Analysis of Delays in Converting from a Redundant Representation.

by

Colin D. Walter

Computation Department, U.M.I.S.T.,
PO Box 88, Sackville Street, Manchester M60 1QD, U.K.

cdw@sna.co.umist.ac.uk
http://www.co.umist.ac.uk/

Key Words: *Computer arithmetic, on-line algorithms, most significant digit first algorithms, binary conversion, redundant number representations, real time systems.*

Abstract

The use of a redundant system allows many arithmetic operations to process digits sequentially most significant digit first. Final conversion back to a standard binary representation can require time to propagate any carries. We analyse and report on the delays encountered when this is done by an on-line algorithm, giving a good upper bound on the expected delay. The delay is approximately $log_r k$ for the kth digit in a representation with base r.

1. Introduction.

High speed arithmetic employs several standard techniques to avoid the delay caused by carries which may propagate the whole length of a number. These include the use of a redundant number system to enable digit operations to be performed in parallel, and reorganising algorithms to operate digit serially from the least significant digit. However, for many applications, such as image compression and real time systems [1], *on-line* methods [2] are preferable. Such algorithms consume and generate data digit serially but most significant digit first.

On-line output is generally in redundant form since carries cannot be propagated up. However, on-line input may have to be in a standard non-redundant form. For example, if redundancy is given by including negative digits, then exponentiating by an on-line exponent which contains negative bits in a ring which contains no inverse (or for which the inverse is expensive to compute) is a problem. Another example might be supplying input to pre-existing components which expected non-redundant input. Hence there is a need for an on-line converter from redundant to standard representations. On-line algorithms tend to produce and consume data at a constant speed, say one digit per clock cycle. Such a converter cannot do this without occasional delays as it awaits sufficient information to propagate the carries. Generally, such delays may accumulate because subsequent on-line processes cannot catch up; they can only process a single digit at once or wait if an input digit is not available. This article looks at the average delay expected, as this may be of interest in real time systems.

Ercegovac and Lang [4] have already shown how to convert back to non-redundant form in such cases. Although their method converts as fast as possible, they do not attempt to measure the average expected delay. The complexity of the delay turns out to be logarithmic, and thus coincides with the complexity of carry propagation and of performing a parallel digit conversion (see [5]). The results are applicable also when on-line output is to be rounded (see [3]).

2. Examples.

We begin with some illustrative decimal examples which use the usual digits together with a digit X of value 10 and, for the first, also a digit $\overline{1}$ of value –1. The redundant numbers 0.1999...9X... and 0.2000...0$\overline{1}$... may convert to 0.2000... and 0.1999... respectively. In both cases we can allow for any eventuality by maintaining a lower bound L = 0.1999... and an upper bound U = 0.2000... which are in standard non-redundant form and are updated as each input digit in received. Eventually an input digit will determine which of these to choose. The common prefix of such bounds will belong to the standard representation of the input and so can be sent to output. If, as here, the largest carry up from the unread part of the redundant input is ±1, then only sequences of 9s or 0s in the converted output could lead to lead to carry propagation affecting more than one digit..

With the same notation, consider the input 0.29993954... Only when the '3' is received is it clear that the '2' is correct and can be output. A delay of three cycles builds up as a result of the three '9's. This holds up the output of all subsequent digits. However, the next '9' does not further delay the output of its preceding '3' because, by the time the '3' can be output, the '5' has already been read. Thus the delay of a given output digit depends not on the total number of preceding digits '9', but on the length of the maximal substring of consecutive '9's which precede it.

Clearly the relative distribution of digits affects the average delay. If only digits '0' to '9' occur, then the number is already in standard form and the delay is clearly 0. If the digit set includes X but not $\overline{1}$, then a process may require to generate an ambiguous sequence of '9's which can be upgraded by 'X' rather than a sequence of '0's which it could not decrease. Thus '9's would be more frequent, increasing the average delay. Complexity also arises because consecutive digits are not usually independent. This is readily verified for a carry save adder in base 10, if the carry and save are added to give a digit between 0 and X: a digit position with output '9' is unlikely to generate a non-zero carry, so that it is unlikely to be followed by an X.

