IDENTIFICATION

Product Name: Application Notes

Product Code: DEC-08-NAAA-D

Date Created: July 12,1965

Maintainer: Special Systems Group

APPLICATION NOTE 801

SCALING FOR FIXED-POINT, 2's COMPLEMENT ARITHMETIC

INTRODUCTION

In the programming of arithmetic operations on a fixed-point, 2's complement arithmetic computer, the position of the scale point; that is, the decimal point in a decimal number or the binary point in a binary number, must be kept track of by the programmer. Once numbers have been entered in the computer, there is no hardware or movable machine point to represent the scale points. The scale point exists only in the mind of the programmer, and only by keeping track of its imaginary position is he able to correctly interpret the machine's calculated results.

The fundamental properties of scaled numbers can be simply explained if we imagine a hypothetical decimal machine that is capable of manipulating numbers consisting of a sign and five decimal digits. In this hypothetical computer, as in real computers, the machine acts as if there were a scale point between the sign and the leftmost decimal digit. It is called the machine point. Thus, every number contained in the computer can be thought of as a signed decimal fraction.

Example: $+_{\wedge}12345$

machine point

However, the programmer is free to assign a decimal point at any position in the number. For example, the above number could represent +123.45 if the scale point were thought of as being three places to the right of the machine point. In that case the number would be written $+_{\wedge}12345$ D3, where D3 indicates that the decimal point is three places to the right of the machine point. In other words, D3 is the decimal scale factor. Any scale factor may be chosen without changing the contents of the machine.

Examples: +12345 D2 = +12.345
^
+12345 D4 = +1234.5
^
+12345 D0 = +.12345
^

The scale factor need not be restricted by the size of the machine word. Numbers in our hypothetical computer can be assigned scale factors that exceed five, or the scale factors can even be negative.

Examples: +12345 D7 = +1234500.
^
+12345 D-4 = +.000012345
^

Of course, these are merely programmer's representations; the machine number is always restricted to a sign and five digits.

Addition and Subtraction

In addition and subtraction, the scale factors of the numbers to be combined must be identical. Thus, +42204 D3 added to +23332 D3 gives a sum of +65536 D3 (422.04 + 233.32 = 655.36). This rule has the same basis as in ordinary arithmetic. If the scale factors differ, one number must be shifted until the scale points are aligned.

Examples: +14271 D1 (+1.4271)
^
+38496 D3 (+384.96)
^

The number +14271 D1 is brought into the accumulator and shifted right two decimal places before addition or subtraction. The scale point shifts with the number.

+001.42	(+.00142 D3)
+384.96	(+.38496 D3)
+386.38	(+.38638 D3)

Notice that if the number of the higher scale factor had been shifted left instead, its two most significant digits would have been lost and the resulting sum would have been seriously in error. The shifting of numbers off the left end of the accumulator in this manner is called overflow.

Multiplication

When two numbers are multiplied together, the scale factor of the product is the algebraic sum of the scale factors of the multiplier and multiplicand.

Example: $(+\overset{\wedge}{0}0200\ D3) \times (+\overset{\wedge}{0}6000\ D2) = +\overset{\wedge}{0}0012\ D5$, or 12.

Normally the most significant part of the product is in the AC and the least significant part is in another register. Thus, the product in the above example would appear in the computer with the machine point between the sign and leftmost digit in the AC. The machine point for the least significant portion of the product is ignored, since this involves double-precision arithmetic.

$$+\overset{\wedge}{0}0012\ +\overset{\wedge}{0}0000\ D5$$

It is important to remember that the two decimal numbers, $+\overset{\wedge}{0}0200$ and $+\overset{\wedge}{0}6000$, when multiplied in the computer will result in the machine product of $+\overset{\wedge}{0}0012\ +\overset{\wedge}{0}0000$ regardless of the positions of the scale points in the multiplier and multiplicand. The scale points must be kept track of by the programmer. Thus fractions, as well as integers, can be multiplied in exactly the same way.

Examples: $+\overset{\wedge}{2}0000\ D5 \times +\overset{\wedge}{0}0060\ D3 = +\overset{\wedge}{0}0012\ +00000\ D8$ (+20,000 x +.6 = +12,000)

$+\overset{\wedge}{0}0200\ D0 \times +\overset{\wedge}{0}6000\ D0 = +\overset{\wedge}{0}0012\ +00000\ D0$ (+.002 x +.06 = +.00012)

$+\overset{\wedge}{0}0200\ D{-}2 \times +\overset{\wedge}{6}0000\ D{-}4 = +\overset{\wedge}{0}0120\ +00000\ D{-}6$ (+.00002 x +.00006 = +.0000000012)

Division

The remarks above for multiplication apply directly to division, with two exceptions. First, the scale point of the result is the algebraic difference of the scale points of the operands.

