

BYTES TO BITS -- MORE EFFICIENT MEMORY UTILIZATION

Brian L. Bournival, Information Resources, Inc., Chicago, IL

ABSTRACT

The quantity of data stored and processed by most companies has increased dramatically in the last few years, leading to a demand for more efficient memory utilization. When variables are outputted to an external file using a SAS® format, the system allocates at least one byte of memory for each data point. This paper describes a method that can be used to store Boolean variables in single bits. SAS already provides an informat to read the bit-sized variables.

INTRODUCTION

Review of Bytes and Bits

A byte of memory consists of eight bits, and can hold a value ranging from 0 to 255. Each bit can have a value of 0 or 1. The value stored at a byte is based on its bit settings. For example, the number "35" would be stored as "00100011" in memory. The "35" is derived as follows:

$$\begin{aligned}
 &(0 \times 2^7) \\
 &(0 \times 2^6) \\
 &(1 \times 2^5) \\
 &(0 \times 2^4) \\
 &(0 \times 2^3) \\
 &(0 \times 2^2) \\
 &(1 \times 2^1) \\
 &+(1 \times 2^0)
 \end{aligned}$$

Moving from right to left in the binary number, the value of each bit within a byte increases twofold. The leftmost

bit is referred to as the highest order bit; it can sometimes represent the sign (+ or -) of the stored number. If larger numbers, or numbers with decimal points, need to be stored, more bytes are required. If a character is being stored, the number stored at the byte is equivalent to the ASCII value of the character (or EBCDIC for mainframes).

SAS Memory Allocation

SAS provides a variety of formats that can be used to output variables to an external file. Unfortunately, none of these formats will allow you to allocate less than one byte of memory per variable value. If your variable is Boolean (i.e., it can only have two possible values), this is a waste of memory, because its information could be stored in a single bit.

Reading Bit-Sized Variables

SAS does provide an informat, *BITSw.d*, that enables the system to read single bits from a file. The "w" in the informat is the number of bits to read, and the "d" is the offset from the first bit. For example, in order to read the leftmost bit in a byte that starts at column 4, type:

```
INPUT @4 VARNAME BITS1.0;
```

Substituting *BITS1.3*, the system would read the *fourth* bit from the left. Remember, the second number represents the *offset* from the leftmost bit.

BYTES TO BITS ALGORITHM

Concept

As mentioned previously, a bit can have a value of 1 or 0. Therefore, the "bytes to bits" algorithm is only useful for storing Boolean variables. Since there are eight bits in a byte, up to eight Boolean variables can be stored in each byte. In order to store multiple variables in a single byte, the algorithm generates a value equal to the sum of the component bits. This can best be illustrated through an example.

Let's say we want to store eight variables with the following values:

```
boovar1 = 1
boovar2 = 0
boovar3 = 0
boovar4 = 1
boovar5 = 1
boovar6 = 0
boovar7 = 0
boovar8 = 1
```

These eight values can be used to construct the binary number 10011001 which translates to the base 10 number 153. (Refer to *Review of Bytes and Bits*, if this is unclear.) Therefore, by storing the number 153 at a single byte, we have effectively stored values for all eight variables. The variables can be read bit-by-bit using the *BITSw.d* informat.

Code

In the following illustration of the code, assume there are 26 Boolean variables to be stored in bits.

```
%LET NUMVAR=26;
```

Therefore, four byte-sized variables are required:

```
NUMBYTE=CEIL(&NUMVAR/8);
CALL SYMPUT
  ('NUMBYTE',COMPRESS(PUT(NUMBYTE,BEST)));
```

Create arrays for all of the Boolean variables and the byte-sized variables, so the bytes to bits processing can be accomplished in one loop.

```
ARRAY BOOVAR {*} BOOVAR1-BOOVAR&NUMVAR;
ARRAY BYTVAR {*} BYTVAR1-BYTVAR&NUMBYTE;
```

If there is a possibility that there will be more byte variables than are necessary to store all of the Boolean variables (e.g., construction of a fixed-length bit map), it is a good idea to initialize all byte variables to zero. Otherwise, byte variables that are not processed by the algorithm will have missing values.

```
DO I=1 TO DIM(BYTVAR);
  BYTVAR{I}=0;
END;
```

Eight Boolean variables can be assigned to each byte variable. The first variable is assigned to the highest order bit; subsequent variables are assigned to the next-highest order bit, until the eighth variable is assigned to the lowest order bit.

```
/* J keeps track of the byte variables. */
J=1;
```

```
/* I keeps track of the bit variables. */
DO I=1 TO &NUMVAR;
```

```
/* First 7 of 8 bits go thru 1st loop. */
IF MOD(I,8)^=0 THEN DO;
  IF BOOVAR{I} THEN
    BYTVAR{J}=BYTVAR{J}+(2**(8-MOD(I,8)));
END;
```

```
/* For 8th bit, above expression resolves to
28; lowest order bit should be multiplied by
20. */
ELSE DO;
```

```

IF BOOVAR{I} THEN
  BYTVAR{J}=BYTVAR{J}+(2**0);

/* After lowest order bit is processed, move
to next byte variable. */
  J=J+1;
END;
END;

```

Each of the byte variables should be outputted using the *PIB1.* format because this format treats the highest order bit as part of the value and not a sign.

```

DO I=1 TO DIM(BYTVAR);
  PUT @I BYTVAR{I} PIB1. @;
END;
PUT;

```

CONCLUSION

If you are faced with limited memory resources, or if the files you generate will be read by programs expecting bit-sized input, the algorithm described in this paper could be useful.

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Brian Bournival
 Information Resources, Inc.
 150 N. Clinton St.
 Chicago, IL 60661-1416
 Phone (312) 474-3425
 Fax (312) 474-2981
 Email Brian.Bournival@infores.com