NCAR-TN/IA-93
NCAR TECHNICAL NOTE

January 1974

Techniques for the Processing, Storage, and Exchange of Data

Roy L. Jenne
Dennis H. Joseph

ATMOSPHERIC TECHNOLOGY DIVISION

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
BOULDER, COLORADO

PREFACE

The National Center for Atmospheric Research receives computer data in many different binary and character formats. We send out data in both binary and character form. In this report we discuss selected aspects of the computer hardware systems and some of the considerations involved in choosing data formats. We also describe some of the procedures and computer subroutines that make it relatively easy to process the data regardless of the format. These procedures lay the groundwork for making the exchange of data between computer systems as simple as possible.

ACKNOWLEDGMENTS

We want to thank Dori Bundy for her constructive comments concerning this text, and Linnea Cookson for typing it.

October 1973

CONTENTS

1. INTRODUCTION

We will discuss some of the considerations involved in deciding how to process and format data and how to communicate it from one place to another. Our goal is to make data exchange as simple as possible. Several programs that help in data processing will be discussed.

In storing and accessing data one should try to achieve some optimum combination of the following desirable features:

a. Minimize the time necessary for the home computer to obtain the data and decode it so that it is available for calculations.

b. Minimize the time necessary for another computer to decode the data.

c. Minimize the storage volume necessary to hold the data.

d. Maintain checks to insure that if any data has been altered since it was originally created, the programmer will be aware of the change.

e. Guard against irretrievable loss of portions of data sets.

f. Minimize programmer time necessary to access the data on the home computer.

g. Minimize programmer time necessary to access the data on another computer.

It has become rather general to think of information as a string of characters where numbers are almost always thought of as digits in the base 10 number system. It has been too generally accepted that while binary information might be output from one's own computer and read back in, the only practical way to exchange information between computers is to convert all binary numbers to base 10 digits, output these, and then read them into a second computer for conversion back to binary form. Such conversions take a lot of computer time and the character data require more storage volume than the alternative binary packing.

Thus there often are large advantages in storing numerical data in packed binary form. However, when one sends out packed binary information, many users on other computer systems spend time writing special purpose machine language subroutines to unpack the data. We will describe a few general purpose subroutines which make it as easy or easier to unpack binary packed data than to decode data packed in character form. These subroutines are also useful for unpacking non-standard character codes.

In the following sections we will first discuss magnetic tapes and then the exchange of data in character codes. Then we will describe information in terms of a string of bits and consider time and volume advantages in storing data in packed binary form. We will then describe selected computer routines that are desirable for processing packed data, and for processing other data too. We will also present the case for keeping data under checksum control. Guidelines for exchanging data between computers will be given. This will be followed by a set of program listings of selected key routines.

2. MAGNETIC TAPE PRACTICES

2.1 Introduction

Magnetic tapes are commonly involved in data processing and in the exchange of information between computers. On a standard 1/2 inch wide, 2400 foot reel of magnetic tape, one can block about 230,000 card images of data at a recording density of 800 bytes per inch (blocked 40 cards per record). In this example, the tape can be stored in about .083 cubic feet whereas the cards would take 30.2 cubic feet. Thus there is volume reduction of about 350 to 1. There are also large reductions in processing time and in shipment cost. However, when the data volume is very low, cards may be more efficient.

We will discuss the practices most commonly used with standard 1/2 inch wide magnetic tape. The bulk cost of 2400 foot reels of this tape has decreased from about $40 to $9 per tape over the last ten years. Other tape systems such as one-inch-wide "hyper" tapes are more costly and have not achieved wide usage.

The tape standard for several years was to record seven channels of data (6 for information and 1 for parity) on the half inch tape in a recording density of 200, 556, or 800 BPI (number of 6-bit bytes of data per inch). Now this standard is paralleled by one in which nine channels of data (8 information and 1 parity) are written on the same 1/2 inch wide physical tape. The common recording densities in this new standard are 800, and 1600 BPI (8-bit bytes of data per inch). One type of tape unit is needed for the 7-track recording and another for 9-track recording. Within each category, the less expensive drives are usually slower, and often they cannot read or record at the higher recording densities. Some small drives are also specialized so that they will read even or odd parity, but not both.

Since a lot of data has been written in each of the two conventions (7 and 9 track), it is often frustrating not to have dual capability. Currently NCAR only has 7 channel drives, and thus must have 9 channel tapes recopied to 7 channel on another computer. We also buy 7-track tape drives for our small computers and for our scientific recording hardware to match our main computers--even though 9 channel tapes may later become the primary standard.

We will first discuss the 7-track tapes in more detail and then consider 9-track tapes.

2.2 7-Track Tapes

Early computers were character oriented, and for a time had input/output only on cards. Thus it is likely that the engineers designing the first magnetic tape systems thought solely in terms of characters and not in terms of general binary information. We will see that their choice of even parity recording on tape does not permit one to record general binary data. Thus the odd parity convention had to be added to record that kind of data. BCD even parity may have been an unlucky happenstance or the choice may have been made so that at least 2 bits would be 1's in any 7-bit byte on the tape.

The 7-track tape standard has been based on a 6-bit character code both in computer memory and on magnetic tape. The six bits describing the character code are written horizontally across the tape and the seventh channel has a parity bit. Such tapes are normally referred to as even parity, BCD mode tapes because the choice of a 1 or a 0 for the parity bit is made so that the sum of the seven bits which describe the character and its parity will be an even number.

On the 7-track tapes a zero bit in a channel is indicated by blank erased tape (no magnetic flux changes) and a one bit is represented by a magnetic field change. This is called a non return to zero (NRZI) recording ~~system~~. A character made up of 6 zero bits would have a parity of zero when recorded in even parity. Thus the whole character slot on the tape would show up as blank tape; such a character code is not permitted because the character would be lost on the read and probably would also cause other read problems because of the long distance between valid data frames. The character is lost because at least a single one bit must be sensed to determine that there is a frame of information.

The permitted 6-bit character codes for BCD even parity tape range from 01 octal to 77 octal or 1 to 63 decimal. Since 00 octal is not a permitted code, general binary information cannot be recorded in this mode.

In order to handle general binary information, the odd parity, binary write convention has to be used. Odd parity could have been used to record the BCD characters as well as the general binary data, but for compatibility the old even parity convention has been maintained as well. Thus most tape drives have to include logic to read and write in either even or odd parity mode.

We will now discuss the internal BCD to external BCD character conversions. With 7-channel tapes and 6-bit character codes on tape, the code for the character in computer memory is usually a 6-bit code, though often a different code than the one on tape. In most systems, when the programmer specifies an even parity read from a tape unit, the hardware gives him an automatic character conversion from external BCD (tape code) to the internal code. A better system design allows the programmer to suppress the conversion if desired. When the internal code set is a full set of 63 codes there is a one-to-one conversion and the programmer then at least is able to convert back to the external code when that is desirable.

With 6-bit character codes, the length of computer words is normally chosen to be 12, 18, 24, 36, 48, or 60 bits. Character oriented machines then have character (or word) sizes of 6 bits each.

2.3 9-Track Tape

In the middle 1960's, IBM decided to change the character standards so that people could work with more than 63 different characters. Thus, IBM has gone to an 8-bit character in the 360 and 370 series computers. They use an associated 9 channel tape and computer words that are usually 16, 32, or 64 bits long.

The specifications for the physical tape were not changed for the 9-channel tape drives: it is still the 1/2 inch wide tape.

The introduction of these tape drives provided the opportunity to use only odd parity. There is no character conversion when 8-bit characters are written from memory onto these tapes.

At 800 BPI a zero bit on the tape looks the same as blank tape; this is the same non-return to zero (NRZI) recording system that is used on the 7-track tapes. However, at 1600 BPI, a phase encoding method is used whereby a zero also gives a magnetic flux change. Reports indicate that phase encoding at 1600 BPI is as reliable as NRZI recording at 800 BPI.

On the 9 channel drives, IBM and the other manufacturers have cut the record gap length from .75 to .60 inch.

One of the better drives will read or write on 9-track tape at 800 BPI with NRZI recording, or read or write at 1600 BPI with phase encoding.

Most IBM 360 installations use EBCDIC (Extended Binary-Coded Decimal Interchange Code) for the internal character code. However, a facility can choose ASCII (American Standard Code for Information Exchange). Since a tape read just moves a string of bits from tape to memory, an internal character conversion is necessary if a tape of character data is read from a machine using a different internal code.

2.4 Tape copy options in going from 9- to 7-track tape

When copying a 9-track tape to a 7-track tape one may take one of the options:

a. Assume that all 8-bit characters can be expressed in a 6-bit code and output a standard external BCD code in even parity.

b. Delete the two high order bits of each 8-bit byte and write the other 6 bits on the tape in odd parity.

c. Retain the 8-bit bytes of character or binary data in one string of bits on the 7 channel tape. In 360 systems this is the "convert on, translate off" option. In this case the 7 channel tape will be in odd parity and the original 8-bit characters will be read into successive 8-bit units of computer memory from the 7 channel tape. This option is also used to read a 7 channel binary tape as a string of bits into a 360 machine.

We generally choose option c for this problem because the information is compact and it hasn't been altered as in options a or b.

2.5 Record gaps and end file marks

A section of blank tape called a record gap is used to separate physical records on a tape. The gap is usually .75 inches long although it is now as short as .60 inches on some units. The gap must be long enough to allow the tape motion to stop, and then to be restarted before the next record. If the data records are short, such as one card image per record, most of the tape is used for record gaps.

An end file mark is a special one character record on the tape that says: "There is no more data of this category on the tape." Usually only one file of data is written on a tape; in some cases multiple files are kept on a tape, but this is some nuisance for the programs that copy, inventory, and access the data. If there are several files on a tape, the last file should be followed by at least two end file marks to signify the end of all data on the tape.

Some software systems use synthetic end file marks. These are usually a special short record that conveys the same message as the standard end file (one character) record.

There also are various schemes to block a number of shorter logical records together into larger physical records on a tape. Different procedures are also used to segment long logical records into shorter physical records.

3. INFORMATION IS A STRING OF CHARACTERS

3.1 Introduction

In the section on magnetic tapes we spent a good deal of time discussing characters of data, where a given character might be 3, 1, 9, +,

=, A, X, etc. We did this because some aspects of computer and tape system design are associated with the character codes that are used.

Each character is represented by a selection of holes in punched cards or paper tape, or is represented by a code in 4, 6, or 8 bits on magnetic tape or in computer memory.

We will now show why character concepts are necessary, and how characters can be used in data exchange. We will also describe some of the limitations in using characters for data input/output.

3.2 The need for characters

Program information is most easily entered into computers by using symbolic letters and base-10 numbers. Computers must also be able to process and print letters giving textual information along with numbers which are usually desired in the normal base-10 number system. In fact, some earlier computers (or somewhat more recent ones that are cheaper and business oriented) were built to process characters and have limited, if any, capability for handling binary information.

There are computer applications (such as for library bibliography data) where most of the information is textual data and must be in character coded form.

3.3 Exchanging character data between computers

We have noted that with 6-bit character codes, the internal code differs from one computer system to the next. Within a computer, code conversions may be necessary because, for example, the printer may require a different code from that used by the system. The programmer is often not aware of such code changes because the operating system makes them when necessary. When data is written onto tape in even parity mode, the code is changed to external BCD.

The external BCD code is common to 7-track tape systems; ASCII or EBCDIC is used on 9-track tape systems. Provision has been made to change between 6- and 8-bit codes to the extent possible. Thus, for example, a 9-track tape with characters for base-10 numbers may be copied to the standard 6-bit external BCD code on 7-track tape.

3.4 Limitations to the exchange of data in character form

The exchange of character data is sometimes simple because of the common external codes which are converted to the necessary internal code and which then can be handled by the formatting routines on a user's computer system. We will now discuss some of the complications that can arise in this apparently simple process.

Much character data is taped directly from punched cards where over-punches are frequently used to save space. An overpunch on a card column means that there is an 11 punch or a 12 punch along with the punch for the numeric digit in rows 0 through 9. Overpunches on a card column might be used to say that the sign of the quantity is negative, or they might change the range of the data. (For example, a punched 27 with an overpunch on one of the two columns might mean 127.) With an overpunch, a character is no longer a standard character for a base 10 number and thus cannot be processed with standard Fortran I or F formats.

In other character data sets, the volume waste in packing numeric characters into 6-bit digits has been recognized, and two of the 6 bits have been used for additional information, thus creating non-standard numeric characters.

Data from character machines has been received in which the sign is carried either as the rightmost character in a field, or as high order bits on the righthand numeric character. This is also non-standard for most formatting packages.

Character data is sometimes included with binary data, so that one receives the data in the internal code of the originating computer system. Internal codes are also written on tape when 4- and/or 8-bit characters are used on a 9-track system, but here the problem is less severe (than for the 6-bit characters) because there are only two different sets of internal codes.

