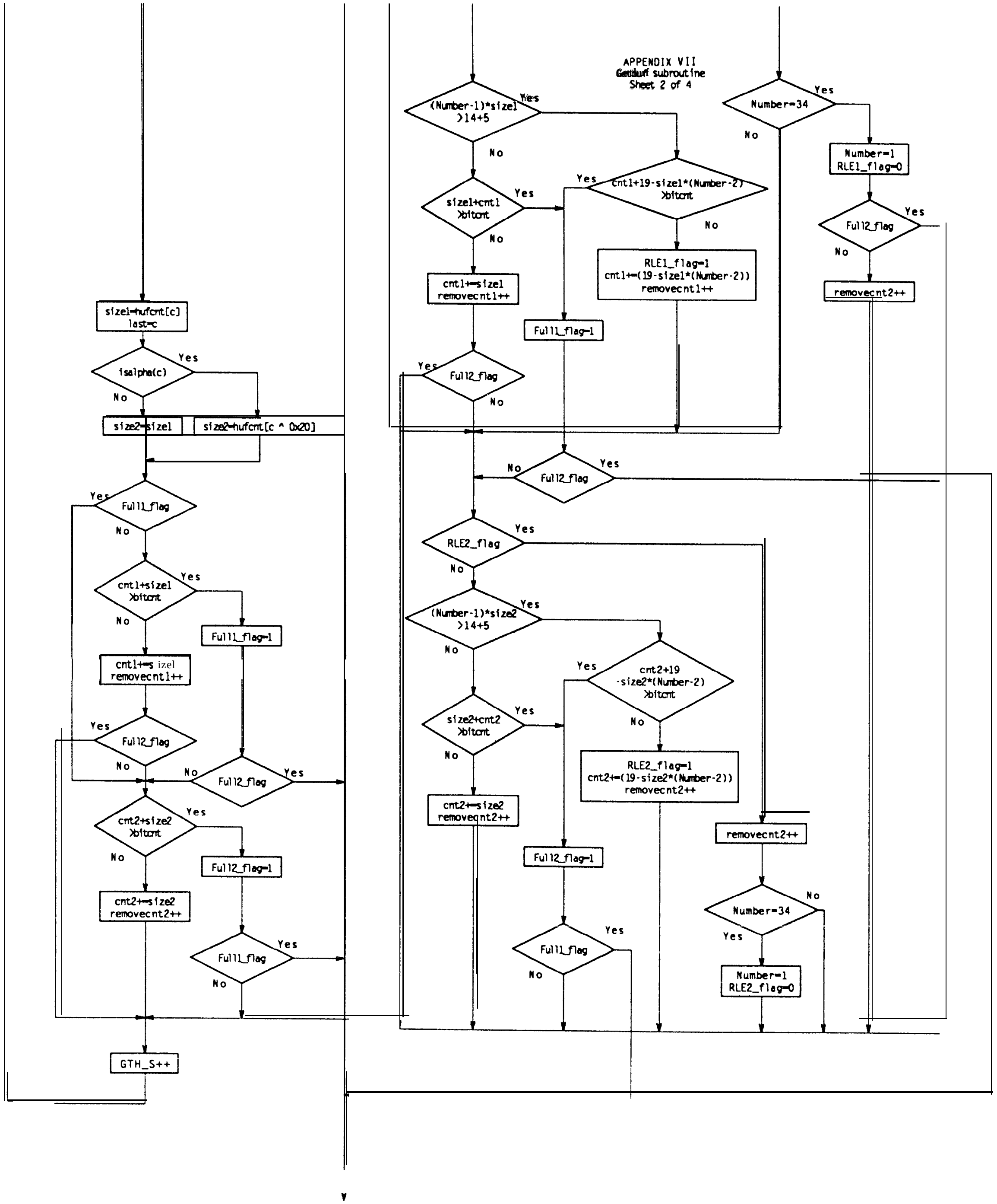

Waiting for Turnaround Ack
Fast Turn
CS1 Received
CS2 Received
*** Received
CS3 Received
Change_flag=0
Change_flag=0
Error_count++
Change_flag=0
AckBuf
Error_count =Max_errors
Yes
No
Block_number=1
Expecting_CS_flag=CS2
CS4_rcvd_flag=0
CS5_rcvd_flag=0
Receive_count=0
ISS_indicator=ON
Go to Standby
1-I
AckBuf
Send Turnaround Block
Slave to Master
1-IV
Waiting for Turnaround Ack
Control_key='R'
Yes
No
Control_key=0
CRC OK
Yes
No
Save status
Send_CS_flag=CS1
Block_number=0
CS=CS1
Send_CS_flag=CS2
Block_number=3
CS=CS2
Print Data
BK request
No
Yes
ISS_indicator=OFF
Receive_count=0
Rq_Baud=Auto
Error_count=0
TURN AROUND
3-IV
Send CS
Waiting for Block
1-V

