machine has an advantage over similar binary machines. The logic connectives used, as well as other instructions, will be monitored and their usefulness evaluated.

The way in which the machine is implemented, i.e. in Version 1.2 of a level-1 machine on top of QM-1 (5) dictates the procedures of evaluation. As there was no attempt to utilize in any optimal way the binary hardware on which we are emulating, there is no point whatsoever to do any speed comparisons between the ternary and binary computers. The final evaluation will consist of monitoring the types of operations performed, the number of instructions used in each category, memory utilization, etc. These numbers will then be compared to binary machines performing the same type of work. Relations between these two sets of data will be used for evaluation purposes.

We now refer the reader to Part II where the results of our preliminary work are reported.

Part II - Emulation

INTRODUCTION

As the first part of a complete emulation of a ternary computer, the authors undertook the study of the emulation of the balanced ternary arithmetic unit.

Briefly, a balanced ternary number is a string of symbols a of the form

$$a_m a_{m-1} \cdots\cdots a_1 a_0 . a_{-1} a_{-2} \cdots\cdots a_{-\ell}$$

This string represents a number

$$\sum_{i=-\ell}^{m} a_i * 3^i$$

where a_i [-1, o, 1], and, as pointed out earlier, a are referred to as trits. The numbers represented are either positive or negative. The sign is determined by the leading nonzero trit.

In what follows we shall describe the representation of trits on our binary hardware, and our implementation of a fixed point balanced ternary unit. The implementation goals were:

1) Easy representation of trits on binary hardware

2) Suitability of that representation for logical and non-balanced operations.

3) Parallel operations on trits in at least addition or subtraction.

4) Adequate speed. By that we mean speed which will enable us to use this unit in a computer and still be able to run meaningful experiments.

5) Implementation in a dynamically changeable microprogrammed environment.

The first goal is dictated by our usage of binary hardware. The second point reflects our wish to be able to implement, eventually, a complete ternary machine. The third and fourth are really connected. The former reflects the fact that serial operations, as will be pointed out in the conclusions, are too slow. The latter is triggered by our wish to have a usable system. Rather arbitrarily, we defined "adequate speed" to be not less than the speed of IBM 360/30. Point five reflects our desire to be able to change and adopt the arithmetic unit to reflect our hopefully growing understanding of ternary processes.

Representation:

The following notations will be used:

Xt, At, Bt	Balanced ternary numbers
'+'	Balanced ternary addition operator
X, X1, X2, A1, A2	Binary numbers

The balanced ternary number system, as presented in the introduction, is a base 3 system using the numbers -1, 0, and +1. We shall use T to denote -1 (T is chosen to denote -1 because it looks nice on computer output, resembling a 1 with a bar on top). To facilitate future discussions, all balanced ternary numbers will be suffixed with the letter "t", e.g. 1t, 0t, Tt, Xt, At, Bt, where Xt, At and Bt are symbolic representations of balanced ternary numbers. Note that Tt has the special meaning of -1t. A number, or the symbolic representation of a number, that does not have the suffix "t" will be a binary number. The following are some examples:

Balanced ternary: 1t, 0t, Tt, 1101T1t, TT01T1t, At, Bt, Xt.

Binary: 1, 0, 10111, A, B, X, A1, A2, B1, B2, X1, X2, LB.

In deciding on the binary representation of a trit, we would like to have separate bits for +1 and -1, and denote the bit for +1 as the positive component and the bit for -1 as the negative component. In general, it appears plausible to have the positive and negative components of a balanced ternary number separated.

Therefore, a balanced ternary number will be represented by 2 binary numbers, e.g. Xt = (X1, X2). Note the digits 1 and 2. X1 will always denote the positive component of Xt, and X2 will always denote the negative component of Xt. In particular, for representing 1 trit:

1t = (1, 0),

0t = (0, 0), and

Tt = (0, 1).

A(1, 1) pair can be considered as (1, 0) '+' (0, 1) = (0, 0). Implementation assures that if a (1,1) pair occurs as a result of arithmetic operations, it is set to (0, 0). This can be interpreted as the addition of positive and negative components that cancel each other.

Some examples of balanced ternary numbers and their binary representations are:

Balanced Ternary	Binary representation
T1t	(01, 10)
11Tt	(110, 01)
T1T0t	(0100, 1010)

Some advantages of this representation are:

a) Nearly all the primitive operations available on our binary host machine, and as a matter of fact, on most microprogrammable computers today, can be used without masking or shifting.

b) The sign of a balanced ternary number can be determined from the relative magnitude of the positive and negative components.

c) Because the complete binary word is used as a unit, a reasonable magnitude range is available. For example on a 16 bit host machine, the range is from (3**16-1)/2 to -(3**16-1)/2, or from 21,523,360 to -21,523,360. This flexibility in range, plus the status information (see "d" below), allows simple implementation of multiple precision and floating point balanced ternary arithmetic.

d) Status information supplied by the primitive operations of the host machine can be used directly. Examples of status information are: sign of result, overflow, carry out, result equal zero etc.

e) The positive and negative carries are easily separated with Boolean operations, and this results in a reasonably fast method for balanced ternary addition, and hence a reasonable speed for all of the computer.

