W. W. Peterson*

On Checking an Adder†

Abstract: It is widely known that a computer adder can be checked by a completely independent circuit using check symbols that are residues of the numbers modulo some base. This paper describes such a residue checking system and shows, moreover, that independent adding and checking circuits are possible only with systems of this type. The discussion includes a method of handling residue-class check symbols when overflow occurs.

Introduction

When the checking circuit forms an integral part of an adder, there is the possibility that the failure of a single component will affect both the result and the check in such a way that an undetected error will occur. It is therefore important to determine what type of check can be accomplished when the checking circuit is made completely independent. The use of a residue-class check such as "casting out nines" is well known. The present paper demonstrates that this is the only type of checking possible where adder and checker are independent. Every check under these circumstances is the same as some residue-class check.

Independent check, modulo some base *b*

The checking system studied in this paper is shown in Fig. 1. The principal feature of this system is the separation of the adder and the checker into independent units. Corresponding to each number n which may occur in this system, there is a check symbol $C(n)$. Whenever a number is stored in the system, the corresponding check symbol is stored with it. When two numbers, n_1 and n_2, are to be added, they are sent to the adder, and their check symbols, $C(n_1)$ and $C(n_2)$, are sent to the checker, which is entirely separate from the adder. The output from the adder is n_1+n_2, and the output from the checker, denoted by $C(n_1)*C(n_2)$, must be the check symbol corresponding to n_1+n_2; that is,

$$C(n_1)*C(n_2)=C(n_1+n_2). \tag{1}$$

A validity check can then be made to see whether the actual outputs from the adder and checker agree.

Such a system of checking can be achieved by using the residue of the numbers modulo some fixed base b as check symbols, and using a checker which is a "mod b" adder. These residue-class checks work because adding and then taking residues gives the same result as taking residues first and then adding.[1] In symbols, if

$$C(n)\equiv n \bmod b \tag{2}$$

then

$$C(n_1+n_2)=C(n_1)+C(n_2)\,, \tag{3}$$

where it is understood that the addition on the right is modulo b. That is, b is added to or subtracted from the

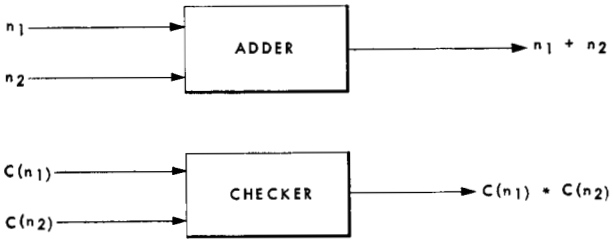

Figure 1 **Checking system with independent adder and checker.**

*Now at University of Florida, Gainesville, Fla.

†Most of the work reported in this paper was done at the IBM Research Center, Poughkeepsie, N. Y.

result if necessary to make the result lie between zero and $(b-1)$ inclusive. If the checker operation (*) is addition, Eq. (3) is the same as Eq. (1).

Proof that independent checks are always residue-class checks

This paper shows that every checking system of the type under discussion is the same as a check modulo some base b. The check symbols can always be considered to represent in some code all the numbers between zero and $(b-1)$ in such a manner that Eq. (2) holds and the checking operation (*) is equivalent to addition modulo b, as in Eq. (3). The proof consists of showing what this code must be and demonstrating that with this code Eqs. (2) and (3) do hold.

Clearly $C(0)$ must be taken as the code symbol for 0. If $C(1)$ is different, it is taken as the code symbol for 1. If $C(2)$ is not the same as either $C(0)$ or $C(1)$, it is taken as the code symbol for 2. Similarly, the process continues until a number b is found such that $C(b)=C(a)$, where a is a positive number smaller than b.†

Since $C(b)=C(a)$, Eq. (1) implies that

$$C(0)=C(a)*C(-a)=C(b)*C(-a)=C(b-a); \quad (4)$$

that is, $C(b-a)$ is the same as $C(0)$. Because of the way it was chosen, however, b is the smallest number such that $C(b)$ is the same as the check symbol for a smaller number. Therefore,

$$b \leqq b-a. \quad (5)$$

But,

$$a \geqq 0, \quad (6)$$

and (5) and (6) can hold simultaneously only if $a=0$, and hence

$$C(b)=C(0). \quad (7)$$

Since $C(n+b)=C(n)*C(b)$, we can now state that

$$C(n+b)=C(n)*C(0)=C(n+0)=C(n), \quad (8)$$

and also that

$$C(n+kb)=C(n). \quad (9)$$

Thus, if two numbers are congruent mod b, they have the same check symbol. Since every integer is congruent mod b to some number between 0 and $(b-1)$, there must be exactly b distinct check symbols, and these symbols are the coded representation of the numbers, 0, 1, 2, ... $(b-1)$. Furthermore, every number $n \geqq b$ has associated with it the check symbol corresponding to the residue of n mod b.