Second, the scale point of the divisor must be smaller than the scale point of the dividend; that is, the scale point of the quotient must be positive. Again, the contents of the registers in our decimal computer will look the same no matter where scale points of the numbers are located.

SCALING ON A BINARY COMPUTER

The fundamental properties of scaled numbers in a computer as outlined above can now be applied to the binary and octal numbering systems as used in a 2's complement, fixed-point, binary computer. The decimal scale factor becomes the binary scale factor and is indicated by a B in front of the scale factor. The machine point is still between the sign and the leftmost digit, but in this case the sign is a binary digit. In a 12-bit computer such as PDP-8, then, the leftmost bit is the sign bit and there are eleven bits for the number, with the machine point between the first and second bits.

Because the number system used by the machine is now different from that customarily used by the programmer, conversion becomes one of our new considerations. The programmer may be dealing with decimal or octal numbers, but because the machine is binary, the scale factors must be determined from the binary equivalents. As will be explained below, a scaling analysis is performed on each problem so that the binary scale factors chosen result in the most efficient use of the 12-bit word. Having selected the appropriate scale factor for a given number, it is expressed in decimal or octal form followed by the binary scale factor. For example, the combination 975 B10 means that the decimal number 975, when converted to binary form, has a binary point ten places to the right of the machine point.

The decimal number 975 B10 when converted to binary would appear in the machine as:

0 11 110 011 11 0
^ ^
machine point binary point

Grouping these bits into threes, it is more convenient to write this number in its octal form:

3636 B10

Notice that in this octal form, we cannot indicate the point which separates the integral from the fractional part, because it comes within one of the octal digits. Also, the sign bit, bit 0, becomes part of the leading digit.

Negative numbers can be written either one of two ways. For example, consider the octal number:

$$-3.2\ B6$$

As a positive octal number, 3.2 B6 would be stored in the computer as:

$$000\ 001\ 101\ 000$$
$$\uparrow$$
binary point

As a negative number in 2's complement, it would be stored:

$$\begin{array}{r} 111\ 110\ 010\ 111 \\ \uparrow \\ +1 \\ \hline 111\ 110\ 011\ 000 \\ \uparrow \end{array}$$
binary point

In the octal form, it would be written:

$$7630\ B6$$

Again we cannot separate the integral from the fractional part, and the sign is incorporated into the leading octal digit.

The summary of the above rules of binary scaling is given in Appendix 1.

OVERFLOW

In addition to shifting digits off the left end of the accumulator, overflow can also occur in arithmetic operations. Suppose we are working with signed quantities and we add the numbers:

Decimal Value	Binary Representation	Octal Equivalent
18 B5	$\overset{\uparrow}{0}10\ 010\ 000\ 000$	2200 B5
5 B5	$\overset{\uparrow}{0}00\ 101\ 000\ 000$	500 B5
23 B5	$\overset{\uparrow}{0}10\ 111\ 000\ 000$	2700 B5

Notice that there was no carry to the left of the first machine position (i.e., into the sign bit). However, if we try to add the numbers:

Decimal Value	Binary Representation	Octal Equivalent
28 B5	011 100 000 000	3400 B5
5 B5	000 101 000 000	500 B5
33 B5	100 001 000 000	4100 B5

The result as given in the machine would be erroneous because the magnitude portion of the AC is not large enough to hold the sum. This situation is described as overflow.

If overflow occurs in division, the link is set to 1, no division takes place, and control returns to the main program.

Overflow occurs in the PDP-8 when:

1. In addition, the sign of the addend and augend are the same and the sign of the sum is different and/or the link is set.

2. In division, the magnitude of the divisor is less than that of the dividend when the scale factors are the same (i.e., when the quotient $\geq$ 1.0 B0).

3. A digit is shifted off the left end of the accumulator.

Overflow is something which must be avoided in all normal circumstances. To accomplish this, the programmer must have some knowledge of the magnitude of the numbers with which he is working and, accordingly, must locate each number at such a scale that overflow cannot occur even in the "worst case."