Although the situation is getting better, many systems have not had the ability to read in a reasonable sized buffer of character data, and to reformat it from the buffer. Either they have lacked the Buffer-read capability or the Decode statements. In some computer systems the number of characters allowed in a BCD-read from a tape is limited by the card reading and printer formatting logic; on these systems, the number of characters in a record is generally limited to 128, 132, or 150 characters. This is too short a record to be practical for the exchange of very much data. We have even encountered modern machine systems that would not accept a BCD record of reasonable length; for example one such system would not accept a record more than 2000 characters long.

It is more difficult and costly in computer time to keep data under checksum control when it is put on tape in even parity character codes than when it is stored in odd parity.

If anything is non-standard about numerical character data, much more programmer time is needed to unpack it than to unpack binary data. Using methods that will be described later, the unpacking of binary data is as easy or easier than the unpacking of even the standard numerical characters.

In the next section we will discuss the computer time and storage volume advantages of packing numerical data in binary rather than in base 10 characters.

4. INFORMATION IS A STRING OF BITS

4.1 Introduction

It is often a simplifying concept to think of information as a string of bits. Since any character or binary number is represented by binary bits in digital computers, we can think of all such information as just a bit string whether it is in core memory, on a tape, or on some other storage device. In different cases the bit string might contain information as follows:

a. A succession of 6-bit characters where the characters might be represented by any of a number of codes.

b. A succession of 4- or 8-bit characters.

c. A succession of floating point binary words from a particular machine where each word is 36 bits long.

d. A series of fixed point binary words from another machine where each number is in 32 bits.

e. A series of satellite brightness data points, where each measurement is packed in 5 bits (each has a gray scale of 0-31).

f. A series of bits giving data for a balloon atmospheric sounding flight where the first 6 bits give status, the next 11 bits give pressure, then 16 for the height, 11 for temperature, 11 for relative humidity, 9 for wind direction, 8 for wind speed and then a repeat of these for the next level.

Any of the above strings of bits can be relatively easily processed using methods to be partially discussed in this writeup (see especially Section 5.2). If information is maintained as a string of bits, one doesn't really care how many channels a recording tape has or what the characteristics of another storage medium such as drum or disk may be. That is, one doesn't care as long as the information can be delivered back as the same string of bits.

We will describe some of the volume and time savings that may be made by storing data in packed binary rather than in base-10 characters.

4.2 Time and volume: binary pack versus characters

We have noted that when numbers are being stored primarily for calculations rather than for printing, one can save both time and volume by packing the information in binary. As an example, suppose that we must store many numbers that range from 0 to 4000 in magnitude. The volume comparison is as follows:

Storage Method	Volume Per Number
binary pack	12 bits
4 bit characters	16 bits
6 bit characters	24 bits
8 bit characters	32 bits

The timing comparison on a Control Data Corporation (CDC) 6600 computer to unpack the preceding data:

	Type	Time
a.	time to unpack a 12 bit number using specialized machine language	3 μs
b.	time to unpack a 12 bit number using a subroutine called from Fortran	20 μs
c.	time to decode a number using a Fortran I4 format	120 μs

Thus one saves appreciable time and space with binary packing of data. Note that when processing character data, one can improve on the timing of the Fortran formatting routine by alternative programming. In fact, it is often necessary to program the formatting in order to maintain program control when illegal character codes are found in the data set as is often the case.

In the preceding timing comparisions the 3 μs time was achieved by taking advantage of the fact that five 12-bit bytes can be evenly packed into a 60-bit word. It is interesting to note that a computer word with 60 bits can be evenly divided into bytes of lengths 2, 3, 4, 5, 6, 10, 12, 15, 20, or 30 bits each, whereas a 64-bit word can only be divided by bytes of 2, 4, 8, 16, or 32 bits.

Additional timing comparisons are made in our discussion of the Gbytes subroutine in 5.2.

When packing a binary number into a byte, we generally use neither the sign-magnitude convention nor one of the complement conventions for negative numbers. We add an appropriate bias so that all numbers are positive. This saves time in the packing and unpacking of information and is easier to program.

If packed data is received which consists of positive and negative numbers from a ones or twos complement machine or in sign magnitude convention (sign first), it is still fairly easy to process. One uses GBYTES (to be discussed) to right adjust a packed data sample into a computer word so that it looks like a positive number. A test is made to see if the number is too big to be positive; if it is negative, some arithmetic is carried out as in the following example to obtain the desired negative number. Consider binary data packed into 4 bits which uses one of the packing conventions: sign-magnitude, ones complement, or twos complement.

Let k = 2^4 corresponding to data packed into 4 bits. The data and needed arithmetic are given by:

decimal number	n = packed sign-magnitude	needed arithmetic	n = packed ones complement	needed arithmetic	n = packed twos complement	needed arithmetic
+2	0010	none	0010	none	0010	none
+1	0001	none	0001	none	0001	none
0	0000	none	0000	none	0000	none
-1	1001	k/2 - n	1110	n - k + 1	1111	n - k
-2	1010	k/2 - n	1101	n - k + 1	1110	n - k

4.3 Volume comparison of methods for storing rawinsonde data

Reductions in storage volume and in access time were made by reformatting mandatory level rawinsonde data into a binary format. This data was received in a BCD tape format that was much easier to process than most of the original punch card data. The format used 6-bit BCD characters; and since it was a fixed format, blanks had been inserted for missing data. The NCAR binary output format specifies the pressure level so that when all data at a level is missing, the level may be omitted. On an average, about a third of the levels in the fixed format had missing data; thus some volume reduction can be attrituted to adding the pressure data but omitting the missing levels. The NCAR binary format also includes more identification information, and six bits are used at each level to give the status of the data. Since the pressure is included in the format, data may also be recorded for significant levels.

One large batch of this raob data was received at NCAR on 56 tapes and reformatted to 16 output tapes. The input tapes used 109,700 feet of tape at 556 BPI and the reformatted output tapes used 22,400 feet at 800 BPI. If the input had been written at 800 BPI, the footage would have been 87,400 feet input compared to 22,400 output or a ratio of 3.9 to 1.0.

If four of the input records had been blocked together to make a 328-word (10 characters per word) record and this written at 800 BPI, the volume ratio would have been 2.64 to 1.0. Note that a 328-word record is about the same size as the packed and blocked reformatted records.

If the data had been reformatted into one element per 60-bit word instead of into packed binary, the volume would have been about six times greater than that achieved.

4.4 Timing comparison for unpacking rawinsonde data

We have noted the volume compaction of 3.9 to 1.0 achieved by reformatting and blocking the rawinsonde (raob) data. Now we will make some timing comparisons.

The time to read, unpack and reformat each original raob on the CDC 6600 is about 45.2 ms for raobs having about 9 levels of data and 51.7 ms for raobs averaging 24 levels. The unpack program uses methods faster than Decode statements, but could be made somewhat faster by having fewer calls to subroutines. It has a few simple checks on the range of the data. The CDC 6600 central processor time to reformat all 602,000 raobs from the original 56 tapes was thus about 502 min. This includes the time (about 38 minutes) to pack the data into the new format.

The average time to read and unpack a raob from the new format (using the Gbytes subroutines) is 3.40 ms for raobs averaging 22.3 levels, 2.76 ms for raobs averaging 16.9 levels, and 1.65 ms for raobs averaging 8.7 levels. The total time necessary to unpack all 602,000 raobs from the 16 tapes is thus about 34 minutes of CPU time.

Since we have used most of these data more than once, there has already been an appreciable time saving: only 34 minutes compared to 464 minutes are needed to access all of the data. It now appears that an optimized version of Gbytes can be written which would reduce this time to about 24 minutes.

4.5 Communication

Communications systems for transmitting data from one computer to another should be able to accept and deliver strings of bits instead of demanding a special character code for the text of the message. Any type of character or packed binary information can be easily sent from one place to another in a string of bits. The data should be under checksum control while being sent and a retransmission should be requested if the checksum cannot be verified.

5. ROUTINES TO MANIPULATE DATA

5.1 Introduction

In this section we will discuss programs desirable to handle the following problems within a computing system.

a. Pack data into a bit string (Sbyte, Sbytes) and unpack it (Gbyte, Gbytes).

b. Input/output problems.

c. Rdtape/Wrtape and Iowait.

d. Tape copy.

e. Block and unblock short, variable length, records of data and maintain checksum control (Rptout, Rptin).

f. Unblock short, variable length records of data that were written by Rptout on a machine with a different word length (RptinX).

g. Sort reports stored in Rptout blocked form, or stored in unblocked form (Sortbb, Sortuu).

Note that the word length of both the CDC 6600 and CDC 7600 at NCAR is 60 bits.

5.2 Packing (or unpacking) of data into bit strings

In the previous section, we discussed the volume and timing reasons for packing information in binary. We will now discuss routines that give one the capability of formatting data in terms of numbers of bits rather than numbers of characters. NCAR routines that do this are Gbyte, Sbyte, Gbytes, and Sbytes. These routines pack or unpack data to (or from) a string of bits and ignore word boundaries in the string.

Once these routines (Gbytes, etc.) are written in machine language for any given machine, they can then be used for various data unblocking and decoding jobs. For example, where variable length reports have been blocked (using Rptout described in 5.6) on a machine with one word length, Gbytes can be used to unblock reports from each record that is read by a computer with another word length. In Section 5.7 we will describe RptinX which unblocks such records.

First we will explain the routines (Gbytes, etc.) by showing their logical similarity to format statements that process characters. In the examples, we will use the same count of characters to skip in the Decode format statement that are used for bits to skip in Gbytes, etc. Similarly, the number of characters (such as I5) to process in the format will be the same as the number of bits to move in Gbytes:

a. Call Gbyte (Npack,Isam,iskip,nbits)

Example: Call Gbyte (Npack,Isam,3,5)

This skips 3 bits in Npack and puts the next 5 bits right adjusted into Isam with the rest of the Isam cell set to zero. The form (but one counts bits and the other counts characters) is similar to:

```
          Decode (8,912, Npack) Isam

      912 Format (3X,I5)
```

b. Call Sbyte (Npack,Isam,iskip,nbits)

This precisely reverses the data flow in Gbyte; thus the right-adjusted data in the Isam cell is stored into Npack. When the data is stored into Npack, the bits around this byte of data are not changed.

c. Call Gbytes (Npack,Isam,iskip,nbits,nskip,iter)

Example: Call Gbytes (Npack,Isam,7,5,2,120)

This skips 7 bits in the first Npack cell, puts the next 5 bits right-adjusted into the first cell of Isam, skips 2 bits in Npack and puts the next 5 bits into the second cell of the Isam array. This pattern of get 5 bits and skip 2 is repeated until 120 Isam cells contain right-adjusted 5-bit bytes of data with lead zeroes. (The final skip of 2 bits is meaningless.) The form is similar to:

```
          Decode (847,915, Npack)  (Isam(I),I=1,120)

      915 Format (7X,120(I5,2X))
```

(However, this Decode statement is not valid at NCAR because our Decode is limited to no more than 150 characters. Thus, six Decode statements would actually have to be used.)

d. Call Sbytes (Npack,Isam,Iskip,nbits,nskip,iter)

This precisely reverses the direction of the data flow in Gbytes and does not change other bits in the Npack array.

In the above examples, nbits has a maximum size of 60 bits, the machine word length. Nskip may be greater than 60 if desired. Iskip must now be less than 60 bits at NCAR.

Note that the routines Gbytes, etc. consider the information to be in one string of bits in memory; they don't care if some bytes of data must be unpacked from two successive memory words. The examples show that when a given call to Sbyte or Sbytes is used to pack data, the same statement, only with the name changed to Gbyte or Gbytes, will unpack the information.

If one has a listing of the calls to Sbyte and Sbytes that packed data on a computer with one word length, the comparable calls for another machine can quickly be written by modifying the starting Npack address and the initial iskip as necessary.

These routines are written up in the NCAR Fortran computing manual, where more examples are presented. Subroutine listings are in Appendicies A, F, and G.

Timing Comparisons

We will show a couple of comparisons between Gbytes statements and Decode statements that do precisely the same thing:

a.
```
          Decode (100,917, Npack)  (Isam(I),I=1,20)
     917  Format (20R5)
```

This is duplicated by:

```
     Call Gbytes (Npack,Isam,0,30,0,20)
```

The average CDC 6600 time for each Isam cell filled is 87 μs for the Decode and 22 μs for Gbytes.

b.
```
          Decode (104,918, Npack)  (Isam(I),I=1,20)
     918  Format (4X,20(R3,2X))
```

This is duplicated by:

```
     Call Gbytes (Npack,Isam,24,18,12,20)
```

The average time for each Isam cell filled is 260 μs for the Decode and 22 μs for Gbytes.