Finally, if the code for n_1, $C(n_1)$, and the code for n_2, $C(n_2)$, are put into the checker, the output is $C(n_1)*C(n_2)=C(n_1+n_2)$, the code for (n_1+n_2) mod b. Thus the star operation on check symbols is equivalent to addition mod b on the code representation, and the checker is essentially a mod b adder.

The proof is essentially algebraic, and the ideas behind it can be described very compactly in terms of modern algebra. The property which the check symbol function $C(n)$ must possess, indicated in Eq. (1), makes this function a *homomorphism*.[2] The theorem is essentially equivalent to the statement that every homomorphic image of the integers is isomorphic to the set of residue classes of integers modulo some number. This follows from the fact that the integers are cyclic, that the homomorphic image of a cyclic group is a cyclic group, and that every cyclic group is isomorphic either to the integers or to the integers modulo some number.[3]

Handling overflow

It has been assumed so far that the adder itself adds the numbers correctly. This will be the case if the sum of the two numbers is not too large for the computer to handle without overflow. Convenience in machine application usually requires that the outputs of the adder and checker should agree even when there is overflow. This brings up a question first as to the condition under which such agreement will occur with residue-class checks, and second as to what modification might be made to bring this condition about if agreement does not occur.

Typically when overflow occurs in an adder, the calculated result differs from the true answer by some fixed amount: 2^k in a k-bit binary adder, (2^k-1) if it has "end-around-carry," or 10^k in a k-digit decimal adder. This means that typical adders are adders modulo 2^k, 2^k-1, or 10^k. To build an adder modulo some other number to use for checking would require only minor modifications of present techniques.

If the difference between the output from the adder and the true result is N when there is overflow, then agreement of check symbols requires that

$$C(n+N)=C(n).$$

Then

$$C(N)=C(n+N-n)=C(n+N)*C(-n)$$
$$=C(n)*C(-n)=C(n-n)=C(0).$$

Since $C(n)$ must be a residue-class check,

$$N \equiv 0, \text{ mod } b,$$

where b is the base of the check. Thus a necessary and sufficient condition for agreement of check symbol with adder result, even with overflow, is that the base of the check divide N.

In the ordinary binary adder, where $N=2^k$, b must be power of two, let us say 2^p, in order to divide N. Thus the check symbols are the numbers modulo 2^p, or essentially duplicates of the p low-order bits. Because no check is provided on the bits of higher order than p, such a check would be of very little use.

A similar result occurs in the decimal case. Here

†If no number b is found such that $C(b)=C(a)$ with $a<b$, then there must be a different check symbol for every number, and the checker is essentially a duplicate adder.

$N=10^k$, and in order to divide N the base b must have the form $b=2^p5^q$. There is a complete check on digits of order lower than p and q, and no check at all on digits of order higher than the greater of p and q.

For the binary adder with end-around-carry, $N=2^k-1$, and if this has any factors of suitable size, a residue-class check based on it would probably work out well.

Certainly in the case of the ordinary binary adder, if a residue-class check is used, the base would not be chosen to be a power of 2, and some modification would be necessary to make the check symbol and adder result consistent when overflow occurs. Fortunately, the necessary modification is simple. When N is dropped from the output of the adder because of overflow, the overflow indication can be used to cause the residue of N modulo b to be subtracted from the check symbol. This feature may in fact be very desirable; if the overflow signal failed, the check symbol would not be corrected, and the error would be detected by the next validity check.

Conclusion

This paper demonstrates that if an independent checker is not a duplicate adder, it must always employ a check of the residue-class type. It has been shown, too, that such checks are generally feasible. Although somewhat negative, these results do suggest some directions for constructive thought on the checking problem.

In the first place, added emphasis is placed on the residue-class checks, and further investigation of their properties appears worth while. A base-three check will provide single-error detection for the ordinary binary and decimal adders. It would be interesting to know, for example, what base should be used for multiple-error detection or for a given degree of error correction with a binary or decimal k-place adder.

In the second place, in situations where a residue-class check is not suitable, integration of the checking and adding circuits is indicated, and all effort can be turned immediately to investigation of integrated systems.

References

1. G. Birkoff, and S. MacLane, *A Survey of Modern Algebra,* Macmillan, New York, 1941, pp. 23-29.
2. B. L. van der Waerden, *Modern Algebra,* Frederick Ungar Publishing Co., New York, 1949, p. 27.
3. G. Birkoff, and S. MacLane; *op. cit.,* p. 138 and p.155.

Received October 17, 1957