In this connection, the concept of "minimum binary scale" is helpful. At a binary scale of 5, the largest positive integer that can be contained is:

011 111 000 000
^
binary point

which in decimal is 31 (i.e., $2^5 - 1$).

The largest positive integers which can be contained at other binary scales are tabulated in Appendix 2. If, for example, we have the number 75 to place in the computer, the table in Appendix 2 indicates that it can be contained at a binary scale of 7 or higher. A binary scale of 6 or lower would not be sufficient to hold a number of that magnitude.

If a binary scale factor greater than 7 were used, the number would be shifted farther to the right than necessary resulting in underflow. The number of significant figures that can be carried in the fractional part is thereby reduced. If the number 75 were carried as 75 B7, there is room in the machine word for fractional results since four binary bits can follow the binary point; if it were carried 75 B11, there is no provision for a fractional part in single-precision arithmetic.

The programmer may not always be successful in his attempts to arrange numbers so that overflow will not occur. If overflow does occur, the PDP-8 does not halt but an indication of some type is given. The type of indication depends upon the operation which produced the overflow. If a programmer suspects that overflow may occur as a result of an addition or division, he should follow such an operation by a program sequence that would correct the error or at least indicate that such an overflow took place.

The proper location of the binary point and the avoidance of overflow, at best, takes some effort on the part of the programmer.

PROGRAMMING TECHNIQUES FOR SCALING

General

A complete set of shift subroutines in both single and double precision is available from the PDP-8 Library (Digital-8-8-U-Sym) for use in scaling numbers both before and after arithmetic operations. When using the routines, it is important to keep in mind that a left shift of one position decreases the binary scale factor by one. Similarly a right shift of one position increases the binary scale factor by one.

One technique used in scaling is to express numbers in a symbolic form that would clearly imply the position of the binary point. The general form is:

$$X2^{-q} = X'$$

where: X is the absolute value of the number.

2^q is the factor such that q is the smallest integer that makes 2^q greater than the maximum value of X.

q corresponds to the minimum binary scale factor which was previously discussed.

X' is the scaled form of X (i.e., X is X' with the binary point q places to the right of the machine point).

A scaling analysis should be performed on each problem to insure maximum accuracy (i.e., the most efficient use of the binary word so that there are no leading insignificant bits). At the same time, the programmer must insure that there will be no loss of the most significant bits by overflow at any step in the calculation. These are the two bounds within which the programmer must keep the numbers as they are stored and manipulated in the machine.

Analysis

In the programming of any given problem or equation, there are three steps prior to the actual coding which should be taken to insure maximum accuracy and to prevent error due to overflow.

Step 1: Determine the limits of the values of all numbers to be used in the problem (maximum and minimum).

Step 2: Determine the scale factors and set up the relationships between the true numbers and the scaled fractions.

Step 3: Substitute the scaled quantities into the original equation and cancel where possible. The scale factors that do not cancel specify the number

of required shift operations. If the scale factor of a term is negative, the number must be shifted right before manipulation is performed. If the scale factor is positive, the number must be shifted left if it is to be stored at minimum binary scale.

Addition Scaling

As emphasized before, quantities to be added (or subtracted) must have the same scale factors. However, in order to prevent an overflow in the summing process, it is not enough to scale the final sum according to its limit. Generally the program must be scaled by the largest limit which applies to any element in the sum or partial sum generated during the summing process.

Example 1

Program the operation specified by:

$$A = \sum_{i=1}^{n} a_i$$

where $a_i < K$ for $i = 1, 2, 3, \ldots n$. The maximum value of A is $\leq K \cdot n$, that is, the maximum value of the sum is obtained by multiplying the upper limit of any element in the list to be summed by the number of elements in the list.

1. Statement

Solve the above problem for $n = 10$ and $K = 100.0$.