Examples of uses for Sbytes/Gbytes

a. A 7-track copy of a 9-track tape with New Zealand upper air data was obtained. It had 4- and 8-bit characters of data. Gbytes was used to put each character into an individual cell so that the data could be easily processed.

b. A tape was received having binary floating point data from a machine with a word length of 48 bits. Gbytes was used to put each 48-bit word in an NCAR 60-bit word. Then a small machine language routine was written to convert from the 48-bit floating point word to the CDC 6600 floating point word.

c. A tape was received with data allowing one to compare standard rawinsonde winds with winds derived from precisison radar tracking of the same balloon. The data was in blocked card image BCD records. But each successive character on the cards was actually an octal character from a 36-bit, IBM 7094 floating point word. Gbytes was used to put each character in a cell where the digits 0-7 were converted to binary 0-7. Then Sbytes was used to make a string of bits containing successive octal digits in 3 bits each. Gbytes was then used in three separate calls to put the signs, exponents, and fractions of the 36-bit floating point words into three arrays. These arrays were then combined with a Fortran arithmetic statement to give an array with the equivalent CDC 6600 floating point words.

d. Navy sea surface temperature data is packed into 16 bits for each grid point. Gbytes makes it easy to unpack such data.

e. Suppose that many numbers that can be represented by the decimal digits ± XXXX.XX will be output on the 6600 for use on a 32-bit machine. One can multiply the numbers by 100, then add 1,000,000 and convert to fixed point numbers that will have a range of 1 to 1,999,999. These numbers are contained within 21 bits. They could be packed (by Sbytes) into successive 21-bit bytes and unpacked by a Gbytes program written for the other machine. However, it is not too wasteful in volume to pack them into successive 32-bit bytes so that they will be available as fixed point numbers in the 32-bit machine without unpacking.

In one case, the numbers were of this range, but it was also desirable to have them in floating point form for the 32-bit machine. Since the fixed point numbers didn't extend into the range of the exponent, a common exponent could be used which is appropriate when the binary point is on the right hand side of the word. This exponent was put into each word prior to packing by Sbytes.

f. A blocked tape was received in which the length of each logical record was 72 characters. Since there are ten characters in our 60-bit word, one can't unblock such records by just moving whole words. We wrote a routine (UZBLOK) that uses Gbytes to unblock such a tape when the length of a logical record is any fixed number of bits.

g. When short variable length logical records have been blocked into longer records on a machine with a different word length, Gbytes can be used to unblock the records. In Section 5.7 we will discuss the routine RPTINX that is used for this purpose when the data has been blocked by a routine called RPTOUT which will be discussed.

h. Sbytes and Gbytes is used in the packing and unpacking of many types of data such as rawinsonde data.

5.3 Input/Output problems

Every system needs a general purpose I/O routine like the Fortran Buffer In, Buffer Out statements that allow one to read (or write) reasonable length records into (or from) a buffer in memory. With odd parity (binary) records, a bit string is delivered into memory that is the same as the bit string on tape (or other storage device) and the same as existed in the output computer. With even parity (BCD) tape reads, it is desirable to be able to read in a bit string the same as the tape code. The reason is that external BCD code is often more convenient to work with, and there are often at least some characters that are invalid, so that Decode statements cannot be used. Data with overpunches is more convenient to use in external BCD. For awhile NCAR didn't have a one-to-one conversion from external code to internal code, so certain overpunch data was lost if code conversion was used. However, many systems have an automatic character conversion that is impossible to bypass.

We feel that systems should be able to read BCD or binary records up to 1000 60-bit words in length, and longer if possible. Because of restrictions in other systems and considerations of buffer sizes and of good tape practice (records not too short or too long), we generally limit output data record lengths to be under 512 words. The maximum length of Rptout physical records is 354 words.

NCAR cannot write or read 9-track tapes as yet.

5.4 RDTAPE, WRTAPE, IØWAIT

The RDTAPE routines are used at NCAR for reading and writing records on tapes. In installations having BUFFER IN operations available, the RDTAPE simulator (Appendix B) may be used to run with programs requiring RDTAPE. At other installations it is necessary to write RDTAPE simulators or to replace the RDTAPE, WRTAPE, and IØWAIT calls with similar routines.

The routines operate as follows:

a. CALL RDTAPE (NUNIT, MØDE, NTYPE, BUF, LENGTH). Input a tape record from unit "NUNIT" in mode "MØDE" beginning at the core location specified by the address of "BUF" and continuing in consecutive core locations up to a maximum of "LENGTH" words. If the actual tape records are longer than "LENGTH", the entire record is read over but the excess words are not transferred into core.

MØDE:	0 = even parity no character conversion
	1 = odd parity no character conversion
	2 = even parity, convert from external BCD to internal machine BCD
NTYPE:	This argument is used to specify a type of record when records can appear on tape in different forms (such as whether the records were written by the home computer).
	Some other special options such as suppression of re-read after parity are available at NCAR.

Note that the call to RDTAPE or WRTAPE (see below) initiates the I/O operation and IØWAIT must be called to ensure that the data transfer is complete.

b. CALL IØWAIT (NUNIT, NSTATUS, NWØRDS). Wait for completion of the previous RDTAPE or WRTAPE call on unit "NUNIT". NSTATUS will return the status of the RDTAPE or WRTAPE operation and "NWØRDS" returns the actual number of words read or written. The NCAR system checks for a file mark first, then for end of tape, and then for parity.

NSTATUS:	0 - operation successful
	1 - file mark read
	2 - parity error
	3 - passed over EØT mark

c. CALL WRTAPE (NUNIT, MØDE, NTYPE, BUF, LENGTH)

Write a record where the arguments correspond to RDTAPE, and "LENGTH" specifies the actual number of words to write.

5.5 Tape copy program

A tape copy program is described in the NCAR Library Routines Manual. This Fortran program uses control card input to specify how many records or files are to be copied. The user can also look at each report in a subroutine, and then decide whether or not to copy the report to the output tape.

5.6 Blocking/Unblocking of short variable length data records

5.6.1 Introduction

RPTOUT is used to pack short variable length logical records into larger physical records on tape. The records are under checksum control. Logical records may be 1 to 352 words long, but the first 12 bits of each logical record must be reserved for use by RPTOUT. RPTIN is used to read logical records from a tape made by RPTOUT. The input or output of blocked records can be active on several tapes at once, but each active tape must have a separate 360-word buffer that is dimensioned in the calling program. Each buffer will hold a physical record that is being blocked in preparation for a tape write (Rptout) or being unblocked after a tape read (Rptin). There is enough information in the first few words of each buffer to give Rptout/Rptin sufficient information on the status of the blocking/unblocking from each I/O unit. Whenever Rptout gets a report that will not fit in the remainder of the current buffer, it writes a record and starts a new buffer with the report. Rptin initiates a tape read whenever it finishes unpacking a buffer.

We use RPTOUT to store many types of short reports such as rawinsonde data, surface data, and card image data (where the character code might be either internal or external BCD). By using RPTOUT to block observed upper air data from the weather centrals, the tape volume has usually been cut in half without changing the format of each report.

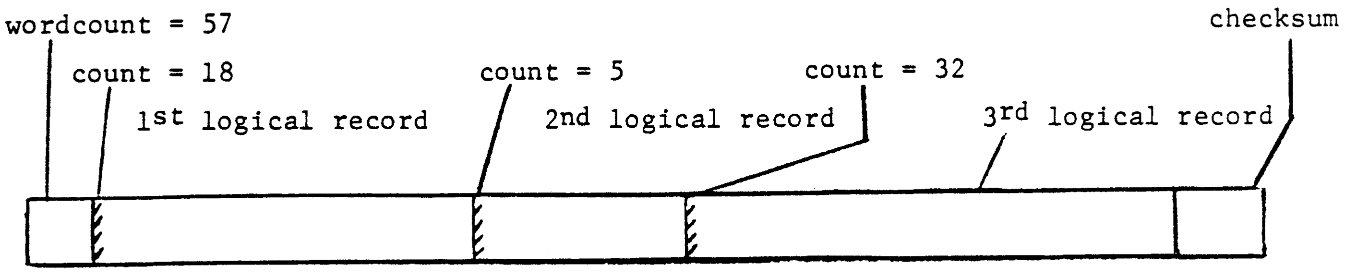
Data is obtained by calling RPTIN to unblock one record at a time from the physical records. Then the record can be decoded by using ordinary character, bit, or word decoding logic as appropriate.

Note that the first 12 bits of each logical record are used for a word-count of the report. Data put there will be erased.

On a machine with a different word length, the routine Gbytes (written for that machine) can be used to unblock a Rptout record written by the NCAR computer (see RptinX, Section 5.7). Once the report is unblocked it may then be written out by Rptout on that computer using a number of words that will contain the whole bit string in the unblocked report.

5.6.2 Layout of a physical record made by Rptout

Example layout of a physical record when the record has only three logical records with lengths of 18, 5 and 32 words respectively. The record then has the following form:



with the detailed format:

Word 1:	Has a right adjusted binary count of the words in the physical record. In this example the count is 1 + 18 + 5 + 32 + 1 = 57.
Word 2:	The leftmost 12 bits contain a count of 18 in binary saying that there are 18 words of data in the first logical record.
Words 2 through 19:	The remainder of word 2 and all of words 3-19 contain the data in this logical record.

Word 20:	Contains a count of 5 in the first 12 bits.
Words 20 through 24:	The data in the second logical record.
Word 25:	Contains a count of 32 in the first 12 bits.
Words 25 through 56:	The data in the third logical record.
Word 57:	Is a checksum of the data in words 1 through 56. This sum is made by the equivalent of an add-and-carry logical instruction.

If Rptout is run on another computer, the record layout will be the same as above; however, the wordcounts apply to each machine which may have word sizes of 32, 36, 48, 60, etc. bits. The NCAR CDC 6600/7600 have word sizes of 60 bits.

5.6.3 Method to use Rptin/Rptout

In general, a program reads or writes the Rptout logical records as if they are not blocked. The exceptions to this are as follows:

a. A special buffer dimensioned by 360 must be set aside by the calling program for use by RPTIN/RPTOUT in blocking and unblocking the logical records (reports). There must be a separate buffer for each unit in use at the same time.

b. There is no backspace command that applies to the logical records.

c. Before putting an end file on a tape, or otherwise terminating writes on a tape, a special RPTOUT command with JL = 2 is required (see below) to output the last buffer of data onto the tape. This also sets Nbuf(1) to zero.

d. After a rewind or before using an NBUF buffer for another tape, NBUF(1) must be set to zero, and the counters in NBUF may be re-initialized if desired.

We will now give more details on how to read and write RPTOUT logical records.

5.6.3.1 To write out a logical record

```
CALL RPTØUT (NUNIT,NBUF,LØCRPT,NWDS,JL)
```

The calling program provides the data in the arguments labelled in; the subroutine puts data in the arguments labelled out. For this subroutine we have:

```
CALL RPTOUT (in,work-out,in,in,in,)
```

NUNIT	Tape numbers 1-10 (not 5 or 6)
NBUF	An array in which the records are built: Dimension NBUF(360)
NBUF(1)	Must be 0 before the first write on a tape. RPTOUT will start a record when it sees the 0.
NBUF(2-4)	Must be cleared if you need to reinitialize the counters.
NBUF(2)	Will have a count of logical records output.
NBUF(3)	Will have a count of physical records output.
NBUF(4)	Will be a count of the words output. But words in reports = NBUF(4) - 2*NBUF(3).
LOCRPT	Location of the report for output. The first 12 bits of the report will be used by RPTOUT and RPTIN. Any data contained in the first 12 bits of the first word will be destroyed.
NWDS	Number of words in the report.
JL	= 0, output this report.
	= 2, no report to output; output the reports in NBUF onto the tape. This is used to output the last buffer onto the tape. This also sets NBUF(1) to 0.

5.6.3.2 RPTOUT error messages

a. Bad physical write, unit, physical record, length, status. This usually indicates some problems with the physical tape, tape drive, or machine, or tape writes are being attempted past the end of tape marker.

b. Bad logical length, unit, logical record #, length. The user tried to write either a zero-length record or a record longer than 352 words.

5.6.3.3 To read a logical record

```
CALL RPTIN (KUNIT,KBUF,KLØC,KWDS,JJ,KLMAX,JEØF)
```

The flow of data into or out of the subroutine is:

```
CALL RPTIN (in,work-in,out,out,in,in,out)
```

KUNIT	Tape 1-10 (not 5 or 6)
KBUF	An array to use for unpacking the records. Dimension KBUF(360). KBUF(1) must be set to 0 before the first read. RPTIN will start a new read when it sees the 0.
KBUF(2-6)	Must be cleared when desired.
KBUF(2)	Will have a count of logical records read.
KBUF(3)	Will have a count of physical records read.
KBUF(4)	Will have a count of words read from the tape.
KBUF(5)	Will have the words in this physical record.
KBUF(6)	Will be the same as JEOF.
KLOC	Array to put report in.
KWDS	The total number of words in this report. Note that this may be greater than KLMAX.
JJ	= 1, return only two words of this report and do not move on to the next report (except for EOF case).
	= 10, return full report up to a maximum of KLMAX words and position on next report. This is the option normally used.
KLMAX	Maximum number of words in a report to be moved to KLOC.
JEØF	= 0, good report returned.
	= 1, EOF
	= 2, report returned from a record with a bad checksum.