GETBUF

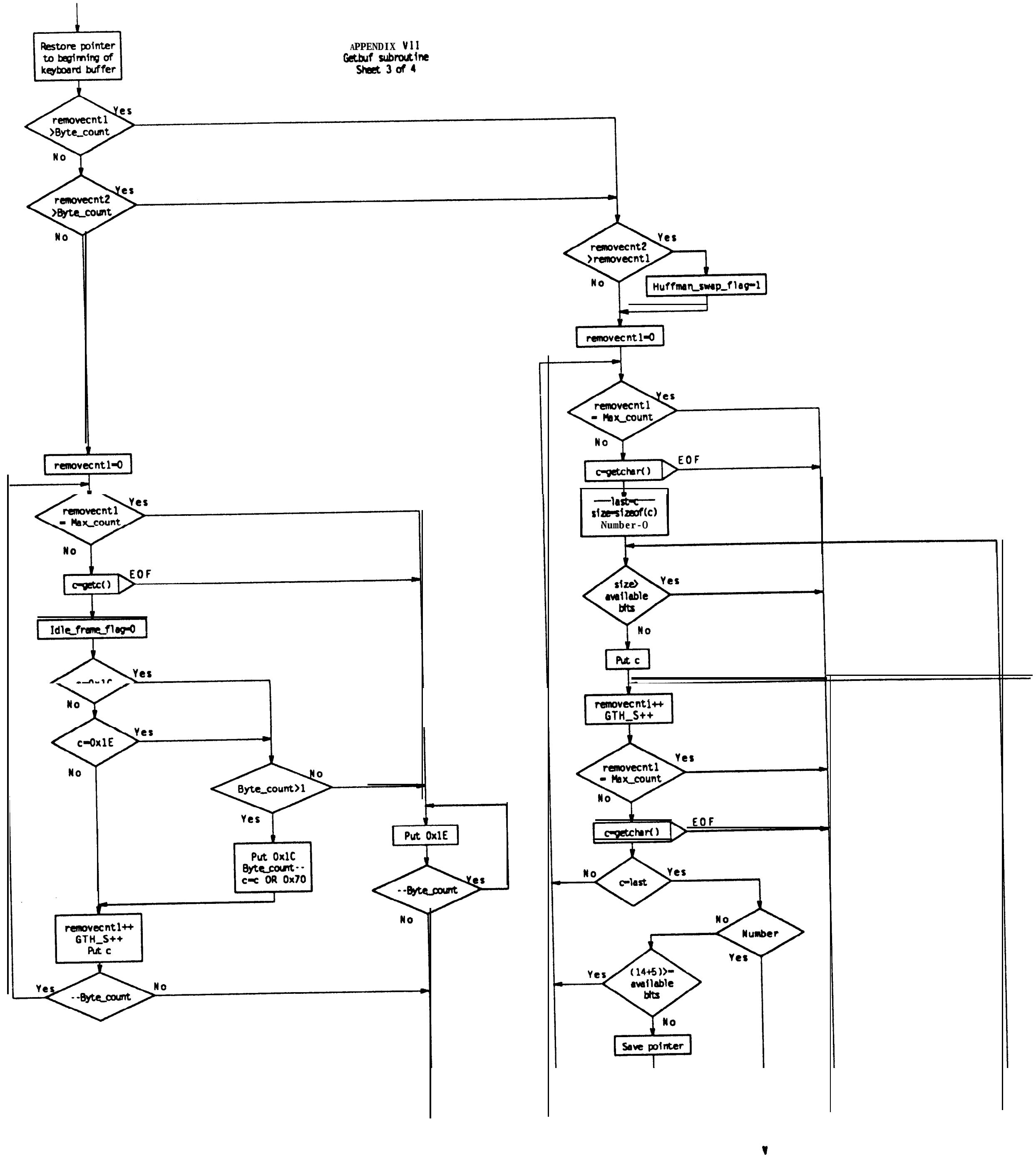


ACKBUF

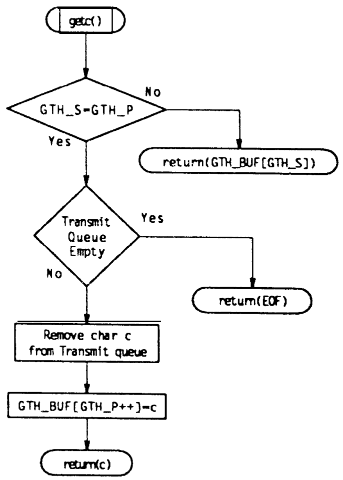


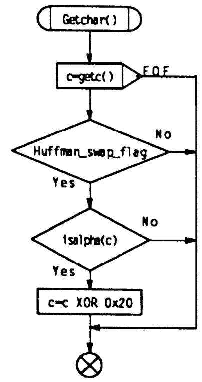

APPENDIX VII
Getbuf subroutine
Sheet 2 of 4
size1=hufcnt[c]
last=c
isalpha(c)
Yes
No
size2=size1
size2=hufcnt[c ^ 0x20]
Full1_flag
cnt1+size1 >bitcnt
Full1_flag=1
cnt1+=size1
removecnt1++
Full2_flag
cnt2+size2 >bitcnt
Full2_flag=1
cnt2+=size2
removecnt2++
GTH_S++
(Number-1)*size1 >14+5
size1+cnt1 >bitcnt
cnt1+19-size1*(Number-2) >bitcnt
RLE1_flag=1
cnt1+=(19-size1*(Number-2))
removecnt1++
Number=34
Number=1
RLE1_flag=0
RLE2_flag
(Number-1)*size2 >14+5
size2+cnt2 >bitcnt
cnt2+19 -size2*(Number-2) >bitcnt
RLE2_flag=1
cnt2+=(19-size2*(Number-2))
removecnt2++
RLE2_flag=0