Using this representation, microcode to perform the four basic operations was written,(6). This code, as all other starting codes in Project MU, was written in level-1 language(s). As the QM-1 machine is not yet operational, the code was run on a simulator. The expected timings of QM-1 were used in the evaluation in all stages. The results of this evaluation are presented in the next section.

CONCLUSION

Relative to the goals stated in the introduction we can say the following:

We have a simple and usable representation of trits on binary hardware. This representation is suitable for completely parallel balanced ternary addition and subtraction and seems to be suitable for those logical operations that are necessary for the implementation of ternary logic [8].

Moreover, we are now convinced that the emulation approach to a ternary computer is a suitable and viable one. This conviction is based on timings and control store space comparisons presented in Table 1. In that table (columns 2, 3 and 5) we compare emulation speeds, on the same hardware, of "traditionally" emulated arithmetic structures such as multiplication, division and binary floating point, with the balanced ternary. It is of interest to note that though ternary structures are not "native" to a binary machine, their emulation is as efficient as the binary structures.

In addition, based on our previous knowledge of the speed, about 10 microseconds at worst, of instruction fetch and decoding in complicated architectures [10], we believe that our arithmetic unit will enable us to achieve our adequate speed goal. Even if we add this 10 microseconds to the

relevant table entries for balanced ternary arithmetic, we obtain speeds which are better than those for the same operations on the IBM 360/30.

The reader may note two different timings for parallel balanced ternary arithmetic in Table 1. Our implementation and timings depend on the primitives supplied by the micro-language. However, since we are in a dynamic microprogramming environment, the primitives can be changed easily. Such a change, and in fact a very trivial change, done by adding a SWAP registers instruction and an AND-NOT instruction to the micro-language, resulted in a speed saving of about 10% in the addition and subtraction (See columns 5 and 6, Table 1).

At this time, we are working on the emulation of a complete ternary computer, and the unit presented in this paper will be used in the implementation of this computer.

Table 1.

Timings per operation (in microseconds) for microcoded arithmetic operations and storage requirements in number of control store words (16 bits).

Operation	Binary integer	Binary floating point	Twitwise balanced ternary integer(N1)	Parallel balanced ternary integer	Parallel balanced ternary integer(N2)
+	(N3)	22	70	9.12	8.16
-	(N3)	22	70	9.66	8.7
min	35.82	116	175	26.46	25.5
max	36.66		1456	173.46	161.94
min	67.2	N/A	N/A	64.56	63.36
max	68.82			423.66	407.85
Precision in bits	16	24	12	25	25
Storage usage					
+		183	207		
-					
x				261	247
/					
61					

N1 Using another binary representation of a balanced ternary number, in which 2 bits of a binary word represented 1 trit, and arithmetic routines operated on 1 trit at a time.

N2 Modified Version 1.2 with special primitives supplied for A = A & B' and SWAP A, B.

N3 Supported directly as a primitive in Version 1.2.

Bibliography

(1) For an interesting list of references and summary of the state of the art in that period see "High Speed Computing Devices", Engineering Research Associates, McGraw-Hill, 1950.

(2) A fine bibliography can be found in G. Epstein, "Application of Post and Generalized Post Algebras to Finite and Infinite Valued Logic", Submitted for publication in the IEEE Transaction of Computers. See also the bibliography in reference 8.

(3) A recent summary of the state of the art can be found in "Conference Record of the 1971 Symposium, on the Theory and Application of Multivalued Logic Design", Dept. of EE, State University of New York at Buffalo, 1971. See also the forthcoming report of the 1972 conference.

(4) D. Knuth, "The Art of Computer Programming", Vol. 2, pp. 173-176, Addison-Wesley, 1969.

(5) R. F. Rosin, G. Frieder and R. Eckhouse, "An Environment for Research in Emulation", to be published in CACM, August 1972. See also the Proceeding of the 4th workshop on microprogramming, Santa Cruz, 1971.

(6) G. Frieder and C. Luk, Emulation of a Balanced Ternary Unit, Project MU, Report 32-72-MU, SUNY/AB, 1972. Part II of our presentation is based entirely on this report, not fully reproduced here for reasons of brevity.

(7) See for instance:

(1) Ref. 3, ref. 6

(2) M. A. Breuer, A Note on three valued logic Simulation, IEEE Trans. C-21, p. 399 (1972)

(3) P. L. Gardner, Functional Memory - Microprogramming Implication, IEEE Trans., C-20, P. 7, (1971)

(4) B. T. Chu, "Some Methods for Simplifying Switching Circuits Using Don't Care Conditions", JACM, 8, No. 4, p. 497, (1961)

(8) G. Frieder, Ternary Computers, Why and How, Project MU Report 30-72-MU, SUNY/AB 1972. Part I of this presentation is based entirely on that report, not fully reproduced here for reasons of brevity.

(9) W. H. Ware, Soviet Computer Technology, 1959, IRE Trans. Elec. Comp., Vol. EC-9, 1960.

(10) M. Lutz, H. Manthey, A Microprogrammed implementation of A Block Structured Architecture, Project MU Report 33-72-MU SUNY/AB, 1972. Also proceedings of this conference.