5.6.3.4 RPTIN error messages

a. Short physical record, unit, physical record, length, expected length, status. Bad tape read or records were not created by RPTOUT. This diagnostic often simply indicates noise in a record gap.

b. Bad physical length, unit, physical record, length, expected length, status. Bad tape read or records were not created by RPTOUT.

c. Bad checksum, unit, physical record, length, expected length, status. Bad tape read or records were not created by RPTOUT.

d. Bad logical length, unit, logical record, length. Probably the result of one of the previous errors.

e. This is not a RPTIN record, first word of the physical record (octal). Indicates problems which might cause execution errors if the record were processed. This current record is bypassed.

5.7 Unblock variable length RPTOUT records from another machine

We have written the program RPTINX (Rptin simulator) to make it easier for people with another computer to unblock our RPTOUT records that are output from a 60-bit machine. RPTINX will make it easy to exchange RPTOUT data between any two machines of reasonable capacity.

To use RPTINX, Gbytes must be available fo the user's computer. The word lengths of the output computer and the user's computer are specified in the program. Each call to Rptinx delivers one unblocked logical record to the calling program. The word count returned is a count of user words that is sufficiently large to contain the entire logical record. The first 12 bits in the report will still contain the word count from the originating machine. The user can either directly process the report, or use RPTOUT on his own machine to reblock it into a form more efficient to use repeatedly.

Rptinx assumes that the user's machine word length is at least 12 bits long. The version listed in the appendix does not check the checksum. To check our 60-bit checksum on a 32-bit machine, one would need to fold our 60-bit sum to become a 30-bit sum, and then use Gbytes to obtain 30-bit bytes as an aid in preparing a 30-bit checksum.

Note that buffer lengths must be defined large enough to contain the longest record written by the machine which created the RPTOUT records. In RPTOUT, the longest record output is six fewer words than the buffer length. Thus if the originating computer had a buffer length of 360 60-bit words, the length required to read the data on a 32-bit machine would be the number of 32-bit words necessary to contain 354 (360 minus 6) 60-bit words plus six words used for accounting by RPTINX. Thus a total of 670 32-bit words would be needed.

RPTIN and RPTINX were compared to show central processor timing differences on the CDC 6600. Both routines read the same 1000 logical records which varied in length from 10 to 50 words. This involved reading 89 physical records and a total of 30,178 words. There were a total of 30,000 words in all of the logical records. RPTIN required 194 ms to read these records including the end of file. RPTINX required 867 ms. Each of these times included 67 ms used in the tape reading routine or an average of 740 μs per physical record. The average time per 60-bit word for RPTIN is thus 6.5 μs including 2.2 μs for input; the average time per word for RPTINX is 28.9 μs including 2.2 μs for input. The checksums on the records were checked by RPTIN, but not by RPTINX.

5.8 How to process RPTOUT data on another machine

Suppose that information such as RAOB data is received on tape where the individual variable length reports have been blocked into longer records using the routine RPTOUT. The layout of a record (always odd parity) from RPTOUT was shown in section 5.6.2. First the record needs to be unblocked; in this process one report at a time is put into a separate buffer so that it can be decoded for use in calculations. The unblocking is done as follows:

a. On a CDC 6400/6600/7600 use our routines RPTIN (Appendix C-2), and the RDTAPE simulator (Appendix B) that is used by RPTIN. Each call to RPTIN will return one unblocked report into a buffer.

b. On a machine with a different word length, use RPTINX (Appendix D). See section 5.7 for a description of how to use RPTINX. The routine RPTINX uses the RDTAPE simulator (Appendix B) to input the physical records as necessary, and it uses GBYTES written for your machine to unblock each report. Each call to RPTINX will return one unblocked report into a buffer. The routine RPTOUT (Appendix C-1) may be used to reblock the data into a format that can be unblocked by RPTIN and is thus more efficient to use repeatedly.

The functions of the RDTAPE simulator are described in Section 5.4.

Unpacking a report

Once a report has been unblocked, it is a string of bits in memory that is ready to be unpacked to use for calculations. If the data is in a packed binary format, it is most convenient to use the routine GBYTES to unpack it. Sections 4.3 and 4.4 give some of the reasons for using packed binary formats, and Section 5.2 describes the use of GBYTES and gives examples.

Appendix E contains an example program (INRAB) that uses RPTIN to unblock a RAOB and then uses GBYTES to unpack it.

It is simple to adapt the program INRAB for use on machines with different word lengths. Cards are available for IBM-360/370 systems. The subroutine GBYTES has been written for these machines and for the Univac 1108. We are hoping to have GBYTES routines available for other machines in the near future.

5.9 Sorting Routines

5.9.1 SORTUU: Fortran sort - merge of variable length unblocked records

This program assumes that the records are unblocked on the input tape (Itape). The SORT subroutine reads logical records from the input tape and writes sorted strings alternately onto tapes NT1 and NT2. The MERGE subroutine merges strings back and forth between tapes (NT1, NT2) and (NT3, NT4) until the string count is one. The final output tape unit number will be NT1 or NT3 and will be printed. Information is also printed on the number of reports, words,and strings that are input and output.

The routines RDTAPE, WRTAPE and IOWAIT are used for input and output of records.

The subroutines GKEY (Irep,Key) must be provided to look at a report starting at Irep, and return a sort key starting in cell Key. The number of words used for the sort key must correspond to the constant NK, and must not exceed 10 unless the dimensions of LKey, Key 1, and Key 2 are changed. The first word of all sort keys must be positive. An integer comparison is used to determine sort order based on sort key words 2-10. Thus, a word is taken as negative if the sign bit is on.

<u>How to use</u>

a. Obtain a copy of the SORTUU program. The following changes in the cards are needed:

Dimension IREP1(LWDS), NARRAY(LENGTH)

MODE	= mode of tape. See writeup of RDTAPE in 5.4.
TYPE	= 0 for systems records.
LWDS	= maximum length of a logical record.
LENGTH	= size of work area for the sort. The sort is more efficient with a long work area unless the input records are short and nearly in sort. It should normally be set between 20,000 and 30,000 words long.
NK	= number of words in the sort key.

b. Write the subroutine GKEY.

<u>To merge two sorted tapes</u>

Mount the tapes on NT1 and NT2. Each must have an End of File. The tape ITAPE is not used and need not be mounted. Set KPASS = 1 in SORTUU. If the two tapes were not in sort, a message will be printed saying that the two tapes cannot be merged in one pass: the tapes NT1 and NT2 will not be written on.

5.9.2 SORTBB: Fortran sort - merge of variable length blocked records

This program assumes that the subroutine RPTOUT was used to block the variable length records onto the input tape (Itape). The SORTB subroutine reads logical records from the input tape and writes sorted strings alternately onto tapes NT1 and NT2. The MERGEB subroutine merges strings back and forth between tapes (NT1, NT2) and (NT3, NT4) until the string count is one. The final output tape number will be NT1 or NT3 and will be printed. Information is also printed on the number of reports, words, and strings that are input and output.

The RPTOUT-RPTIN subroutine package is used. These in turn use the routines RDTAPE, WRTAPE, and IOWAIT.

The subroutine GKEY (Irep, Key) must be provided to look at a report starting at Irep, and to return a sort key starting in cell Key. The number of words used for the sort key must correspond to the constant NK, and must not exceed 10 unless the dimensions of LKey, Key 1 and Key 2 are changed. The first word of all sort keys must be positive. An integer comparison is used for determining sort order when using sort key words 2-10.

<u>How to use</u>

a. Obtain a copy of the SORTBB program. The following changes in the cards are needed:

LWDS	= maximum length of a logical record
LENGTH	= size of work area for the sort. The sort is more efficient with a medium length work area. The length should usually be shorter if the data are nearly in sort. It should normally be set between 15,000 and 25,000 words long.
NK	= number of words in the sort key.

Dimension IREP1(LWDS), IREP2(LWDS), NARRAY(LENGTH)

b. Write the subroutine GKEY.

<u>To merge two sorted tapes</u>

Mount the tapes on NT1 and NT2. Each must have an End of File. The tape <u>Itape</u> is not used and need not be mounted. Set KPASS = 1 in SORTBB. If the two tapes were not in sort, a message will be printed saying that the two tapes cannot be merged in one pass: the tapes NT1 and NT2 will not be written on.

6. <u>The case for checksums on stored computer data</u>

To indicate the types of errors that can be introduced into computer data which cannot be detected by parity bit checking on magnetic tape, we will give three examples of errors that have occurred in the input/output of data. A number of other error cases have occurred, many of which could not be as accurately diagnosed as to what really did happen. Compared to the number of cases in which the input and output of data is reliable, the number of failure cases is certainly low. However, even a small number of errors is usually intolerable unless one knows that an error has occurred. Finally, we will end with some additional discussion as to why it is imperative that at least the primary data banks be kept under strict checksum control by the central arithmetic unit.

Example 1

We received three tapes from the Bureau of Commercial Fisheries that gave grid analyses of monthly mean Pacific Ocean sea surface temperatures. They used a 48-bit word machine to copy the data from their tapes to ours. The format of the data was also patterned after a 48-bit word, each of which gave the data at one grid point.

In processing the data, curve fitting on the grid was done to check for incorrect data points. On the basis of these checks, about 320 corrections were made.

When the Bureau of Commercial Fisheries compared these corrections with a print of their tape, they found that our corrected values corresponded closely with their printout. They also found that their values were exactly 256 (2.56°C) less than the uncorrected values which we had also listed. This suggests that the 9th bit from the right of some 48-bit words was periodically sticking on. In a few words, a different bit that set the sign of the value negative was incorrectly turned on.

By considering tape dumps and the word lengths of the computers, it became almost certain that the errors were made during the copy on the 48-bit machine. Perhaps the register through which the data passed on the way from memory to the tape drive was introducing the bit. However, even this theory isn't completely satisfying because the errors seem to show some preference for certain words within the physical records.

While it is not clear as to exactly which components of the hardware failed, it is clear that tape parity checking is of no help in detecting errors of this type that occur before the data is put on the tape. However, a checksum included in the format would made it possible to detect the records that do have such errors.

Example 2

The subroutines (RPTOUT and RPTIN) that block and unblock logical records were discussed in 5.6; these routines enable one to block short, variable length reports onto tape. The blocked records, which may contain data in either binary or BCD form, are on tape as odd parity records that include a checksum of the entire record.

In this example of errors during the input of data, a blocked record was read with no tape parity problems, but there was a checksum error. The tape drive had correctly read 78 bits, then it dropped 6 bits of octal 01, then it read 18 bits, then it dropped 6 bits of octal 01, and finally it read the rest of the approximately 20,000 bits in the record. Since the two 6-bit bytes that it dropped were identical in value, it is obvious that the horizontal tape parity would still check out all right. However, the data was essentially destroyed by being partly missing, and also by being shifted badly out of phase.

Example 3

On another computer, we once had a period where there was crosstalk between the tape channels and the microfilm output device. It turned out that in certain cases, the microfilm device would steal 12 bits of the data going to a tape drive. This caused all sorts of problems in data sets until the machine problem was diagnosed and the hardware fixed. It was especially bad since some of the information being processed was in even parity (BCD) records which were not under checksum control.

These examples should make it clear that it is very important, especially in the main data libraries, to keep information under a checksum control that is computed and verified by the central processor. Even if the hardware were much more reliable than it is, the checksums would still be very important since as multiprogramming capability is introduced in the newer computers (and the multiprogramming capability is very desirable), the software systems necessarily become more complex. Since these systems are both complex and changing, the possibility increases that errors may be introduced by software. We should also note the tendency for an increase of hardware between the I/O device and the central computer. This increase is often in the form of peripheral computers which feed the fast central computer. This, too, is often a desirable trend, but it does increase the possibility of errors.

We should also note that the checksums will help spot some program errors where the input/output buffering scheme is incorrect. This type of error was probably made on tapes that we received having northern hemisphere grids of weather data in a BCD format. Our error detection programs found a few grids that were obviously one type of data in the first random percentage of the grid, followed by another type of data in the rest.

A checksum can be easily added to any existing binary format. The question is how to protect information that one wants to keep in a BCD format. There is no really satisfactory answer to this question. We usually keep this data in binary mode with a binary checksum on tapes. But since the character coding is often in internal BCD code which is often different for difference machines, it is somewhat more difficult for a programmer to use the data on another machine. However, to send the data it can be recopied into BCD parity and the checksum dropped or an expensive BCD checksum could be calculated. Actually, with a simple subroutine it isn't very difficult to change the character codes from one set to another, thus the data could be sent in binary tape mode (odd parity) with a binary checksum included.