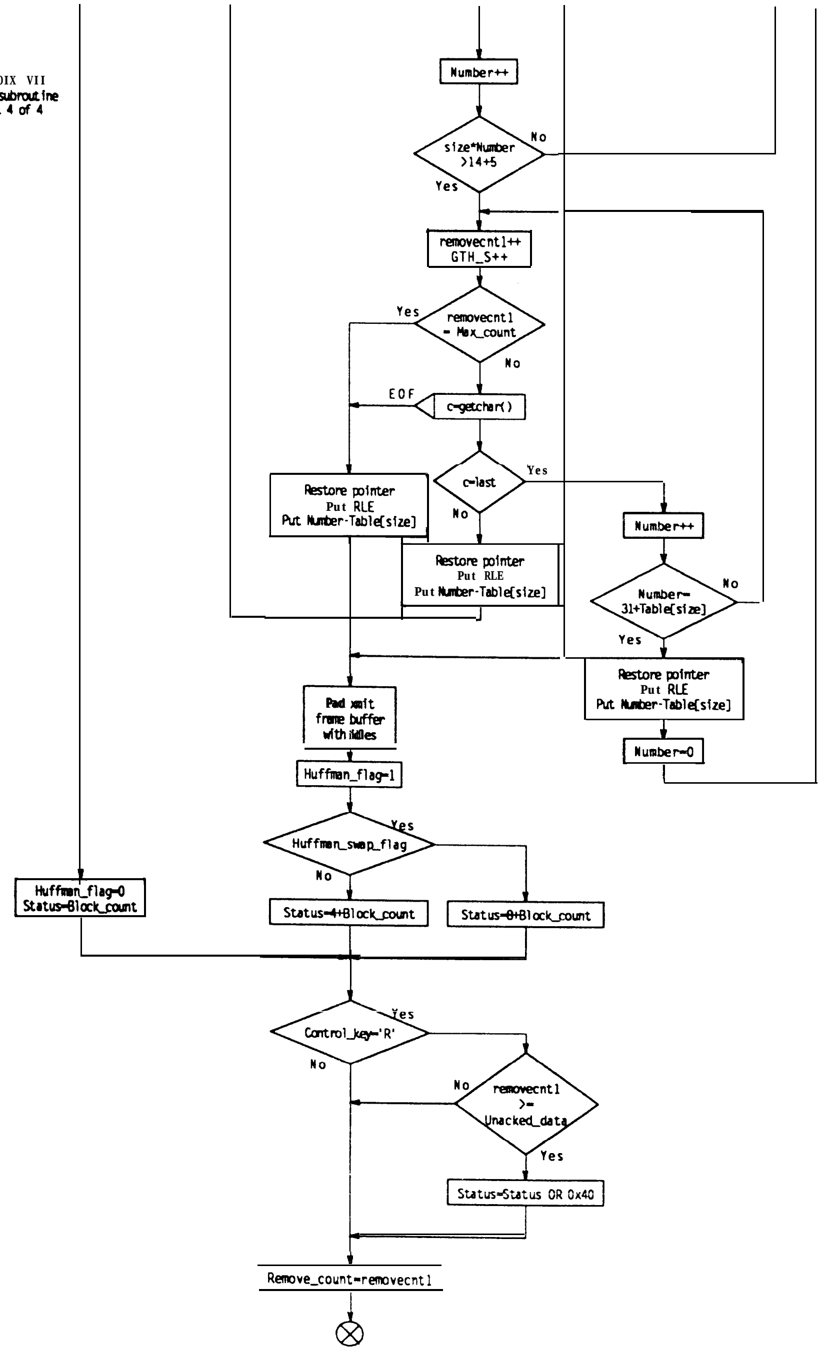
APPENDIX VII
Getbuf subroutine
Sheet 3 of 4



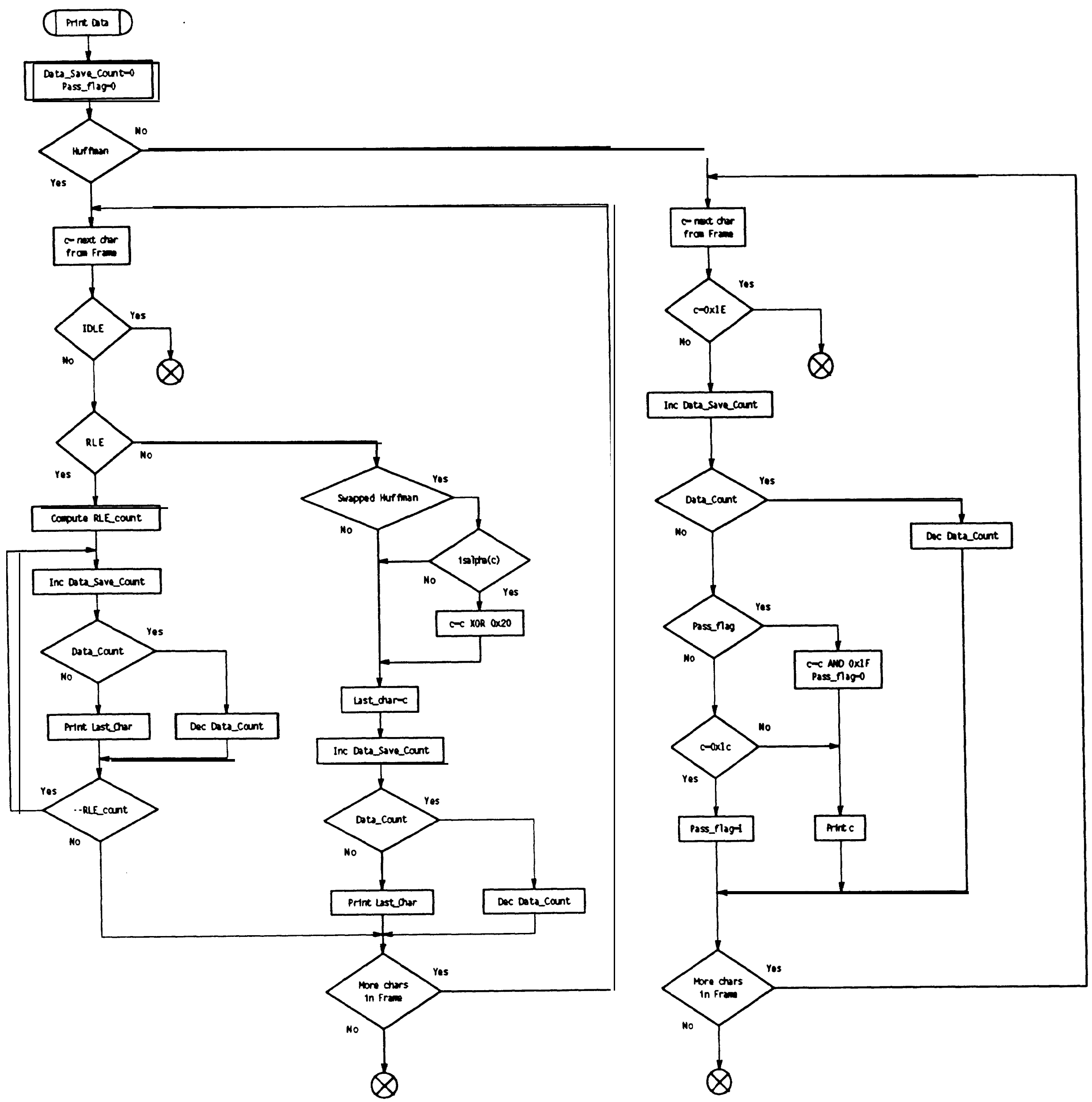
APPENOIX VII
Getbuf subroutine
Sheet 4 of 4