3. Determination of the Output Digits.

We will consider converting the redundant representation $A = \Sigma_{i\geq 1} a_i r^{-i}$ of a fractional real number into a standard, non-redundant representation $A' = \Sigma_{i\geq 1} a'_i r^{-i}$. Here r is the *radix* of the representations. We assume the redundant digits a_i are in the range $0..2(r-1)$ whereas those of A' are to be in the standard range $0..r-1$. In the representation A we assume consecutive digits occur independently and $r-1$ occurs with probability $1/r$. (Since no other digit has the same residue *mod* r as $r-1$, it is reasonable to assume it occurs with probability $1/r$.) We will use $B[k]$ to denote a number representation of k digits, i.e. one of the form $\Sigma_{1\leq i\leq k} b_i r^{-i}$. Then $B[k] = C[k]$ will mean the values of $B[k]$ and $C[k]$ are equal, whereas $B[k] \equiv C[k]$ will mean their representations are identical.

The "on-the-fly" conversion method in [4] defines two non-redundant k-digit numbers $L[k]$ and $U[k]$ such that either $L[k] \equiv A'[k]$ or $U[k] \equiv A'[k]$. Thus, any common prefix of $L[k]$ and $U[k]$ is part of the output we desire. The properties $A[k] \leq A < A[k]+2r^{-k}$, $A'[k] \leq A' < A'[k]+r^{-k}$, $A = A'$ and $A'[k] = A[k]+nr^{-k}$ for some integer n, entail that $n = 0$ or 1. Hence $L[k] = A[k]$ and $U[k] = A[k]+r^{-k}$. These values can be constructed incrementally. By equating values, if $a_{k+1} < r$ then $L[k+1] \equiv L[k]+a_{k+1}r^{-k-1}$, whereas if $a_{k+1} \geq r$ then $L[k+1] \equiv U[k]+(a_{k+1}-r)r^{-k-1}$. Similarly, $U[k+1] \equiv L[k]+(a_{k+1}+1)r^{-k-1}$ if $a_{k+1} < r-1$ whereas $U[k+1] \equiv U[k]+(a_{k+1}+1-r)r^{-k-1}$ if $a_{k+1} \geq r-1$. Thus $L[k+1]$ and $U[k+1]$ agree on their first k digits when $a_{k+1} \neq r-1$, as we expected from the decimal examples above. (We have actually chosen the largest range of redundant digits for which this is the case.) So, for $a_{k+1} \neq r-1$ the delay to outputting the kth digit is the same as for the $k-1$th digit. Moreover, $L[k+t]$ and $U[k+t]$ will agree on only their first k digits precisely when $a_{k+1} \neq r-1$ but $a_{k+i} = r-1$ for $1 < i \leq t$. This happens with probability $(1-r^{-1})r^{1-t}$. Then, if $a_{k+t+1} \neq r-1$, the next t digits are all determined together.

4. The Average Delay – An Upper Bound.

When output digits are determined they are added to an output buffer, from which any available digits are output at the rate of one per cycle, the same rate as digits are input. If the kth output digit is determined by the k+1st input digit and the output buffer is empty, then it will be output immediately, i.e. one cycle after the corresponding input digit. This is the minimal conversion time for a digit, and so its delay will be defined to be 0. So, if after queueing in the output buffer the kth digit is output on the $(k+1+d)$th cycle, it is defined to have a delay of d. We wish to determine the average value of d as a function of k.

An initial, crude upper bound on the average delay is easy to obtain. On average, an input of k digits will contain $r^{-1}k$ digits $r-1$, each of which contributes up to 1 to the delay. However, there may be a further delay if the k+1st digit is $r-1$. The average number of consecutive digits $r-1$ starting at the k+1st is $\sum_{i=0}^{\infty} ir^{-i}\,(1-r^{-1}) = (r-1)^{-1}$. (The sum is over sequences of length i terminated by a digit other than $r-1$.) So, overall, the average delay is at most $r^{-1}k + (r-1)^{-1}$. Numerical work soon shows that this is a rather poor estimate.