It is regrettable that even parity (BCD mode) ever got started on tapes. One minor irritation that this caused was that since a character code of 00 octal could not be used in even parity, a zero character had to be assigned another non-zero code (12 octal). However, a more important problem which resulted from the illegal 00 octal code in BCD parity was that binary information could not be included in even parity records. No standardization has been achieved with the even parity records that could not have been achieved as well or better with odd parity, where binary checksums can be recorded. It is only the standardization of the external (tape) BCD character set that has made it convenient for exchanging many types of data; this standardization could have been achieved just as well with odd parity tapes as noted in section 2.2.

We should insure that at least the major data sets are stored under a secure checksum control. In most cases this will probably mean that the records are written in odd parity mode with a binary checksum.

7. Guidelines for the exchange of information between computers

The following standards are suggested for the exchange of meteorological data between computers.

7.1 Magnetic tape: 1/2-inch standard tape

a. 7 channel tape: use 556 or 800 BPI.

b. 9 channel tape: use 800 or 1600 BPI.

There is some preference for 800 BPI because drives that can read at 1600 BPI aren't as common yet.

7.2 Tape label records: encourage them.

The first record is a standard tape label record and is followed by an end of file mark, often called a tape mark.

a. 7 channel tape--Label is usually even parity and standard external BCD. However, when the tape data is in odd parity, there is an advantage to having the label in odd parity also. The character code should then still be in external BCD or be specified. Some international standards call for all character data to be in odd parity; we don't know of any operating systems that would easily handle this practice.

b. 9 channel tape--Label is in EBCDIC or ASCII characters. Prefer EBCDIC because most machines use it. (Apparently most IBM systems use EBCDIC.) Certain writeups on standards still specify ASCII.

7.3 File structure on a tape

There may be more than one file of data on a tape. The last file of data should always be followed by at least two EOF (end of file) marks. Other files are separated by only one mark. The EOF's are the standard hardware EOF marks and not software labels to simulate these. The use of special end of file label records at the end of each file should be strongly discouraged.

7.4 Mixed mode tapes

With the possible exception of the initial label record, all the data on a tape should be written in one mode--either even or odd parity. Thus on an odd parity tape, there must not be any even parity label records after the initial label record.

7.5 Record length

The length of each physical record should not exceed about 5000 characters. If the logical records are longer than this, they should be segmented. Very short logical records should normally be blocked together into records at least 1000 characters long. Some computer systems now have trouble reading physical records that are not an even number of 8-bit words in length. This is an unreasonable hardware difficulty and should be fixed. It has been noted that if physical records are a multiple of 360 bits in length, they are an even number of words on machines with word lengths of 6, 8, 12, 18, 24, 36, 48, or 60 bits. The IBM 360 is an 8-bit machine for I/O purposes.

Some of the current standards indicate a maximum record length of 2048 characters. This length now seems unreasonably short and would demand splitting of analyses such as those from the National Meteorological Center.

7.6 Blocking shorter records together

When all blocked logical records in a file are the same length, no indication of that uniform length is required within the file. The last physical record in a block should not be padded out to be the standard length.

When variable length records are blocked, the length of each record (in characters, bytes, or words) shall be recorded as the first field in each record. This field shall be counted as part of the length of the record. The field length may be expressed as a decimal or binary number. But note that a binary number should not be used on an even parity tape.

At NCAR, the first 12 bits in each logical record are used for a binary word count and the physical records have a checksum at the end. In section 5.6.1 we discussed our blocking routine called RPTOUT.

7.7 Data structure within a logical record

Much of the satellite data, grid analyses, and other data will be available only in packed binary form. Computer routines and concepts are available which make the use of such data very easy on most binary machines. The floating point word structure of any one machine should generally <u>not</u> be used for data exchange. Use positive integers packed into a specified number of bits. Sign-magnitude integers are permitted, but are not as efficient as adding a bias to make all numbers positive. Thus, to pack numbers in the range of -99 to +99, add 100 and then pack each number into 8 or more bits. Often one can pack arrays of numbers by first subtracting a base value (the minimum value less an epsilon) to insure positive numbers and to reduce the size of them. Then each resulting number is divided by 2^n, where the scaling facter n is chosen so that the range of numbers will fit into the number of bits chosen to hold each packed number. The base value and the scaling factor should be packed with the data unless they are always constant. An unpacked number may be reconstructed according to the equation:

$$\text{Value} = \text{unpacked} \cdot 2^n + B$$

where B is the base value, and n is the scaling factor. If the binary point is assumed to the left of the packed number, the above equation must have another power of two factor to account for this fact. Some fields of data may have such a large range of values that a floating point representation must be used. Floating point numbers should be represented by two positive integers where one of the positive integers is the biased exponent and the other is the fraction. The bias on the fraction can be used to indicate the sign of the number, or the sign could be given separately.

Character data may be formatted into 4-, 6-, or 8-bit characters. Each digit should be recorded in its standard code and should not contain additional bits that are included to change the interpretation of the digit. The sign of each data element should be given in a separate character to the left of the digits and not to the right (or attached to the lower order digit) which are the practices in some computer systems.

7.8 Documentation

A detailed description of the format and data content of each data set must be available. This should include information about the volume of data as well as any necessary comments on data sources and data problems.

8. Word structure of various computers

This section shows how numbers are represented in fixed point form and in floating point form on various machines.

8.1 CDC 6400/6600/7600 (60-bit words, ones complement)

a. The floating point word format has one bit for sign, 11 bits for biased exponent, 48 bits for integer coefficient (normalized to the left, binary point to the right).

b. The exponent bias is 2000_8. The biased exponent of 2 can range from 0 to 3777. Numbers from about 3.13E-294 to 1.26E+322 can be represented in this format. Examples for exponents -1, -0, 0, 1 are:

2^{-1}	is coded as	1776
2^{-0}	"	1777
2^{0}	"	2000
2^{1}	"	2001

c. Examples of numbers are given in octal for the 60-bit words. The rather low exponents in the floating point words arise because the binary point is to the right. A dash means that the same digits are repeated.

	fixed		floating	
number	positive	negative	positive	negative
4	0---04	7---73	172240--0	605537--7
1	0---01	7---76	172040--0	605737--7
0	0---00	7---77	0-------0	7-------7

8.2 CDC 1604/3400/3600/3800 (48-bit words, ones complement)

a. The floating point word format has one bit for sign, 11 bits for biased exponent, and 36 bits for fraction (normalized to the left, binary point to the left).

b. The exponent bias is 2000_8. The biased exponent of 2 can range from 0 to 3777 as on the CDC 6600. The only change from the CDC 6600 is that -0 is sensed as +0 and is therefore biased by 2000 rather than 1777. Thus the biased exponent 1777 is never used.

c. Examples of numbers are given in octal for the 48-bit words. A dash means that the same digits are repeated.

number	fixed positive	fixed negative	floating positive	floating negative
4	0---04	7---73	200340--0	577437--7
1	0---01	7---76	200140--0	577637--7
0	0---00	7---77	0-------0	7-------7

8.3 IBM 360/370, Spectra 70 (32 or 64-bit words)

a. The floating point word format has 1 bit for sign, 7 bits for biased exponent (power of 16), 24 bits for the fraction (or 56 bits in long precision format). The fraction is normalized to the left, but since the exponent is in powers of 16, the normalization only insures that there is a bit in at least one of the 4 high-order bits of the fraction.

b. The exponent bias is 64_{10}. The biased range of powers of 16 is $0\text{-}127_{10}$ giving a power of 10 range from about -78 to +75.

c. Negative fixed point numbers are in twos complement notation (complement and add one). Negative floating point numbers are not complemented, but the sign bit is a 1.

d. Examples of numbers given in binary for 32 bit words. A dash means the same bits are continued.

number	positive	negative	floating S bit	floating positive 7 bits	floating positive 24 bits	negative
4	0-0100	1--1100	0	1000001	01000--0	same only sign bit on
1	0-0001	1--1111	0	1000001	00010--0	
0	0-0000	none	0	0000000	00000--0	

8.4 IBM 709/7094 (36-bit words, sign magnitude)

a. The floating point word format has 1 bit for sign, 8 bits for power of 2 biased exponent, 27 bits for the fraction (binary point on the left, number is normalized to the left).

b. The exponent bias is $128_{10} = 200_8$. Thus:

2^{-1}	is coded as	177_8
2^0	"	200_8
2^1	"	201_8

c. Examples of numbers given in octal for the 36-bit words. A dash means that the same digits are repeated.

	fixed		floating	
number	positive	negative	positive	negative
4	0---04	same only sign bit is on	20340--0	same only sign bit is on
1	0---01		20140--0	
0	0---00		00000--0	

8.5 UNIVAC 1108 (36-bit word, ones complement)

The word structure is exactly like that of the IBM 7094 except that negative numbers are complemented. Thus:

fixed point	-1	=	77----76
floating point	-1	=	57637-76

Appendix A

GBYTE, GBYTES, SBYTE, SBYTES

These are machine language listings of GBYTE, GBYTES, SBYTE, SBYTES written at NCAR for the CDC 6400/6600/7600 systems. These assume that the calling argument addresses are in the B-registers.

```
          ENTRY     GBYTE,GRYTES
C  CALL GRYTE (SOURCE,DEST,OFFSET,BYTE SIZE)
C  CALL GRYTES (SOURCE,DEST,OFFSET,RYTSIZE,SKIP,LOOP)
 GRYTES   CON       0
          SA4       GRYTES
          RX6       X4
          SA6       GBYTE
          SA3       R6
          SB6       X3-1        .SET UP LOOP COUNTER
          SA2       B5          .GET SKIP
          SB5       X2          .SVAE SKIP
          JP        REG
 GRYTE    CON       0
          SB6       B0
 REG      SA1       B1          .GET SOURCE WORD IN X1
          SR7       B2          .MOVE DESTINATION ADDRESS
          SA4       B3          .GET CALCULATED OFFSET IN X4
          SA5       R4          .GET BYTE SIZE IN X5
          SR2       X5-1        .B2=BYTE SIZE-1 (SET FOR GENERATING MASK)
          SX3       60          .CONSTANT OF 60
 LOOP     SA2       A1+1        .GET SECOND WORD (WHETHER NEEDED OR NNOT)
          IX7       X3-X4       .ACTUAL OFFSET = 60-CALC OFFSET
          SB3       X7          .PUT INTO B3
          MX0       1           .FORM THE
          AX0       B2,X0       .  MASK IN X0
          IX6       X7-X5       .FIND IF RYTE IS JUST IN WORD
          SB1       X6          .   PUT INTO B1
          LX0       B3,X0       .LINE UP MAXK
          BX1       X0*X1       .GET THE BYTE ISOLATED IN X1
          PL        B1,RTJST    .IF WE ARE DONE,JUMP
          MX0       1           .IF
          SB3       -B1         .  NOT
          SR3       B3-1        .     MAKE UP
          AX0       R3,X0       .        SECOND MASK
          RX2       X2*X0       .ISOLATE 2ND PART OF RYTE
          RX1       -X0*X1      .CHOP OFF TRASH IN 1ST PART OF BYTE
          RX1       X1+X2       .COMBINE PORTIONS
 RTJST    ZR        X4,LEFT     .IF SIGN BIT IS INVOLVED,JUMP
          AX6       B1,X1       .RIGHT JUSTIFY THE BYTE
 STR      SA6       R7          .STORE IT IN OUTPUT FIELD
          ZR        B6,GRYTE    .IF DONE,LEAVE
          SB6       R6-1        .IF NOT DONE COUNT
          SR7       B7+1
          PX7       X3          .FLOAT 60  =  B
          IX4       X4+X5
          SX4       X4+R5       .COFFSET = OLD OFFSET+ BYTE SIZE +SKIP SIZ
          NX7       X7          .NORMALIZE B
          PX6       X4          .FLOAT NEW COFFSET (IN BITS)  =  A
          NX6       X6          .NORM  A
          FX2       X6/X7       .Q = A/B  .GET QUOTIENT INX2
          UX2       R1,X2       .
          LX2       B1,X2       .GET INTEGER QUOTIENT
          SR1       X2          .PUT IN B1
          PX2       X2          .REPACK AND
          NX2       X2          .  NORMALIZE
          FX4       X7*X2       .X4 = B*Q  .GET INTEGER PRODUCT
          FX4       X6-X4       .R = A - B*Q  (GET REMAINDER IN X4)
          UX4       B3,X4       .GET INTEGER
          LX4       B3,X4       .     REMAINDER IN X4 = NEW CALC OFFSET)
          SA1       A1+B1       .GET NEW WORD IN
          JP        LOOP        .GO LOOP
 LEFT     SR1       X5          .GET RYTE SIZE
          LX6       R1,X1       .RIGHT JUSTIFY THE RYTE
          JP        STR         .GO STORE IT
          END
```

```
          ENTRY     SBYTE,SBYTES
C     CALL SBYTE  (DEST,SOURCE,OFFSET,BYTE SIZE)
C     CALL SBYTES (DEST,SOURCE,OFFSET,RYTESIZE,SKIP,LOOP)
C   THIS ASSUMES CALLING ARG ADDRESSES ARE IN B REGISTERS
C   ON SOME CDC6400/6600 SYSTEMS THE CON STATEMENTS ARE CALLED DATA
 SBYTES   CON       0
          SA4       SBYTES
          BX7       X4
          SA7       SBYTE
          SA3       B6
          SR6       X3-1        .IF LONG CALL,SET UP LOOP COUNTER
          SA2       B5          .GET SKIP
          SR5       X2          .SAVE SKIP
          JP        REG
 SBYTE    CON       0
          SB6       B0
 REG      SA1       B1          .GET 1ST DESTINATION WORD IN X1
          SR7       B2          .MOVE SOURCE ADDRESS
          SA4       B3          .GET CALCULATED OFFSET IN X4
          SA5       B4          .GET BYTE SIZE IN X5
          SR2       X5-1        .B2=RYTE SIZE - 1 (SET FOR GENERATING MASK
          SX3       60          .CONSTANT OF 60.
          SB4       X5          .BYTE SIZE ALSO IN B4
 LOOP     SA2       A1+1        .GET SECOND DEST WORD (WHETHER NEEDED OR N
          IX7       X3-X4       .ACTUAL OFFSET = 60 - CALC OFFSET
          SB3       X7          .PUT INTO B3
          SA3       B7          .GET SOURCE WORD
          SB1       B3-R4       .FIND IF RYTE IS SPLIT OR NOT
          BX6       X3          .DOUBLE IMAGE
          BX7       X3          .  OF SOURCE BYTE  TO X6 AND X7
          NG        B1,NGC
          MX0       1           .FORM 1ST MASK
          AX0       R2,X0       .   IN X0
          LX0       B4,X0       .RIGHT JUSTIFY MASK
          LX0       B1,X0       .  AND MASK
 RET      LX6       B1,X6       .LINE UP BYTE
          BX1       -X0*X1      .MAKE HOLE IN OUTWORD
          BX6       X0*X6       .TAKE CARE OF ANY SIGN EXTENSION
          BX6       X1+X6       .COMBINE THE BYTES
          SA6       A1          .STORE THE WORD
          PL        B1,DONE     .IF B1 IS NOT NEG, WE ARE DONE
          SX1       B4-B3       .2ND MASK SIZE = B.S. - ACTUAL OFFSET
          SX3       60          .RETURN CONSTANT
          IX6       X3-X1       .LEFT SHIFT = 60 - 2ND MASK SIZE
          SB1       X1-1        .MAKE
          MX0       1           .  2ND
          AX0       B1,X0       .     MASK
          SR1       X6          .LEFT SHIFT
          LX7       R1,X7       .  BYTE
          BX2       -X0*X2      .MAKE HOLE IN OUT WORD
          RX7       X0*X7       .KNOCK OFF TRASH
          BX7       X2+X7       .COMBINE
          SA7       A2          .STORE
 DONE     ZR        B6,SRYTE    .IF DONE,LEAVE
          SR6       B6-1        .IF NOT DONE COUNT
          SB7       R7+1
          SX3       60
          PX7       X3          .FLOAT 60  =  B
          IX4       X4+X5
          SX4       X4+B5       .COFFSET = OLD OFFSET+ BYTE SIZE +SKIP SIZ
          NX7       X7          .NORMALIZE B
          PX6       X4          .FLOAT NEW COFFSET (IN BITS)  =  A
          NX6       X6          .NORM  A
          FX2       X6/X7       .Q = A/R  .GET QUOTIENT INX2
          UX2       R1,X2       .
          LX2       B1,X2       .GET INTEGER QUOTIENT
          SR1       X2          .PUT IN R1
          PX2       X2          .REPACK AND
          NX2       X2          .  NORMALIZE
          FX4       X7*X2       .X4 = B*Q  .GET INTEGER PRODUCT
          FX4       X6-X4       .R = A - B*Q  (GET REMAINDER IN X4)
          UX4       B3,X4       .GET INTEGER
          LX4       B3,X4       .     REMAINDER IN X4 = NEW CALC OFFSET)
          SA1       A1+B1       .GET NEW WORD IN
          JP        LOOP        .GO LOOP
 NGC      SB2       B3-1        .GET SHIFT CONSTANT
          MX0       1
          AX0       R2,X0       .FORM THE MASK IN X0
          LX0       B3,X0       .RIGHT JUSTIFY MASK
          SR2       B4-1        .RESET B2
          JP        RET
          END
```

Appendix B

RDTAPE, WRTAPE, IOWAIT

These routines use buffer statements to simulate the I/O statements used in various NCAR data processing programs. The routines RDTAPE and WRTAPE are often used at NCAR because they are somewhat more general than the buffer statements.

```
SUBROUTINE RDTAPE(NUNIT,MODE,NTYPE,BUFFER,LENGTH)
DIMENSION BUFFER(1)
IF (MODE .GT. 1) MODE=MODE-2
BUFFER IN(NUNIT,MODE)(BUFFER(1),BUFFER(LENGTH))
RETURN
END
```

```
SUBROUTINE WRTAPE(NUNIT,MODE,NTYPE,BUFFER,LENGTH)
DIMENSION BUFFER(1)
IF (MODE .GT. 1) MODE=MODE-2
BUFFER OUT(NUNIT.MODE)(BUFFER(1),BUFFER(LENGTH))
RETURN
END
```

```
    SUBROUTINE IOWAIT(NUNIT,NSTATE,NWORDS)
10  IF (UNIT.NUNIT)10.11.12.13
11  NSTATE=0
    NWORDS=LENGTHF(NUNIT)
    RETURN
12  NSTATE=1
    RETURPN
13  NSTATE=2
    NWORDS=LENGTHF(NUNIT)
    RETURN
    END
```

Appendix C-1

Listing of RPTOUT

This routine blocks short variable length reports into a buffer (NBUF) that is dimensioned in the main program. It outputs a record from the buffer whenever the buffer is too full to hold another report. If the dimension of NBUF is changed in order to accept longer logical records, one must also set KMAX equal to the dimension and set KLMAX equal to the dimension minus 8.

```
*FORTRAN
      SUBROUTINE RPTOUT(NUNIT,NBUF,LOCRPT,NWDS,JL)
C JULY 71 VERSION FOR NON NCAR SYSTEMS        D JOSEPH
C REQUIRES  A COMPATIBLE SBYTE ROUTINE AND WRTAPE SIMULATOR
      DIMENSION LOCRPT(1)
      DIMENSION NBUF(360)
      DATA (KMAX = 360)
      DATA (KLMAX = 352)
C  THIS ROUTINE PACKS LOGICAL RECORDS INTO A PHYSICAL RECORD
C  THE WORDCOUNT IS PUT INTO 1ST 12 BITS OF REPORT
C  KLMAX IS THE MAX LENGTH OF A LOG RECORD
C  NBUF(1) IS 1ST AVAIL CELL IN BUFFER.  MAKE IT 8 BEFORE 1ST WRITE
C  NBUF(2) WILL HAVE COUNT OF LOGICAL RECORDS OUTPUT
C  NBUF(3)  WILL BE COUNT OF PHYSICAL RECORDS OUTPUT
C  NBUF(4) IS COUNT OF WORDS OUTPUT. WDS IN RPTS = NBUF(4) -2*NBUF(3).
C  NBUF(7) IS THE 1ST WORD FOR THE RECORD.
C  JL = 2 MEANS NO REPORT. ONLY OUTPUT LAST RECORD
C  USER MUST END FILE AND REWIND TAPES
C  WRITTEN BY ROY JENNE AT NCAR IN MAY 1966
      IF (JL .EQ. 2)  GO TO 60
      IF (NBUF(1) .NE. 8)  GO TO 4
      NBUF(1) = 8
    4 IF (NWDS .GT. KLMAX)  GO TO 70
      IF (NWDS .LE. 0) GO TO 70
   10 NN = NBUF(1) + NWDS
      IF(NN .GT. KMAX)  GO TO 60
C  MOVE THE REPORT INTO THE BUFFER
      K = NBUF(1)
      CALL SBYTE(LOCRPT,NWDS,0,12)
      DO 50 I=1,NWDS
   50 NBUF(K+I-1)=LOCRPT(I)
      NBUF(2) = NBUF(2) + 1
      NBUF(1) = NN
      RETURN
C
C  OUTPUT A RECORD HERE
   60 IF(NBUF(1) .GT. KMAX)  GO TO 70
      IF (NBUF(1) .EQ. 8)  RETURN
      KKK = NBUF(1)
      KKDAT = KKK-7
      KKWDS = KKDAT+1
      NBUF(7) = KKWDS
      IEND=KKDAT+6
      NBUF(KKK)=0
      DO 59 I=7,IEND
   59 NBUF(KKK)=NBUF(KKK)+NBUF(I)
      IF (KKWDS .LT. 3)  GO TO 70
      CALL WRTAPE(NUNIT,1,0,NBUF(7),KKWDS)
      NBUF(3) = NBUF(3) + 1
      NBUF(4) = NBUF(4) + KKWDS
      NBUF(1) = 8
      CALL IOWAIT(NUNIT,NSTATE,NWDWRT)
      IF (NSTATE .NE. 0)  GO TO 61
      IF (NWDWRT .EQ. KKWDS)  GO TO 63
   61 PRINT 962,NUNIT,NBUF(3),NWDWRT,NSTATE
  962 FORMAT (* ***RPTOUT-BAD PHYSICAL WRITE,UNIT,PHYS REC,LENGTH,STATUS
     2 *,4I8)
   63 IF (JL .NE. 2)  GO TO 10
   65 NBUF(1) = 8
      RETURN
C  HERE FOR RECORD LENGTH MYSTERY
   70 PRINT 960,NUNIT,NBUF(2),NWDS
  960 FORMAT (* ***RPTOUT-BAD LOGICAL LENGTH  ,UNIT,LOG REC,LTH *,3I8)
      GO TO 65
      END
```