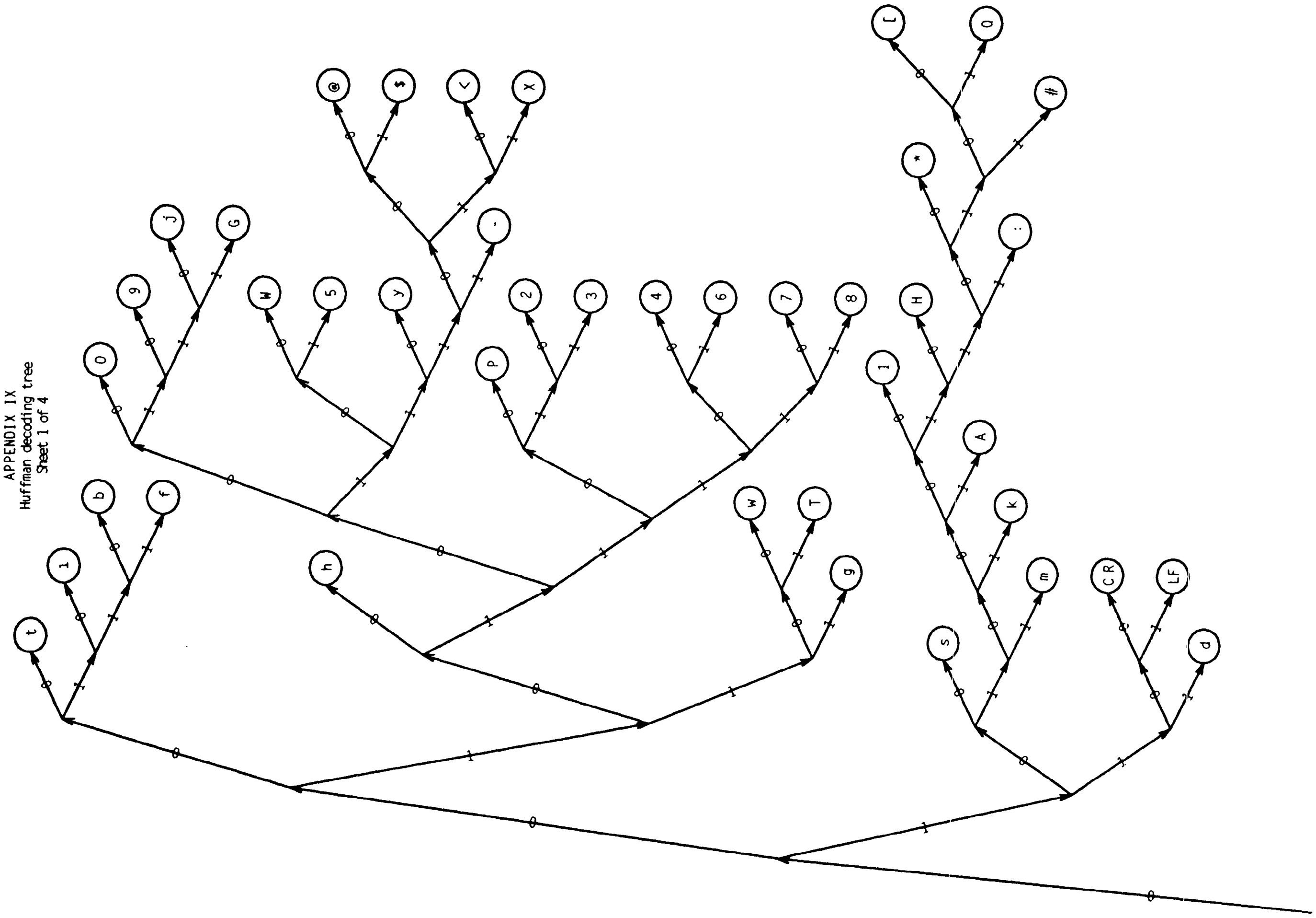




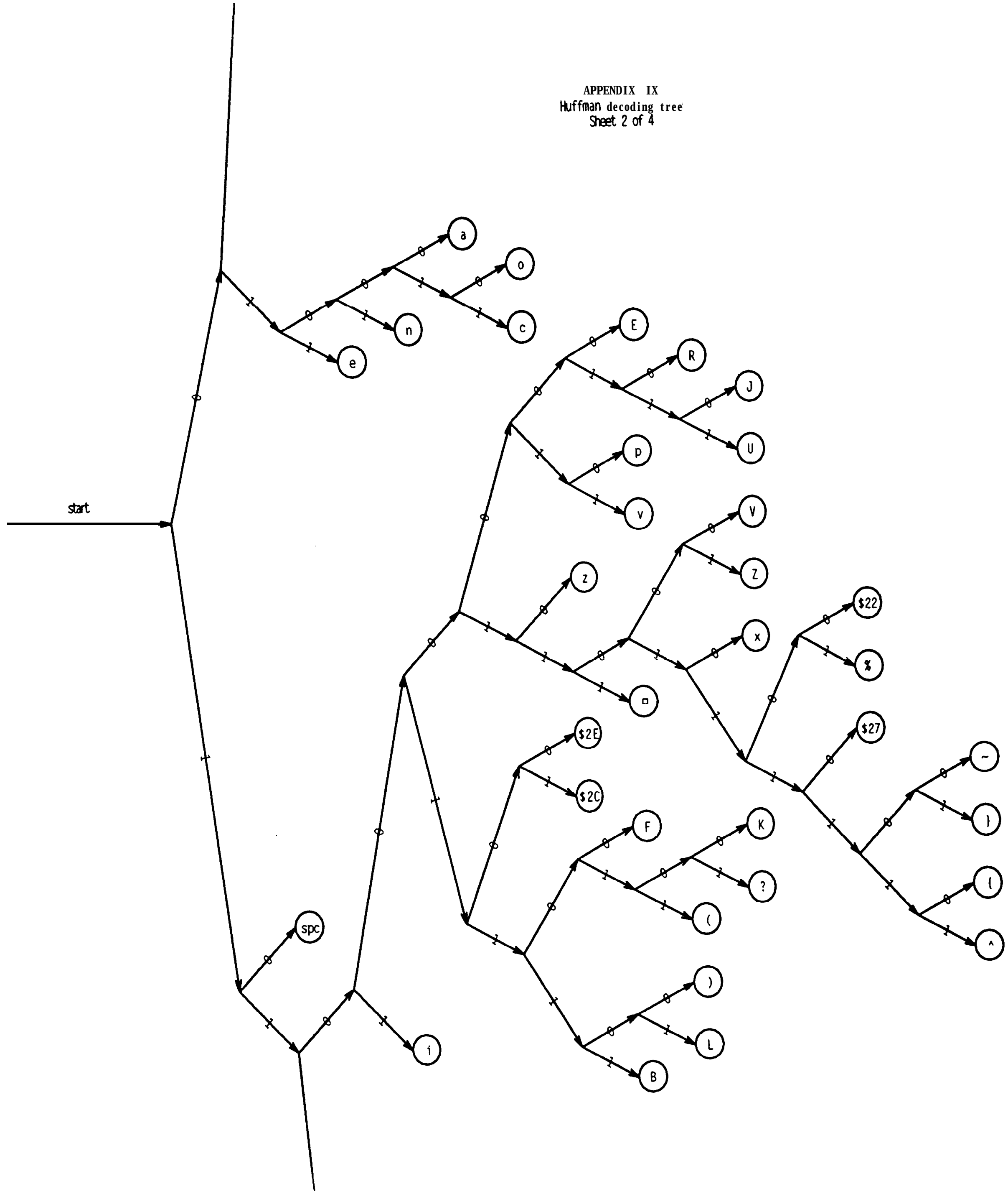
APPENDIX VIII
Print Data