5. An Improved Upper Bound.

Each occurrence of an input digit $r-1$ leads to an extra delay of *up to* 1 time unit. The delay may actually be less, as noted in one of the previous examples. The delay is, in fact, determined by the *maximum* string of input digits $r-1$ from the beginning up to the next one different from $r-1$. To see this, it suffices to establish that if the input digit is not $r-1$ then the output buffer contains d digits where d is the maximum length of any input sequence of digits $r-1$. The output buffer length does not change if another digit different from $r-1$ is received. Otherwise, if d' consecutive digits $r-1$ occur and $d' \le d$ then the buffer temporarily decreases in size by d', but is eventually restored to size d. However, if a new maximal sequence of d' consecutive digits $r-1$ occurs, so $d' > d$, then the buffer empties and is restored with d' digits, all of which then experience a delay of d'.

The main problem with the estimate of the average delay in the last section is that there are probably many submaximal sequences of consecutive digits $r-1$ for each of which 1 was unnecessarily added to the estimate. One way to avoid most of this unwanted contribution is to force a minimum delay of t, say, on all output. This makes an allowance once for all the subsequences of length t or less, rather than for every occurrence. In effect the buffer starts with t elements.

So, suppose there is a built-in delay of t between input and output. The kth digit will be processed at or before a time which is $k+t$ plus the delay caused by digits $r-1$ lying in subsequences of length greater than t. A subsequence of length $l > t$ will add at most $l-t$ to the delay. So a single digit in such a sequence will add on average at most $(l-t)l^{-1}$ to the delay. The formula for the average delay therefore needs to be re-expressed with a summation over sequences of length l for each digit position.

Consider the probability that the κth input digit lies in a sequence of l consecutive $(r-1)$s, of which exactly p occur at or before position κ. We assume $p \le \kappa$ and $p \le l$ since otherwise the parameters are impossible. For $l = 0$ and $p = 0$ the probability is just $1-r^{-1}$. For $l > 0$ and fixed $p < \kappa$ the probability is $(1-r^{-1})^2 r^{-l}$ since both ends of the subsequence must have a digit other than $r-1$. Otherwise, i.e. for $l > 0$ and $p = \kappa$, all the first l digits are $r-1$, but the l+1st is not, so the probability is $(1-r^{-1})r^{-l}$. Summing over the possible values of p, namely $0 \le p \le \min\{\kappa,l\}$, we find that the probability of the κth digit lying in a string of exactly l consecutive digits $r-1$ is $1-r^{-1}$ for $l = 0$, $l(1-r^{-1})^2 r^{-l}$ for $0 < l < \kappa$ and $(\kappa-1)(1-r^{-1})^2 r^{-l} + (1-r^{-1})r^{-l}$ for $\kappa \le l$. As a check, summing these over all $l \ge 0$ gives probability 1.

The contribution to the average delay given by the first k digits is the sum over appropriate κ and l of the product of the probability of the case (κ,l) and the bound on the delay which it might cause. This yields:

$$\sum_{\kappa=1}^{k}\left(\sum_{l=t+1}^{\max\{\kappa-1,t\}}(l-t)(1-r^{-1})^2 r^{-l} + \sum_{l=\max\{\kappa,t+1\}}^{\infty}\left(\tfrac{l-t}{l}\right)\Big((\kappa-1)(1-r^{-1})^2 + (1-r^{-1})\Big)r^{-l}\right)$$

For the contribution of the k+1st and subsequent digits, we sum over the lengths l of the subsequences of digits r–1 to which the k+1st digit might belong and the possible sizes p of the part which starts at position k+1. The delay to add for each case is probability of the case times the length of the sequence starting there times the delay per sequence digit. This yields:

$$\sum_{l=t+1}^{k}\sum_{p=0}^{l} p(l-t)l^{-1}(1-r^{-1})^2 r^{-l} \qquad +$$
$$\sum_{l=k+1}^{\infty}\left((l-k)(1-r^{-1})+\sum_{p=l-k+1}^{l} p(1-r^{-1})^2\right)(l-t)l^{-1}r^{-l}$$

When $k = t = 0$ the sum of the two expressions is easily $(r–1)^{-1}$. Using induction we obtain $\{kr^{-1} + (r–1)^{-1}\}r^{-t}$ for the sum when $k = t$. Then, with this as the base case, induction over k will establish the sum as $\{kr^{-1} + (r–1)^{-1}\}r^{-t}$ whenever $0 \le t \le k$. So,

LEMMA Under the above hypotheses and a built-in delay of $t < k$, the average delay for the kth digit is less than $t + \{ kr^{-1} + (r–1)^{-1} \}r^{-t}$.