2. Analysis

Step 1: $a_i \leq 100.0$ for $i = 1, 2, 3, \ldots 10$
Therefore, $A \leq K \cdot n \; 100.0 \cdot 10 = 1000$

Step 2: $A = 2^{10} \cdot A'$
$a_i = 2^7 \cdot a_i$

Step 3: $2^{10} A' = 2^7 a_1' + 2^7 a_2' + \ldots\ldots + 2^7 a_{10}'$

$$A' = 2^{-3} a_1' + 2^{-3} a_2' \ldots\ldots + 2^{-3} a_{10}'$$

3. Machine Instruction Coding

Assume that the ten values of the numbers are stored in consecutive locations starting at location A1 as a_i B7 and that the sum is stored in A.

```
            .
            .
            .
ADDUP,      CLA                 /INITIALIZE
            DCA     A
            TAD     M12
            DCA     COUNTR
            TAD     ADR1
            DCA     POINTR
LOOP,       TAD     M3          /SUMMATION LOOP
            JMS     SPSR        /ENTER SHIFT RIGHT SUBROUTINE
            POINTR
            TAD     A
            DCA     A
            ISZ     POINTR
            ISZ     COUNTR
            JMP     LOOP
            .                   /EXIT
            .
            .
POINTR,     0                   /CONSTANTS
M3,         0-3
COUNTR,     0
M12,        0-12
ADR1,       A1
A,          0
```

Multiplication Scaling

When multiplication is performed in digital computers, note that the product of two "n" bit numbers is one "2n" bit number. Usually the high-order portion is left in the AC and the low-order portion is stored in a specified register. The fundamental rule, again, is:

Scale factor of multiplier + scale factor of multiplicand = scale factor of product.

1. Statement

Program the multiplication operation:

$$x = a.b$$

2. Analysis

Step 1: $a<400.0$
$b<1000.0$
Therefore $x<400,000.$

Step 2: $a = 2^{9} a'$
$b = 2^{10} b'$
$x = 2^{17} x'$

Step 3: $2^{17} x' = 2^{9} a' \cdot 2^{10} b'$
$x' = 2^{2} a' \cdot b'$

3. Machine Instruction Coding

Assume that the values of a and b are stored in locations A and B. Assume that they are scaled B9 and B10, respectively.

```
            .
            .
            .
SETMUL:     CLA
            TAD     A
            JMS     MULT
B,          0
            DCA     SVA         /MOVE PRODUCT TO TEMP STORE
            TAD     MP1
            DCA     SVA +1
            TAD     M2
            JMS     DPSL        /SHIFT PRODUCT LEFT 2 PLACES
            SVA
```

```
          .
          .
SVA,      0
          0
M2,       0-2
```

After the multiplication (22-bit signed product), the shift brings two more significant bits into the high-order portion of the product. Knowing the maximum value of y more definitely (i.e., if a and b could never be maximum at the same time) would allow for even more accuracy. In this example, the limit of y was not known so it was assumed to be 400,000 as calculated in Step 1 of the analysis.

Division Scaling

When division is performed in digital computers, the dividend is a "2n" bit word and the divisor is an "n" bit word.

Remember that in division the divisor must be greater than the dividend for division to occur without overflow. Therefore, the programmer should scale the values so that division will occur with maximum dividend and minimum divisor. For example, if both dividend and divisor are stored at minimum binary scale, the dividend should be shifted one position to the right by a double-shift subroutine before division to insure that overflow does not take place.

1. Statement

Program the division operation:

$$y = \frac{m}{n}$$

2. Analysis

Step 1: $m<24,000$

$60<n<1,000$

Therefore, $24<y<400$

Step 2: $m = 2^{15} m'$

$n = 2^{10} n'$

$y = 2^{6} y'$

Step 3: $2^{6} y' = \frac{2^{15} m'}{2^{10} n'} = 2^{5} \frac{m'}{n'}$

$y' = 2^{-1} \frac{m'}{n'}$

In this example, the 2^{-1} implies that the quotient would have one significant bit to the left of the machine point (i.e., in the sign bit). Thus, the division would result in overflow. This problem could be averted by shifting the dividend right one position, as previously mentioned, before division takes place.

3. Machine Instructions

Assume that the single-precision division subroutine (Digital 8-12-F) is used. The dividend is stored in locations num1 and num1 + 1 at a scale factor of B15 and the divisor is stored in num2 at a scale factor of B10.

```
          .
          .
          .
SETDIV,   CLA     CMA          /LOAD -1 INTO AC
          JMS     DPSR         /SHIFT RIGHT SUBROUTINE
          NUM1
          DCA     SVA          /SAVE  MSB
          TAD     LSH          /MOVE  LSB
          DCA     CALDIV + 1
          TAD     SVA
CALDIV,   JMS     DIV          /DIVIDE B16/B10 = B6
          0
NUM2,     0
          DCA     ANS          /SAVE QUOTIENT
          .
          .
          .
```

APPENDIX 1

RULES OF BINARY SCALING

ADDITION

The binary scale factor of the addend, augend, and sum are alike.