Sheet 1 of 1



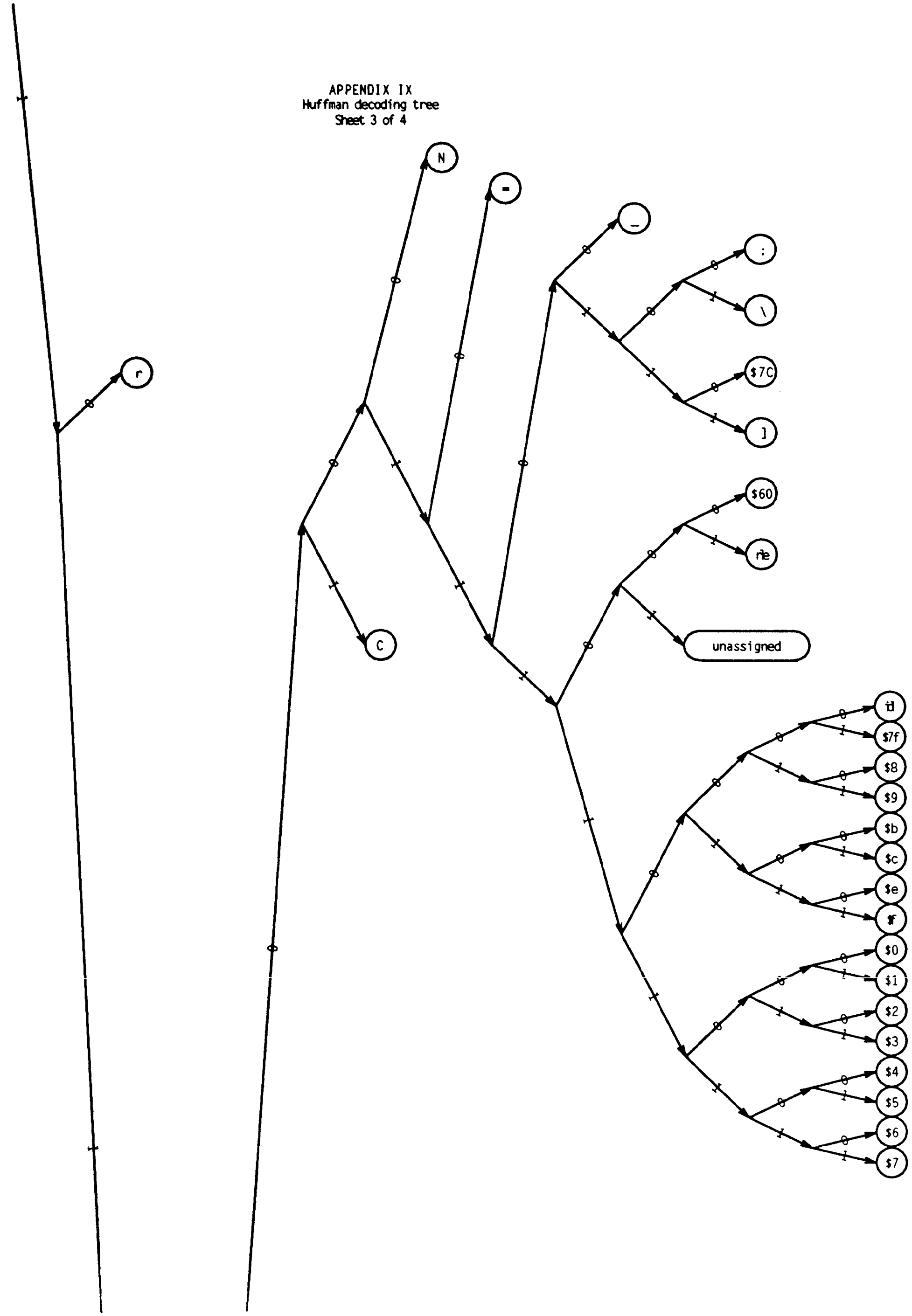
APPENDIX IX
Huffman decoding tree
Sheet 1 of 4