Appendix C-2
Listing of RPTIN

This routine unblocks variable length reports from records that are read into a buffer (NBUF) that is dimensioned in the main program. The input records were created by RPTOUT on the user's machine or on a machine with the the same word length as the user's machine. If one must change the buffer size in order to accept a larger record from another machine, he must change the NBUF dimension and set KMAX equal to the dimension minus 6.

```
      SUBROUTINE RPTIN(NUNIT,NBUF,LOCRPT,NWDS,JJ,KLMAX,JEOF)
C JULY 71 VERSION FOR NEW NCAR SYSTEMS      D JOSEPH
C REQUIRES  A COMPATIBLE GBYTE ROUTINE AND RDTAPE SIMULATOR
      DIMENSION LOCRPT(1)
      DIMENSION NBUF(360)
C  THIS ROUTINE UNPACKS LOGICAL RECORDS WRITTEN BY RPTOUT
C  NBUF(1) MUST BE SET TO 0 BEFORE 1ST READ ON TAPE
C  NBUF(2) WILL HAVE A COUNT OF LOGICAL RECORDS READ
C  NBUF(3) WILL HAVE A COUNT OF PHYSICAL RECORDS READ
C  NBUF(4) WILL HAVE A COUNT OF WORDS READ FROM TAPE
C  NBUF(5) HAS WORDS IN THIS RECORD
C  NBUF(6) IS THE RECORD STATUS. 0=GOOD,1=EOF,2=BAD CHKSUM -RECORD IGNOR
C  NBUF(7) IS THE FIRST WORD OF THE RECORD. IT HAS WDCNT OF RECORD
C  IF JJ IS 1 WE DONT MOVE ON TO NEXT REPORT EXCEPT FOR EOF
C  IF JJ IS 11 RETURN FULL RPT UP TO MAX OF KLMAX WORDS
C  WRITTEN BY ROY JENNE AT NCAR IN JUNE 1966
      DATA KMAX / 354/
      IF (NBUF(1).EQ.0) GO TO 3
      IF (NBUF(1) .NE. 7)  GO TO 40
C  HERE TO LOOK AT A NEW RECORD
      GO TO 6
    3 CALL IOWAIT (NUNIT,NBUF(6),NWORED)
    5 CALL RDTAPE (NUNIT,1,1,NBUF(7),KMAX)
    6 CALL IOWAIT (NUNIT,NBUF(6),NWORED)
      NBUF(1) = 8
      JEOF = NBUF(6)
      IF (JEOF .EQ. 1)  GO TO 42
      IF (NWORED.GT.2) GO TO 10
      IEREC=NBUF(3)+1
      PRINT 900,NUNIT,IEREC,NWORED,NBUF(7),NBUF(6)
  900 FORMAT (* ***RPTIN-SHORT PHYS REC,UNIT,PHYS REC,LTH,EXP LTH,STATUS
     2 *,5I6)
      GO TO 5
   10 NBUF(5) = NWORED
      IF (NBUF(7) .EQ. NWORED) GO TO 18
      IEREC=NBUF(3)+1
      PRINT 902,NUNIT,IEREC,NWORED,NBUF(7),NBUF(6)
  902 FORMAT (* ***RPTIN-BAD PHYS LTH,UNIT,PHYS REC,LTH,EXP LTH,STATUS *
     2,5I6)
      IF (NBUF(7) .LE. 2 .OR. NBUF(7) .GT. 354) GO TO 55
      IF(NWORED.LT.NBUF(7)) GO TO 18
      NBUF(5) = NBUF(7)
   18 KKWDS = NBUF(5) - 1
C  KKJ IS THE NBUF INDEX FOR THE CHKSUM WORD
      KKJ = KKWDS + 7
      IEND=KKWDS+6
      NSUM=0
      DO 101 I=7,IEND
  101 NSUM=NSUM+NBUF(I)
      NBUF(6) = 0
      IF (NBUF(KKJ) .EQ. NSUM)  GO TO 20
      IEREC=NBUF(3)+1
      PRINT 906,NUNIT,IEREC,NWORED,NBUF(7),JEOF
  906 FORMAT (* ***RPTIN-BAD CHECKSUM,UNIT,PHYS REC,LTH,EXP LTH,STATUS *
     2,5I8)
      NBUF(6) = 2

   20 NBUF(3) = NBUF(3) + 1
      NBUF(4) = NBUF(4)+NBUF(5)
C
C  HERE TO GET A LOGICAL RECORD
   40 JEOF = NBUF(6)
      K = NBUF(1)
      CALL GBYTE(NBUF(K),NWDS,0,12)
      IF (JJ .NE. 1) GO TO 44
      DO 104 I=1,2
  104 LOCRPT(I)=NBUF(K+I-1)
   42 RETURN
C  GET A FULL LOGICAL RECORD
   44 CONTINUE
      IF (NWDS .GT. 0 .AND. NWDS+NBUF(1) .LE. NBUF(5)+6) GO TO 46
      IEREC=NBUF(2)+1
      PRINT 904,NUNIT,IEREC,NWDS
  904 FORMAT (* ***RPTIN-BAD LOGICAL LENGTH,UNIT,LOG REC,LTH *,3I8)
      GO TO 3
   46 CONTINUE
C  NBUF(5) + 6  IS THE INDEX FOR THE CHKSUM WORD
      LK = NWDS
      IF (NWDS.LT.KLMAX) GO TO 48
      LK = KLMAX
   48 CONTINUE
      DO 106 I=1,LK
  106 LOCRPT(I)=NBUF(K+I-1)
      NBUF(2) = NBUF(2)+1
      NBUF(1) = NBUF(1)+NWDS
      IF (NBUF(1) .EQ. NBUF(5) + 6)  GO TO 52
      RETURN
   52 NBUF(1) = 7
      CALL RDTAPE (NUNIT,1,1,NBUF(7),KMAX)
      RETURN
   55 CONTINUE
      PRINT 1001,NBUF(7)
 1001 FORMAT(*  THIS IS  NOT A RPTIN RECORD  *,O20)
      JEOF=2
      GO TO 52
      END
```

Appendix D

RPTINX

The following is a listing of the routine RPTINX which is used to read RPTØUT records created on a machine with a word length different from that of the reading machine. In general, this routine operates like RPTIN. NBUF is dimensioned in the calling routine and must be large enough to contain the longest record written. IMAX is set to the dimension of NBUF and KMAX=IMAX-6. The parameters LIN and LØT must be set to the appropriate word lengths.

```
*FORTRAN
      SUBROUTINE RPTINX(NUNIT,NBUF,LOCRPT,NWOS,JJ,KLMAX,JEOF)
C TO READ RPTOUT RECORDS WRITTEN ON A MACHINE OF WORD LENGTH $LIN$ BITS IN TO
C A MACHINE OF WORD LENGTH $LOT$ BITS    - BUFFER LENGTH IN IS IMAX
C LOT WORDS - IE LARGE ENOUGH FOR THE BUFFER SIZE IN LIN BIT WORDS
C ALL LENGTHS IN THE CALLING SEQUENCE ARE $LOT$ BIT WORDS
C NCAR    2 JUL 71      D JOSEPH
      DIMENSION NBUF(1),LOCRPT(1)
      DATA IMAX/360/
      DATA KMAX/354/
      DATA LIN/72/
      DATA LOT/60/
      COMMON/IND/LOT
      IF (NBUF(1) .EQ. 0) GO TO 3
      IF (NBUF(1) .NE. 7) GO TO 40
      GO TO 6
    3 CALL ICWAIT(NUNIT,NBUF(6),NWRED)
      XL=LOT
      XF=LIN
      XF=XL/XF
      YF=1./XF
      ID=LIN-LOT
      LII=7
      LGET=LIN
      LOFF=0
      IF (LOT .GE. LIN) GO TO 4
      LOFF=ID
      LGET=LOT
      XOFF=LOFF
      XP=XOFF/XL
      IP=XP
      LOFF=(XP-IP)*XL+.01
      LII=IP+7
    4 CONTINUE
    5 CALL RDTAPE(NUNIT,1,0,NBUF(7),KMAX)
    6 CALL ICWAIT(NUNIT,NBUF(6),NWRED)
      IBITS=LIN
      NBUF(1)=0
      JEOF=NBUF(6)
      IF (JEOF .EQ. 1) GO TO 42
      NTST=XF*NWRED+.01
      IF (NTST .GT. 2) GO TO 10
      IEREC=NBUF(3)+1
      PRINT 960,NUNIT,IEREC,NWRED,NBUF(6)
  960 FORMAT(* +++ RPTINX SHORT PHYS REC,UNIT,PHYS REC,EXP LTH *,4I8)
   10 NBUF(5)=NWRED
      CALL GBYTE(NBUF(LII),IWOS,LOFF,LGET)
      NWU=XF*NWRED
      NWL=NWU-XF+.99
      IF (IWOS .GE. NWL .AND. IWOS .LE. NWU) GO TO 18
      IEREC=NBUF(3)+1
      PRINT 962,NUNIT,IEREC,NWRED,IWOS,NBUF(6)
  962 FORMAT(* +++ RPTINX - BAD PHYS LTH,UNIT,PHYS REC , LTH, EXP LTH,
     2 STATUS*,5I8)
      GO TO 5
   18 CONTINUE
      NBUF(6)=0
      NBUF(3)=NBUF(3)+1
      NBUF(4)=NBUF(4)+NWRED
      LBITS=LIN*(IWOS-1)
   40 JEOF=NBUF(6)
      XBITS=IBITS
      XP=XBITS/XL
      IP=XP
      IOF=(XP-IP)*XL+.01
      II=IP+1
      CALL GBYTE(NBUF(II+6),NWOS,IOF,12)
      IF (NWOS .GT. 0 .AND. NWOS .LT.(IWOS-2)) GO TO 46
      IEREC=NBUF(2)+1
      PRINT 964,NUNIT,IEREC,NWOS
  964 FORMAT(* +++ RPTINX- BAD LOGICAL LENGTH,UNIT,LOG REC,LTH *,5I8)
      GO TO 3
   46 CONTINUE
      LTH=XF*NWOS+.99
      IBITS=IBITS+LIN*NWOS
      NWOS=YF*NWOS+.99
      CALL GBYTES(NBUF(II+6),LOCRPT,IOF,LOT,0,NWOS)
      NBUF(2)=NBUF(2)+1
      NBUF(1)=NBUF(1)+NWOS
      IF (IBITS .GE. LBITS) GO TO 52
      RETURN
   52 NBUF(1)=7
      CALL RDTAPE(NUNIT,1,0,NBUF(7),KMAX)
   42 CONTINUE
      RETURN
      END
```

Appendix E

Routine to Unblock and Unpack Rawinsonde Reports at NCAR

The call to RPTIN unblocks each rawinsonde report (raob). The calls to GBYTE/ GBYTES unpack the raob. Part of an example print for station number 3 (permanent ship C) is listed below: The data is in order of decreasing pressure, except that the surface pressure may be first as in this example. The height at the surface level should have been coded as missing. When the data is used in calculations, the bits giving the units for wind speed and moisture must also be unpacked.

```
      PROGRAM INRAB
      DIMENSION IBUF(360), LOC(180),IPRESS(50),INT(50),ITEMP(50),
     X  IRH(50),IDD(50),IFF(50)
      ITAPE = 2
      REWIND ITAPE
    4 CALL RPTIN(ITAPE,IBUF,LOC,INOS,10,180,IEOF)
       IF (IEOF .EQ. 1) GO TO 900
      NREC = NREC + 1
      IF (IEOF .EQ. 0) GO TO 15
      PRINT 936,NREC,INOS
  934 FORMAT(1H ,*REDUND*, 2I7)
   15 CONTINUE
      CALL GBYTE (LOC,NFORM,12,6)
      CALL GBYTE(LOC,NSTA,18,17)
      CALL GBYTE(LOC,NYR,35,7)
      CALL GBYTE(LOC,NO,42,4)
      CALL GBYTE(LOC,NDY,46,5)
    3 CONTINUE
      CALL GBYTE(LOC,NHR,51,5)
      CALL GBYTE(LOC,NLAT,56,11)
      CALL GBYTE(LOC(2),NLON,7,12)
      CALL GBYTE(LOC(2),NELEV,19,14)
      CALL GBYTE(LOC(2),NSORC,33,7)
      CALL GBYTE(LOC(2),NINO,46,3)
      CALL GBYTE(LOC(2),LVL,49,6)
      CALL GBYTES(LOC(3),IPRESS,6,11,61,LVL)
      CALL GBYTES(LOC(3),INT,17,16,56,LVL)
      CALL GBYTES(LOC(3),ITEMP,33,11,61,LVL)
      CALL GBYTES(LOC(3),IRH,44,11,61,LVL)
      CALL GBYTES(LOC(3),IDD,55,9,63,LVL)
      CALL GBYTES(LOC(4),IFF,4,8,64,LVL)
      DO  40  I=1,LVL
      INT(I) = INT(I) - 1000
      ITEMP(I) = ITEMP(I) - 1000
      IRH(I) = IRH(I) - 1000
   40 CONTINUE
      PRINT 936,NREC,NFORM,NSTA,NYR,NO,NDY,NHR,NLAT,NLON,NELEV,NSORC,
     X  NINO,LVL
  936 FORMAT (1H ,*STATION IDENT*, 13I8)
      DO 50  I=1,LVL
      PRINT 937,NREC,I  ,IPRESS(I),INT(I),ITEMP(I),IRH(I),IDD(I),IFF(I)
  937 FORMAT(1H ,*TEST*, I8,I3,5X, 6I7)
   50 CONTINUE
   51 KL = KL + LVL
      GO TO 4
  900 CONTINUE
      PRINT 960,NREC,KL,(IBUF(I),I=2,4)
  960 FORMAT (1H ,*DONE*, 5I8)
      CALL EXIT
      END
```

```
STATION IDENT        4         1         3        60         1         4
TEST         4  1       992         0         4        87        60        11
TEST         4  2      1000       -43       990       990       500       250
TEST         4  3       950       330       -20        87        50         9
TEST         4  4       900       778       990        88        47        11
TEST         4  5       850      1224       -40        88        52        12
TEST         4  6       800      1693      -107        87       109         7
TEST         4  7       750      2200      -128        83       152         6
TEST         4  8       700      2712       990        81       176         7
TEST         4  9       650      3280      -159        80       183         7
TEST         4 10       600      3872      -181        78       182         5
TEST         4 11       550      4508      -213        75       160         9
TEST         4 12       500      5215       990        71       137        27
TEST         4 13       450      5970      -301        66       152        23
TEST         4 14       400      6797      -365        61       170        37
```