23.9 B8 + 169.7 B8 = 146.2 B8

SUBTRACTION

The binary scale factor of the minuend, subtrahend, and difference are alike.

107.8 B7 - 23.2 B7 = 84.6 B7

MULTIPLICATION

The binary scale factor of the product is the sum of the binary scale factor of the multiplier and multiplicand.

12.2 B6 x 3 B7 = 36.6 B13

24.9 B5 x 135.5 B8 = 3373.95 B13

(minimum binary scale)

DIVISION

The binary scale factor of the quotient is the binary scale factor of the dividend minus the binary scale factor of the divisor.

88 B15 ÷ 11 B5 = 8 B10

APPENDIX 2

MINIMUM BINARY SCALE

Binary Scale	Maximum Decimal Capacity ($2^n - 1$)
1	1
2	3
3	7
4	15
5	31
6	63
7	127
8	255
9	511
10	1 023
11	2 047
12	4 095
13	8 191
14	16 383
15	32 767
16	65 535
17	131 071
18	262 143
19	524 287
20	1 048 575
21	2 097 151
22	4 194 303
23	8 388 607

APPLICATION NOTE 802

The question of matrix inversion comes up occasionally concerning the PDP-8. There is no general answer to the matrix-inversion problem. The approach depends upon the "behavior" of the given matrix. Basically, the problem is:

Given a matrix A, find a matrix B, such that
AB = I where I is the unit matrix.

There are three basic approaches. All numbers below are decimal, all operations are floating-point without EAE.

GAUSS - JORDAN METHOD

Time $\approx$ (2.5) (1.46 mils) (M^3) where the matrix is M x M

Storage $\approx 3M^2$ + 630 + about 550
(This does not include input/output which would require about 450 locations.)

For a 10 x 10 (well behaved) matrix:

Time $\approx$ 3.66 seconds

Storage $\approx$ 1480 words + 450 for I/O

For a 20 x 20 (well behaved) matrix:

Time $\approx$ 29.28 seconds

Storage $\approx$ 2380 words + 450 for I/O

RANK ANNIHILATION

Time $\approx 5M^3$ (1.96 mils) where the matrix is M x M

Storage $\approx 6M^2$ + 12M + 630 + about 400 words
(This does not include I/O which would require about 450 locations.)

For a 10 x 10 well behaved nonsymmetric matrix:

Time $\approx$ 9.80 seconds

Storage $\approx$ 1750 + 450 for I/O

For a 20 x 20 well behaved nonsymetric matrix:

Time $\approx$ 78.4 seconds

Storage $\approx$ 3670 + 450 for I/O

GAUSS - SEIDEL (ITERATIVE METHOD)

Time $\approx M^2$ (.88 mils) per iteration (M x M matrix)
(It would be impossible to estimate the number of iterations required. The method may not work in some cases.)

Storage $\approx 3M^2 + 6M + 630$ + about 200 (not including I/O)

For a 10 x 10 matrix:

Time $\approx$88 mils / iteration

Storage $\approx$1190 + 450 for I/O

For 20 x 20 matrix:

Time $\approx$352 mils / iteration

Storage $\approx$2150 + 450 for I/O

Generally, a matrix is well behaved if the elements along the major diagonal dominate are greater than the other elements of the matrix. As nondiagonal elements become larger than the diagonal elements, the matrix becomes ill behaved and it becomes increasingly difficult to invert.

APPLICATION NOTE 804

THROUGHPUT TO IBM-COMPATIBLE MAGNETIC TAPE

A common data-acquistion situation is the real-time conversion of an analog signal(s) to digital form together with the recording of the resulting digital information on magnetic tape in IBM-compatible format (IBM Binary).

A typical system consists of a 138B Converter, PDP-8, 57A and 570 Tape Transport. The use of the 57A permits the controlling program to load one core buffer while the 57A writes (via the data break) the data in a second core buffer on tape and produces the interrecord gap. The function of the buffers is then reversed by the program.

Within a record, the character density must conform to IBM standards; 200, 556, or 800 characters to the inch. IBM specifications govern the longitudinal tolerance as each character may be recorded. Each interrecord gap must be regardless of density (within allowable tolerance) 3/4 inch to conform to IBM standards. These comments apply for tape that will be processed on IBM Magnetic Tape Units IBM 729 II, IV, V, and VI.