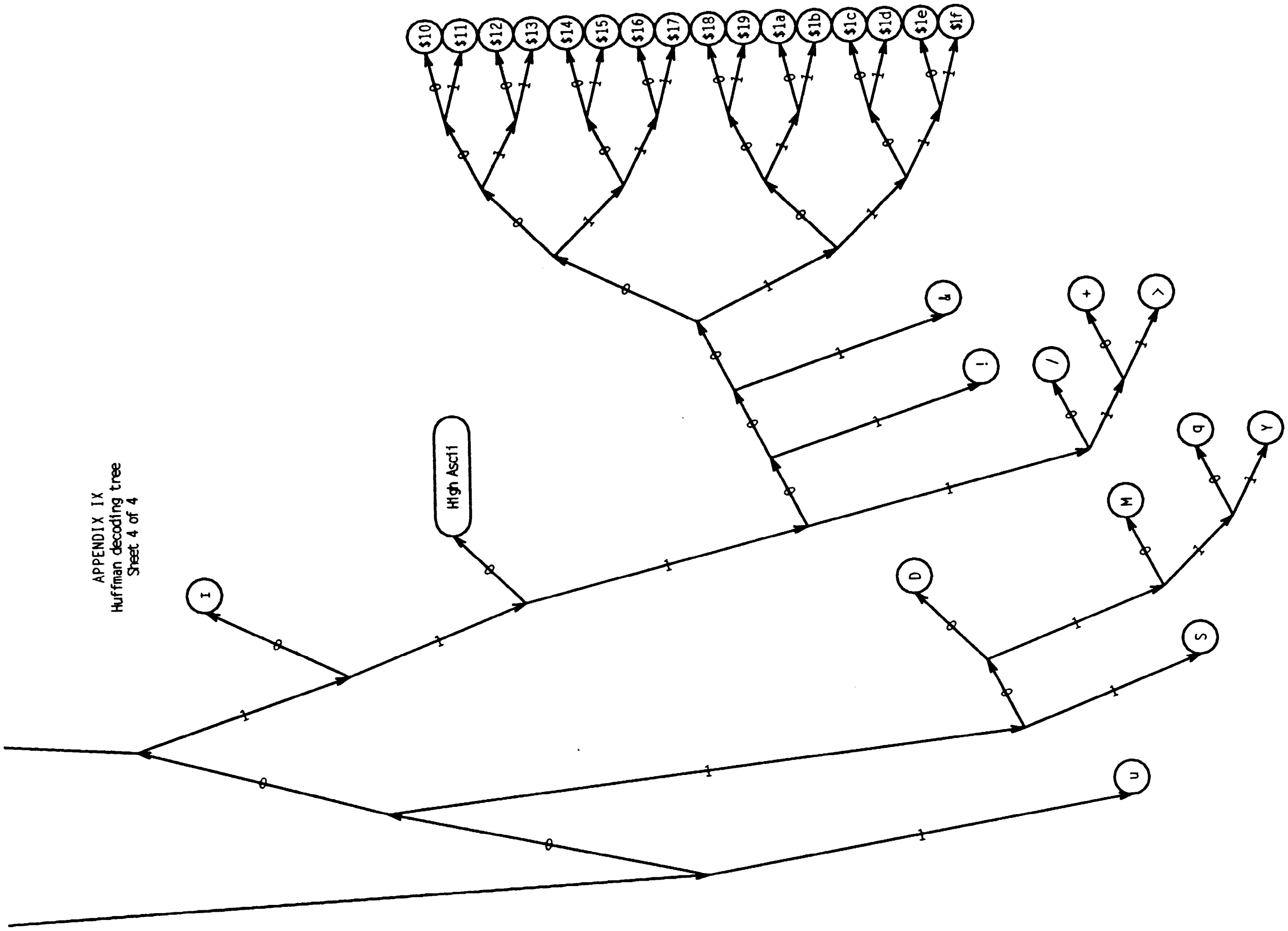
APPENDIX IX
Huffman decoding tree
Sheet 2 of 4