Observe that for $t = 0$ this coincides with our initial, rough bound. Taking $t = log_r k$ now gives our main conclusion, namely:

THEOREM With the above assumptions about digit probabilities and independence for the input, an on-line converter from redundant to standard base r representations produces an average delay of at most $log_r k + r^{-1} + (r–1)^{-1}k^{-1}$ between inputting and outputting the kth digits.

A marginally better choice of t can be obtained for known values of r and k by putting $t = log_r k – s$ in the lemma, and differentiating the formula for the delay with respect to s in order to obtain the minimum. The theorem, with $s = 0$, provides a reasonable choice for small k, whilst taking $s = 1$ works well for medium sized k (as in Table 1). Letting k tend to infinity, the best choice is $s = 1 + log_r log_r e$, and it yields the following corollary:

COROLLARY. With the above notation and hypotheses, for sufficiently large k the average delay in establishing the kth non-redundant digit is less than $log_r k$.

6. Computational Results.

Despite the apparently reckless counting above of extra delays for digits which were not in a maximal subsequence, computational results suggest that the above formulae are fairly accurate. Indeed, from Table 1 the difference between the actual delays in adjacent columns tends to 1, as the formula suggests. So the most significant term does indeed seem to be $log_r k$ with coefficient 1, and the relative error between the true delay and our bound slowly decreases as k increases.

k	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Actual delay	1.500	1.875	2.383	3.014	3.759	4.592	5.485	6.420	7.382	8.360	9.348
Lemma bound	1.500	2.000	2.500	3.250	4.125	5.062	6.031	7.015	8.008	9.004	10.002

Table 1. Delays for $r = 2$.

Exact computation of the true delay is not trivial because of the large number of numbers involved. The most efficient manner found was the following. For each κ, the number of representations of length κ with a maximal subsequence of digits r–1 with length l and an end subsequence of such digits with length l' were tabulated. The results were used iteratively to compute the figures for κ+1 from those of κ, with a very careful tabulation order to minimise disk accesses and make the accesses sequential. The table was computed in this way and similar results were also obtained for some small $r > 2$.

7. Conclusion.

The aim of this paper has been to prove the theorem of section 5, which shows that in converting numbers from a redundant representation to a non-redundant one in a most significant digit first order, the average delay due to waiting for carries to propagate is bounded essentially by the logarithm of its length. This result depends on several realistic suppositions about the way the digits are distributed. Computational results indicate that the bound is close to being the least upper bound.

References.

[1] A. Blum & P. Chalasani, *An On-Line Algorithm for Improving Performance in Navigation*, Proc. 34th Symp. on Foundations of Computer Science, 1993, pp. 2-11.

[2] M. D. Ercegovac & T. Lang, *On-Line Arithmetic: A Design Methodology and Applications in Digital Signal Processing*, VLSI Signal Processing **3**, R.W. Broderson & H.S. Moscovitz (Eds.), IEEE Press, N.Y. 1988, pp. 252-263.

[3] M. D. Ercegovac & T. Lang, *On-the-fly Rounding for Division and Square Root*, Proc 9th Symp. on Computer Arith., Santa Monica, Ca, USA, 6th-9th Sept 1989, IEEE Press, 1989, pp. 169-173.

[4] M. D. Ercegovac & T. Lang, *On-the-fly Conversion of Redundant into Conventional Representations*, IEEE Trans on Computers, Vol. **C-36**, No. 7 (July 1987), pp. 895-897.

[5] T. Stouraitis & C. Chen, *Fast digit-parallel conversion of signed digit into conventional representations*, Electronic Letters, Vol. **27**, No. 11 (May 1991), pp. 964-965.

23rd September 1996