In a given file on tape (disregarding the end-of-file gap and character), the effective density of recorded information is a function of the relationship between record length as compared to gap length. For example a file consisting of a single long record with no record gaps would consist entirely of data or would be 100% efficient. On the other hand, a file consisting of many 3/4-inch records would be only 50% efficient since half of the available tape would be used for the 3/4-inch interrecord gaps required by each record.

While the 57A is writing the contents of the one buffer on tape and then writing the blank interrecord gap, the second buffer is being filled by information coming from the A-to-D converter. Since the conversion rate must be constant, this process continues both during the emptying of the first buffer and during gapping. There is, therefore, a difference in the rate at which data enters one buffer from the A-to-D converter and the rate at which data as distinguished from gap is written on tape from the second buffer.

Consider a 570 Transport with a speed of 75 ips recording at 200 cpi. The number of characters that may be recorded per second is equal to:

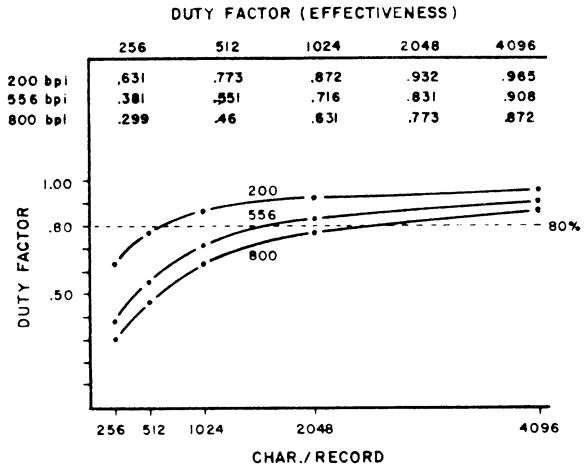
$$75 \times 200 = 15000 \text{ cps}$$

Suppose, though, that the situation discussed above (3/4-inch records separated by 3/4-inch gaps) existed. The actual conversion rate (assuming a 12-bit conversion) would be:

$$15{,}000 \times 1/2 \times 1/2 = 3750 \text{ conversions per second}$$

The second factor of 1/2 arises because only half of the tape is available for actual data (0.50 duty factor) due to the interrecord gaps.

Figure 1 illustrates the duty factor as a function of characters per record for all three standard IBM densities. A 3/4-inch interrecord gap is assumed.



Note that these rates could never be realized in practice with a double-buffer type of throughput since the assumption of the necessity of a double buffer implies that at least two records-an ideal case and therefore one interrecord gap-will be written.

Figure 2 lists the throughput rates which could be achieved if no interrecord gaps were required. Next, the duty factor is used to calculate actual (or effective) character rates as a function of characters per record and this information is plotted. Consider the top curve of Figure 3. In this case the nominal (or ideal) character rate is 90,000 cps. The vertical bars at the two "ends" of the curve illustrate how far below the nominal character rate the actual character rate is at any point. Note that for small records the actual character rate falls off very rapidly.

Writing Rates (cps)

	200 bpi	556 bpi	800 bpi
45 ips	9,000	25,000	36,000
75 ips	15,000	41,700	60,000
112.5 ips	22,500	62,500	90,000

Effect of Record Length

		256	512	1024	2048	4096
45 ips	200 bpi	5,670	6,960	7,850	8,380	8,680
	556 bpi	9,520	13,800	17,810	20,800	22,700
	800 bpi	10,770	16,600	22,700	27,850	31,400
75 ips	200 bpi	9,450	11,600	13,100	13,990	14,490
	556 bpi	15,900	23,000	29,530	34,670	37,870
	800 bpi	17,950	27,640	37,800	46,400	52,300
112.5 ips	200 bpi	14,180	17,400	19,620	20,970	21,730
	556 bpi	23,800	34,500	44,500	52,000	56,800
	800 bpi	26,900	41,500	56,700	69,600	78,400

One interesting aspect of Figure 3 is that by proper choice of record length (or core buffer size, see below), tape speed, and density, a desired ADC conversion rate may be selected. Note the horizontal line at 40,000 cps. Assuming 12-bit conversions, the conversion rate here would be 20,000 conversions per second and this could be achieved by using a double buffer, each one of which was capable of holding the number of characters indicated on the lower scale below the intersection of the horizontal line (at 40,000 cps) with the respective equipment curves.