APPENDIX IX
Huffman decoding tree
Sheet 3 of 4



APPENDIX IX
Huffman decoding tree
Sheet 4 of 4



Appendix 10

C Program for Golay Encoding and Decoding

```
#include "stdlib. h"
#include "stdio. h"
#include "string. h"
#include "ctype. h"
unsigned g[4096],wt[4096];
unsigned b[12]=
   {0xDC5,0xB8B,0x717,0xE2D,0xC5B,0x8B7,
    0x16F,0x2DD,0x5B9,0xB71,0x6E3,0xFFE};
void create_golay_table(void)
{
unsigned i,j,data;
for(i=0;i<4096;i++)
   {
   for(j=0,data=0;j<12;j++)
      {
      if(i&(0x800>>j))data^=b[j];
      }
   g[i]=data;
   }
}
void   create-weight-table(void)
{
unsigned i,j,data;
for(i=0;i<4096;i++)
   {
   for(j=0x800,data=0;j;j>>=1)
      {
      if(i&j)data++;
      }
   wt[i]=data;
   }
}
main(argc,argv)
int   argc;
char *argv[];
{
unsigned input,parity,i;
if(argc<2 I I argc>3 I I isalnum(argv[1][0])==0)
   {
   printf("g xxx       displays golay coding of "
                       "hex value xxx\n");
   printf("g xxx yyy   displays results of error "
                       "correction of xxx data "
                       "and yyy parity\n");
   return(O);
   }
if(sscanf(argv[1],"%x",&input)!=1)
   {
   printf("invalid data input\n");
   exit(l);
   }
if(input>0xFFF)
   {
   printf("input too large\n");
   exit(2);
   }
create_golay_table();
create-weight-tableo;
if(argc==2)printf("%3.3X ==> "
                  "%3.3X\n",input,g[input]);
else
   {
   if(sscanf(argv[2],"%x",&parity)!=1)
      {
      printf("invalid parity input\n");
      exit(3);
      }
   if(parity>0xFFF)
      {
      printf("parity too large\n");
      exit(4);
      }
   if(wt[input^g[parity]]<=3)
      {
      printf("%3.3X and %3.3X ==> "
             "%3.3X\n",input,parity,g[parity]);
      return(O);
      }
   for(i=0;i<12;i++)
      {
      if(wt[input^g[parity]^b[i]]<=2)
         {
         printf("%3.3X and %3.3X ==> "
                "%3.3X\n",
                input,parity,g[parity]^b[i]);
         return(O);
         }
      }
   if(wt[g[input]^parity]<=3)
      {
      printf("%3.3X and %3.3X ==> "
             "%3.3X\n",input,parity,input);
      return(0);
      }
for(i=0;i<12;i++)
      {
      if(wt[g[input]^parity^b[i]]<=2)
         {
         printf("%3.3X and %3.3X ==> "
                "%3.3X\n",
                input,parity,input^(0x800>>i));
         return(O);
         }
      }
   printf("cannot correct\n");
   }
return(O);
}
```

Appendix 11

Huff man Table

by ASCII Code

0x00	1111000011111000	
0x01	1111000011111001	
0x02	1111000011111010	
0x03	1111000011111011	
0x04	1111000011111100	
0x05	1111000011111101	
0x06	1111000011111110	
0x07	1111000011111111	
0x08	1111000011110010	
0x09	1111000011110011	
0x0A	001101	
0x0B	1111000011110100	
0x0C	1111000011110101	
0x0D	001100	
0x0E	1111000011110110	
0x0F	1111000011110111	
0x10	1111001110000000	
0x11	1111001110000001	
0x12	1111001110000010	
0x13	1111001110000011	
0x14	1111001110000100	
0x15	1111001110000101	
0x16	1111001110000110	
0x17	1111001110000111	
0x18	1111001110001000	
0x19	1111001110001001	
0x1A	1111001110001010	
0x1B	1111001110001011	
0x1C	1111001110001100	
0x1D	1111001110001101	
0x1E	1111001110001110	
0x1F	1111001110001111	
0x20	10	;‘ ’
0x21	11110011101	;!
0x22	110001101100	;"
0x23	0010100011011	;#
0x24	0001010111001	;$
0x25	110001101101	;%
0x26	111100111001	;&
0x27	110001101110	;'
0x28	110011011	;(
0x29	110011100	;)
0x2A	001010001100	;*
0x2B	111100111110	;+
0x2C	1100101	;,
0x2D	00010101111	;-
0x2E	1100100	;.
0x2F	11110011110	;/
0x30	11000111	;0
0x31	001010000	;1
0x32	0001011010	;2
0x33	0001011011	;3
0x34	0001011100	;4
0x35	0001010101	;5
0x36	0001011101	;6
0x37	0001011110	;7
0x38	0001011111	;8
0x39	0001010010	;9
0x3A	00101000111	;:
0x3B	11110000110100	;;
0x3C	0001010111010	;<
0x3D	1111000010	;=
0x3E	111100111111	;>
0x3F	1100110101	[illegible]
0x40	0001010111000	;@
0x41	00101001	;A
0x42	11001111	;B
0x43	11110001	;C
0x44	11110100	;D
0x45	11000000	;E
0x46	11001100	;F
0x47	00010100111	;G
0x48	0010100010	;H
0x49	11110010	;I
0x4A	1100000110	;J
0x4B	1100110100	;K
0x4C	110011101	;L
0x4D	111101010	;M
0x4E	111100000	;N
0x4F	000101000	;O
0x50	000101100	;P
0x51	00101000110101	;Q
0x52	110000010	;R
0x53	1111011	;S
0x54	0001101	;T
0x55	1100000111	;U
0x56	1100011000	;V
0x57	0001010100	;W
0x58	0001010111011	;X
0x59	1111010111	;Y
0x5A	1100011001	;Z
0x5B	00101000110100	;[
0x5C	11110000110101	;\
0x5D	11110000110111	;]
0x5E	11000110111111	;^
0x5F	111100001100	;_
0x60	11110000111000	;`
0x61	01000	;a
0x62	0000110	;b
0x63	010011	;c
0x64	00111	;d
0x65	011	;e
0x66	0000111	;f
0x67	000111	;g
0x68	000100	;h
0x69	1101	;i
0x6A	00010100110	;j
0x6B	0010101	;k
0x6C	000010	;l
0x6D	001011	;m
0x6E	0101	;n
0x6F	010010	;o
0x70	11000010	;p
0x71	1111010110	;q
0x72	1110	;r
0x73	00100	;s
0x74	00000	;t
0x75	11111	;u
0x76	11000011	;v
0x77	0001100	;w
0x78	1100011010	;x
0x79	0001010110	;y
0x7A	1100010	;z
0x7B	11000110111110	;{
0x7C	11110000110110	;|
0x7D	11000110111101	;}
0x7E	11000110111100	;~
0x7F	1111000011110001	
111100110xxxxxxx		;upper ascii
0x80	1111001100000000	
0X81	1111001100000001	
0X82	1111001100000010	
etc.		
IDLE	1111000011110000	
RLE	11110000111001	
UNUSED	1111000011101	