Appendix F

Subroutines GBYTE and GBYTES for IBM 360 Systems

```
STMT   SOURCE STATEMENT                                          FOOS CL3-8 01/17/72

  1 *
  2 *       ****  DOCUMENTATION  ****
  3 *
  4 *  THE FOLLOWING ROUTINES ALLOW FORTRAN ACCESS TO BIT STRINGS (BYTES)
  5 *  OF ARBITRARY LENGTH AND POSITION, PERHAPS CROSSING WORD BOUNDARIES,
  6 *  IN THE MANNER SPECIFIED BELOW:
  7 *
  8 *     CALL GBYTE (PCKD,UNPK,INOFST,NBIT)
  9 *     CALL GBYTES(PCKD,UNPK,INOFST,NBIT, NSKIP,ITER)
 10 *
 11 *          PCKD:    THE FULLWORD IN MEMORY FROM WHICH UNPACKING IS TO
 12 *                   BEGIN; SUCCESSIVE FULLWORDS WILL BE FETCHED AS
 13 *                   REQUIRED.
 14 *          UNPK:    THE FULLWORD IN MEMORY INTO WHICH THE INITIAL BYTE
 15 *                   OF UNPACKED DATA IS TO BE STORED; SUBSEQUENT BYTES
 16 *                   WILL BE STORED INTO SUCCESSIVE FULLWORDS AS
 17 *                   REQUIRED.
 18 *          INOFST:  A FULLWORD INTEGER SPECIFYING THE INITAL OFFSET
 19 *                   IN BITS OF THE FIRST BYTE, COUNTED FROM THE
 20 *                   LEFTMOST BIT IN PCKD.
 21 *          NBITS:   A FULLWORD INTEGER SPECIFYING THE NUMBER OF BITS
 22 *                   IN EACH BYTE TO BE UNPACKED.  LEGAL BYTE WIDTHS
 23 *                   ARE IN THE RANGE 1 - 32; BYTES OF WIDTH .LT. 32
 24 *                   WILL BE RIGHT JUSTIFIED IN THE LOW-ORDER POSITIONS
 25 *                   OF THE UNPK FULLWORDS, WITH HIGH-ORDER ZERO FILL.
 26 *          NSKIP:   A FULLWORD INTEGER SPECIFYING THE NUMBER OF BITS
 27 *                   TO SKIP BETWEEN SUCCESSIVE BYTES.  ALL NON-NEGATIVE
 28 *                   SKIP COUNTS ARE LEGAL.
 29 *          ITER:    A FULLWORD INTEGER SPECIFYING THE TOTAL NUMBER OF
 30 *                   BYTES TO BE UNPACKED, AS CONTROLLED BY INOFST,
 31 *                   NBIT AND NSKIP ABOVE.   ALL NON-NEGATIVE ITERATION
 32 *                   COUNTS ARE LEGAL.
 33 *
 34 *            NOTES ...
 35 *               1)  A MULTIPLE-BYTE ACCESS (GBYTES) WITH ITER=0 (BUT
 36 *                   NOT 1)  IS EXACTLY EQUIVALENT TO A SINGLE-BYTE
 37 *                   ACCESS (GBYTE).
 38 *               2)  AN ERROR DETECTED IN THE CALLING SEQUENCE OF
 39 *                   EITHER GBYTE OR GBYTES SUPPRESSES BYTE ACCESS,
 40 *                   AND SETS THE FIRST ELEMENT OF UNPK = X'FFFFFFFF'
 41 *
 42 *
 43 *
 44 *       ****  REGISTER USAGE  ****
 45 ZERO     EQU   X'0'                      CONSTANT ZERO
 46 PCKD     EQU   X'1'                      ADDR OF PACKED ARY ELEMENT
 47 UNPK     EQU   X'2'                      ADDR OF UNPACKED ARY ELEMENT
 48 OFST     EQU   X'3'                      INITIAL BIT OFFSET
 49 NBIT     EQU   X'4'                      BYTE LENGTH IN BITS (.LE.32)
 50 NSKP     EQU   X'5'                      ITERATIVE BIT SKIP
 51 ITER     EQU   X'6'                      ITERATION COUNT (.GE.0)
 52 BITR     EQU   X'7'                      BITS REMAINING IN HIWD
 53 BITS     EQU   X'8'                      BITS TO SHIFT (OFST OR NSKP)
 54 WORK     EQU   X'9'                      WORK REGISTER
 55 HIWD     EQU   X'A'                      HI-ORDER WORD (EVEN REGISTER)
 56 LOWD     EQU   X'B'                      LO-ORDER WORD (ODD REGISTER)
 57 BYTE     EQU   X'C'                      RESULTANT BYTE
 58 *        EQU   X'D'                      SAVE AREA ADDR (NOT MODIFIED)
 59 *        EQU   X'E'                      RETURN ADDR (NOT MODIFIED)
 60 BASE     EQU   X'F'                      BASE ADDR REGISTER
 61 NBFW     EQU   4                         NUMBER OF BYTES/FULL WORD
 62 *
 63 *
 64 *        ****  CODE  ****
 65 *
 66 GBYTES   CSECT ,                         PRIME ENTRY POINT
 67          ENTRY GBYTE                     ALTERNATE ENTRY POINT
 68 *
 69          USING GBYTES,BASE               ENTRY PT ADR ESTB IN R15 BY CALLER
 70          SAVE      (0,12)                SAVE CALLING PROG REGISTERS (MACRO)
 71+* 360N-CL-453 SAVE      CHANGE LEVEL 3-8                                3-8
 72+         STM   0,12,12+4*(0+2-(0+2)/16*16)(13)
 73          LM    NSKP,ITER,4*NBFW(1) PICK UP NSKIP, ITER PARM ADDRESSES
 74          L     NSKP,0(,NSKP)             PICK UP BIT SKIP COUNT
 75          L     ITER,0(,ITER)             PICK UP ITERATION COUNT
 76          B     INIT                      JUMP INTO CODE PROPER
 77 *
 78 GBYTE    DS    0H                        FORCE HALFWD ALLIGNMENT
 79          USING GBYTE,BASE                ALT ENTRY ADDR ESTB IN R15 BY CALLER
 80          SAVE      (0,12)                SAVE CALLING PROG REGISTERS (MACRO)
 81+* 360N-CL-453 SAVE      CHANGE LEVEL 3-8                                3-8
 82+         STM   0,12,12+4*(0+2-(0+2)/16*16)(13)
 83          SR    ITER,ITER                 MAKE ITERATION COUNT ZERO
 84 *
 85 INIT     BALR  BASE,0                    RE-ESTABLISH ADDRESSABILITY
 86          USING *,BASE
 87          LM    PCKD,NBIT,0(1)            PICK UP ARY AND PARAMETER ADDRESSES
 88          L     HIWD,0(,PCKD)             PICK UP FIRST WORD OF PACKED ARY
 89          LH    BITR,=H'32'               AFTER HIWD FETCH FULL 32 BITS REMAIN
 90          L     OFST,0(,OFST)             PICK UP INITIAL OFFSET BIT COUNT
 91          L     NBIT,0(,NBIT)             PICK UP NO. OF BITS/BYTE TO UNPACK
 92          SR    ZERO,ZERO                 ZERO OUT REG 0
 93          LR    WORK,BITR                 MAKE WORK=32
 94          CR    ITER,ZERO                 TEST FOR ZERO ITERATION COUNT
 95          BH    INIT1                     IF NOT, CONTINUE WITH INITIALIZATION
 96 INIT0    CR    OFST,BITR                 ELSE, TEST INITIAL OFFSET IN RANGE
 97          BNL   FAIL                              IF NOT .LT. 32, FAIL
 98          LH    ITER,=H'1'                FORCE EXACTLY 1 ITERATION
 99          B     INIT2
100 INIT1    AR    NSKP,NBIT                 ADD NBIT TO NSKP FOR MULT ACCESS
101 INIT2    LA    PCKD,NBFW(PCKD)           INCR PACKED ARY POINTER
102          L     LOWD,0(,PCKD)             PICK UP SECOND WD OF PACKED ARRAY
103          SR    WORK,NBIT                 COMPLEMENT NBITS (MODULO 32)
104          CR    WORK,ZERO                 TEST NBITS IN RANGE
105          BL    FAIL                      IF NOT..LE.32, FAIL
106          STH   WORK,STORE+4              ELSE USE TO SET UP BYTE SHIFT COUNT
107 *                                          OF SRL (RX) INSTR IN STORE SEQ
108          LR    BITS,OFST                 MAKE BITS TO SHIFT = INITIAL OFFSET
109          LR    WORK,BITR                 MAKE WORK=32, AGAIN
110 *
111 TEST     CR    BITS,BITR                 COMPARE BITS TO SHIFT WITH REMAINING
112          BL    SHIFT                     IF BITS.LT.BITR, PROCEED WITH SHIFT
113          BH    SPLIT                     IF BITS.GT.BITR, ITER OVER SPLIT WDS
114 *        BE    COPY                      ELSE, IF BITS.EQ.BITR, RECOPY HIWD
115 *
116 COPY     L     HIWD,0(,PCKD)             RELOAD HIWD FROM MEM WITH PREV LOWD
117          LA    PCKD,NBFW(PCKD)           INCR PACKED ARY POINTER
118          L     LOWD,0(,PCKD)               AND PICK UP NEXT PACKED LOWD
119          LR    BITR,WORK                   RESET BITR TO 32
120          B     STORE                     PROCEED WITH STORE OPERATIONS
121 *
122 SPLIT    STH   BITR,*+6                  SET UP BITR AS SLDL SHIFT COUNT
123          SLDL  HIWD,*-*                  SHIFT LEFT DOUBLE LOGICAL, HIWD/LOWD
124          SR    BITS,BITR                 DECR BITS TO SHIFT BY SHIFTED BITR
125          B     SPLIT1                    BYPASS FAST FULLWORD SKIPS
126          LR    OFST,BITS                 COPY BITS REMAINING TO SHIFT TO OFST
127 *                                        (DESTROYING INITAL OFFSET PARM)
128          SRA   OFST,5                    DIVIDE BY 32 TO GET WORDS TO SHIFT
129 *        CR    OFST,ZERO                   SIMULTANEOUSLY SETTING COND CODE
130 *                                        IF REMAINING SHIFT .LT. 1 WORD
131          BNH   SPLIT1                    FETCH NEXT SEQUENTIAL LOWD IMMED
132          SLA   OFST,2                    ELSE, MULTIPLY WDS BY 4 TO GET BYTES
133          AR    PCKD,OFST                 INCR PACKED ARRAY POINTER
134          L     HIWD,0(,PCKD)               AND LOAD NEW HIWD
135          SLA   OFST,3                    MULTIPLY BYTES BY 8 TO GET BITS
136          SR    BITS,OFST                   AND DECR BITS REMAINING TO SHIFT
137 SPLIT1   LA    PCKD,NBFW(PCKD)           INCR PACKED ARY PTR
138          L     LOWD,0(,PCKD)               AND PICK UP NEXT PACKED LOWD
139          LR    BITR,WORK                   RESET BITR TO 32
140          B     TEST                      ITERATE AS NECESSARY
141 *
142 SHIFT    STH   BITS,*+6                  SET UP BITS AS SLDL SHIFT COUNT
143          SLDL  HIWD,*-*                  SHIFT LEFT DOUBLE LOGICAL, HIWD/LOWD
144          SR    BITR,BITS                 DECR BITS REMAINING BY BITS SHIFTED

145 *
146 STORE    LR    BYTE,HIWD                 COPY HIWD TO BYTE
147          SRL   BYTE,*-*                  SHIFT RIGHT(WITH HI-ORDER ZERO FILL)
148 *                                          *-* SHIFT COUNT SET UP AT INIT2
149 STORE1   ST    BYTE,0(,UNPK)             FILL BYTE INTO UNPACKED ARY
150          LA    UNPK,NBFW(UNPK)             AND INCR UNPACKED ARY POINTER
151          LR    BITS,NSKP                 RESTORE BITS = (NSKP+NBIT)
152          BCT   ITER,TEST                 DECR ITER COUNT, CONTINUE TILL ZERO
153 *
154 *
155 EXIT     RETURN    (0,12)                RESTORE CALLING PROG REGS (MACRO)
156+* 360N-CL-453 RETURN   CHANGE LEVEL 3-0
157+EXIT     LM    0,12,12+4*(0+2-(0+2)/16*16)(13)
158+         BR    14
159 *
160 FAIL     LH    BYTE,=X'FFFF'             MAKE BYTE A FULL WD OF 1'S
161          ST    BYTE,0(,UNPK)               AND FILL INTO UNPACKED ARRAY
162          B     EXIT
163 *
164          LTORG
165                =H'32'
166                =H'1'
167                =X'FFFF'
168          END
```

Appendix G

Subroutines GBYTE and GBYTES for UNIVAC 1108 Systems

```
*FORTRAN,FL

      SUBROUTINE GBYTE(SOURCE,DEST,OFFSET,SIZE)
C
C  GBYTE ROUTINE FOR UNIVAC 1108
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION SOURCE(2)
      TEMP=0
      I=OFFSET/36+1
      OFF=IABS(OFFSET-(I*36-36))
      N=SIZE
C
C  IS BYTE ALL IN ONE WORD
C
      IF((OFF+N).LE.36) GO TO 10
C
C  NO - MOVE PARTIAL TO TEMP
C
      NN=36-OFF
      N=N-NN
      TEMP=FLD(OFF,NN,SOURCE(I))*2**N
      OFF=0
      I=I+1
   10 DEST=FLD(OFF,N,SOURCE(I))+TEMP
      RETURN
      END


      SUBROUTINE GBYTES(SOURCE,DEST,OFFSET,SIZE,SKIP,LOOP)
C
C  GBYTES ROUTINE FOR UNIVAC 1108 , CALLS COMPANION GBYTE ROUTINE
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION DEST(2)
      OFF=OFFSET
      DO 10 I=1,LOOP
      CALL GBYTE(SOURCE,DEST(I),OFF,SIZE)
   10 OFF=OFF+SIZE+SKIP
      RETURN
      END
```