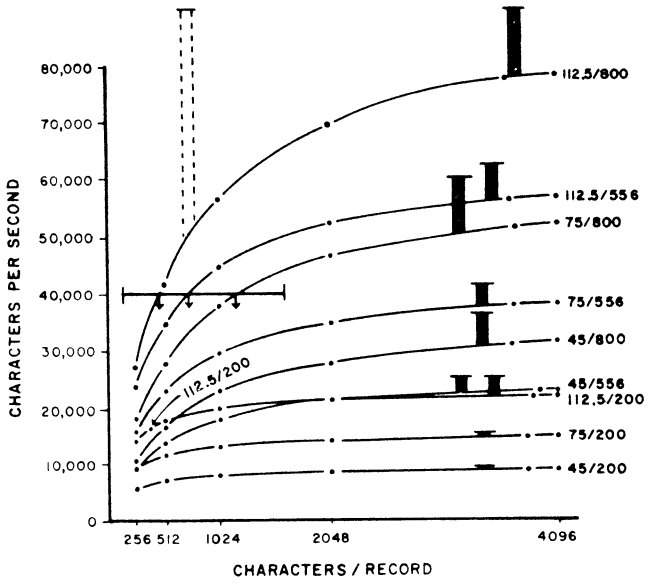
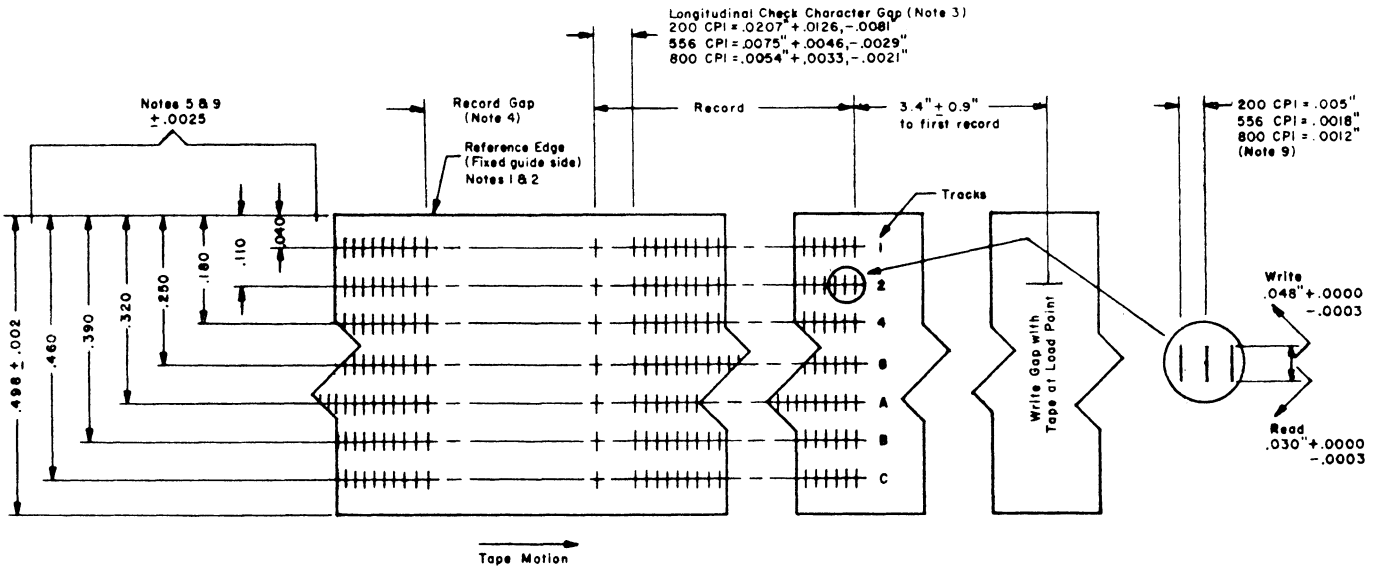


Figure 4 defines certain details of IBM-compatible tapes. Note that the longitudinal check character of the end of a record is separated from the other characters in a record by a gap that is about four times as large as the intercharacter spacing within a record (regardless of density) and that the record gap proper starts following the check character. (No longitudinal check bit is written if the longitudinal count in the associated track is even).

This fact is of consequence only if records composed of an extremely few characters are of concern and would effect the numerical data used to construct Figure 2 in the immediate vicinity (i.e., just to the right) of the vertical scale.