Huffman Table

by Huffman Code

0x20	10	;' '
0x65	011	;e
0x69	1101	;i
0x6E	0101	;n
0x72	1110	;r
0x61	01000	;a
0x64	00111	;d
0x73	00100	;s
0x74	00000	;t
0x75	11111	;u
0x0A	001101	;LF
0x0D	001100	;CR
0x63	010011	;c
0x67	000111	;g
0x68	000100	;h
0x6C	000010	;l
0x6D	001011	;m
0x6F	010010	;o
0x2C	1100101	;,
0x2E	1100100	;.
0x53	1111011	;S
0x54	0001101	;T
0x62	0000110	;b
0x66	0000111	;f
0x6B	0010101	;k
0x77	0001100	;w
0x7A	1100010	;z
0x30	11000111	;0
0x41	00101001	;A
0x42	11001111	;B
0x43	11110001	;C
0x44	11110100	;D
0x45	11000000	;E
0x46	11001100	;F
0x49	11110010	;I
0x70	11000010	;p
0x76	11000011	;v
0x28	110011011	;(
0x29	110011100	;)
0x31	001010000	;1
0x4C	110011101	;L
0x4D	111101010	;M
0x4E	111100000	;N
0x4F	000101000	;O
0x50	000101100	;P

0x52	110000010	;R
0x32	0001011010	;2
0x33	0001011011	;3
0x34	0001011100	;4
0x35	0001010101	;5
0x36	0001011101	;6
0x37	0001011110	;7
0x38	0001011111	;8
0x39	0001010010	;9
0x3D	1111000010	;=
0x3F	1100110101	;?
0x48	0010100010	;H
0x4A	1100000110	;J
0x4B	1100110100	;K
0x55	1100000111	;U
0x56	1100011000	;V
0x57	0001010100	;W
0x59	1111010111	;Y
0x5A	1100011001	;Z
0x71	1111010110	;q
0x78	1100011010	;x
0x79	0001010110	;y
0x21	11110011101	;!
0x2D	00010101111	;-
0x2F	11110011110	;/
0x3A	00101000111	;:
0x47	00010100111	;G
0x6A	00010100110	;j
0x22	110001101100	;"
0x25	110001101101	;%
0x26	111100111001	;&
0x27	110001101110	;'
0x2A	001010001100	;*
0x2B	111100111110	;+
0x3E	111100111111	;>
0x5F	111100001100	;_
0x23	0010100011011	;#
0x24	0001010111001	;$
0x3C	0001010111010	;<
0x40	0001010111000	;@
0x58	0001010111011	;X
UNUSED	1111000011101	
0x3B	11110000110100	;;
0x51	00101000110101	;Q
0x5B	00101000110100	;[
0x5C	11110000110101	;\
0x5D	11110000110111	;]
0x5E	11000110111111	;^

0x60	11110000111000	;`
0x7B	11000110111110	;{
0x7C	11110000110110	;|
0x7D	11000110111101	;}
0x7E	11000110111100	;~
RLE	11110000111001	
0x00	1111000011111000	
0x01	1111000011111001	
0x02	1111000011111010	
0x03	1111000011111011	
0x04	1111000011111100	
0x05	1111000011111101	
0x06	1111000011111110	
0x07	1111000011111111	
0x08	1111000011110010	
0x09	1111000011110011	
0x0B	1111000011110100	
0x0C	1111000011110101	
0x0E	1111000011110110	
0x0F	1111000011110111	
0x10	1111001110000000	
0x11	1111001110000001	
0x12	1111001110000010	
0x13	1111001110000011	
0x14	1111001110000100	
0x15	1111001110000101	
0x16	1111001110000110	
0x17	1111001110000111	
0x18	1111001110001000	
0x19	1111001110001001	
0x1A	1111001110001010	
0x1B	1111001110001011	
0x1C	1111001110001100	
0x1D	1111001110001101	
0x1E	1111001110001110	
0x1F	1111001110001111	
0x7F	1111000011110001	
11100110xxxxxxx	;upperascii	
0x80	1111001100000000	
0X81	1111001100000001	
0X82	1111001100000010	
etc.		
IDLE	1111000011110000	

For the Huffman decoding tree, see Appendix 9.