Figure 5 illustrates the time available between analog-to-digital conversions as a function of tape speed, density and buffer size. Figure 5 may be used to determine how many machine cycles are available for programming in throughput situations of this nature.



NOTES:

1. Tape is shown with oxide side up, Read/Write head on same side as oxide.

2. Tape shown representing 1 bit in all tracks, NRZI recording; 1 bit produced by reversal of flux polarity, tape fully saturated in each direction.

3. Variation permitted in the location of the Check Character assuming nominal values for tape speed and all oscillator timings in the Tape Control. No longitudinal check bit is written if longitudinal count in the track is even.

4. Mylar Tape: 3/4", +5/32", -1/16". Acetate Tape. 3/4", +5/32", -1/8". Zero Backward creep. Forward creep less than 0.2" per cycle.

5. Dimensions of tape measured at 50% relative humidity and 70°F. Tape thickness (Mylar or IBM HD) is 0.0022", +.0003", -.0004".

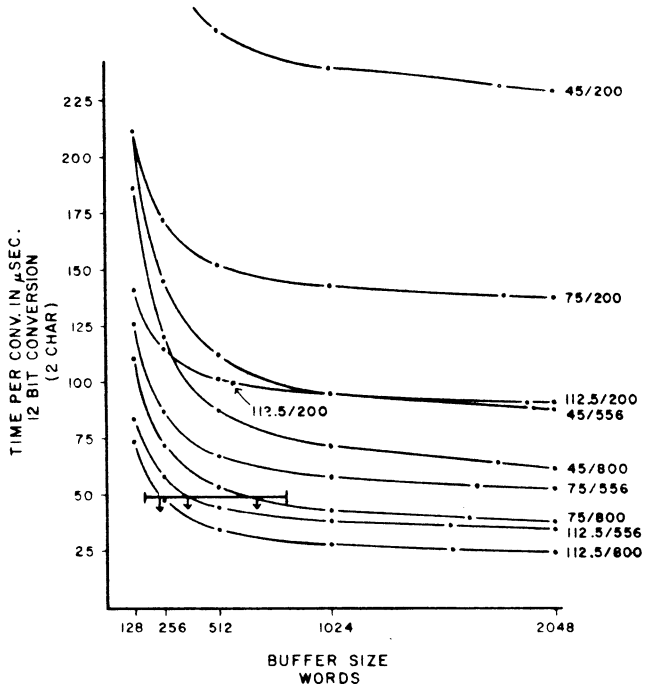
6. To insure complete interchangeability, skew of each tape unit is adjusted to 0.25 μsec or less at the read bus of the tape unit when reading-while-writing continuous 1 bits. Maximum skew for any reel of tape, read by any tape unit connected to any tape control, must be equal to or less than the read character gate for the bit density and tape speed at which the tape was written.

	Time from First Bit	
7. Write Skew Gate, ±5%	Rise	Fall
729 II or V, 556 cpi	6.3 μsec	16.1 μsec
729 II or V, 200 cpi	16.9 μsec	44.0 μsec
729 IV or VI, 556 cpi	4.3 μsec	10.8 μsec
729 IV or VI, 200 cpi	11.4 μsec	29.5 μsec
729 V, 800 cpi	6.3 μsec	10.4 μsec
729 VI, 800 cpi	4.0 μsec	6.8 μsec

When reading, while writing coded information, all bits within a character must be received before the rise of the write skew gate.

8. Read or Write Character Gate, ±5%	
729 II or IV, 556 cpi	10.5 μsec
729 II or V, 200 cpi	29.2 μsec
729 IV or VI, 556 cpi	7.5 μsec
729 IV or VI, 200 cpi	21.0 μsec
729 V, 800 cpi	7.9 μsec
729 VI, 800 cpi	5.4 μsec

9. Time Between Characters: Writing—shall not be less than fall of the skew gate timing plus 1 μsec, including variations due to tape speed, skew and bit configuration. Reading—shall not be less than read character gate timing plus 1 μsec, including variations due to tape speed, skew, and bit configuration.



It must be emphasized that the data graphed as continuous lines in Figures 1, 3, and 5 actually consists for each curve of a series of discrete points, which are shown as continuous lines for convenience only. For example there can never be a record consisting of 356.135 characters, and no point associated with this number of characters is actually present or intended to be present in the curves of Figure 1.