Memo to: Dr. S. G. Campbell August 28, 1961

Subject: Floating-Point Arithmetic

The following and a subsequent memo consist of a number of observations and suggested improvements on 7030 Floating-Point Arithmetic. It is believed that the suggestions, if implemented, will make the 7030 much faster, and vastly more powerful. This memo is concerned mainly with correcting observed deficiencies of the 7030. The subsequent memo will contain specific recommendations.

1. <u>7030 Floating-Point Arithmetic</u>

The 7030 is basically designed as a floating-point machine. The unparalleled speed and flexibility at floating point arithmetic are its basic selling points. The following is a list of some of the distinct features of 7030 floating-point arithmetic:

a. Microsecond speed.

+, -, 1.5 μs; *, 2.7 μs; /, 10 μs, *+(K2), 4.2 μs.

b. Extensive checking of data

ECC check for memory opnds and instructions;
Parity check for data transport;
Residue (mod 3) check for airthmetic.

c. Universal accumulator avoids misplacement of results.

d. Addressability of the accumulator allows direct referral of previous result as input operand.

e. Noisy mode allows the estimate of round-off error.

f. Data flagging enables interruption of program by exceptional quantities tagged with non-zero flag bits.

g. 30 basic instructions, each one capable of being modified by 3 bits: the unnormalized bit, the negative bit and the absolute bit. (29 instructions in X-1 and K-1.)

h. Provision for convenient double precision arithmetic.

i. Exponent flag to prevent exceptionally large or exceptionally small quantities to disguise as normal numbers.

2. Criticisms

The 7030 floating point arithmetic design, however, is not perfect and the following is a list of areas which could be improved.

a. The design does not solve the temporary storage problem effectively.

There are three kinds of problems which call for temporary storage: misplacement of arithmetic result (characteristic of von Neumann type of AC-MQ design), lack of accumulator addressability(as needed in multiplying the accumulator contents by itself), and possible destruction of an arithmetic result needed some steps later. The first two problems are solved conveniently on the 7030.

The third problem is solved in principle by the"forwarding" of data from one lookahead level to the other. The forwarding is expensive and timewise unrewarding and is removed from K-2 and subsequent machines. The remaining alternative of putting the result in some other internal register is not effective because of the lack of high speed linkage and because the multiplier register ($MR), installed as an afterthought, has no address.

b. Relative imbalance of speeds.

The multiply time of 2.7 us will probably not be matched for some time. One wishes however, that the floating add is faster than 1/2 the multiplication time. The load, store time on K-2 will be 0.9 to 1.2 μs usually. The outsider may, however, wonder why it is not possible to bring these down to 0.3 μs in the neighborhood of slow floating point instructions.

c. Lack of sorting-merging facilities.

The conditional branch on results of accumulator compare operations is slow. On K-2 it is fast only if the testing loops are large and if optimal programming techniques are used. There is no fast way to perform a "bubble sort" or a merge.

d. Inadequate unnormalized arithmetic facilities.

Unnormalized floating point arithmetic is theoretically capable of being faster than normalized arithmetic. In the 7030, however, no effort is made to clearly separate the two.

More important, there is no provision to tackle the overflow of the fraction part, except the LC (lost carry) type of indicator bits. If fixed-point arithmetic is to be done as unnormalized floating point arithmetic, there should be ways to allow a temporary fraction overflow.

e. Inaccessibility of the remainder on double divide operations.

The remainder register, being in location 13.0 in the instruction part of core memory, is one of the most inaccessible "registers". This means whenever a double divide instruction is used, there is a good chance of conflict with instruction access. It also creates complications within the lookahead.

The factor register in location 14.0 also in the core memory, and the LFT, *+ operations on X-1 and K-1 has the same difficulties as in double divide. On K-2 the use of the D register avoids this difficulty, except in VFL arithmetic.

f. Relative slowness of double-precision load-store arithmetic.

The convenience of the double-length accumulator and the flexibility of the class of "double" operations are hampered by the fact that the double precision load-store operations are still quite slow.

Suppose one has two numbers A1, A2 in the memory. They have the same sign, the exponent of A2 is precisely 48 units lower than that of A1. Together they represent the double precision quantity A. What is the price to put this quantity A in the accumulator?

The usual way would be to double load A1, then double add A2. The former operation takes only 1.2 μs, but the latter takes 5.4 μs because the A2 fraction has to be shifted 48 places at the price of 0.3 μs per shift of 4 units.

The situation is similar when one attempts to store the double-length accumulator operand into, say A1 and A2. The "store, store low order" sequence, of course, creates an over-demand of the lone LAAR on X-1 and K-1, but this is not serious on K-2. The store probably will not take more than 1.2 μs, but the "store low order" requires 5.4 μs because of 48 shifts, 42 of which required 0.6 μs per shift of 6 units.

It would seem more desirable to load and store the lower accumulator directly in such cases. However, there are no convenient instructions in the 29-instruction set to achieve this purpose; further, the accumulator format actually destroys the distinction between the upper and lower accumulators, since there is only one exponent, one sign byte, and the low order fraction begins at the 60th bit rather than the 64 bit of the double accumulator.

g. Overabundance of flag bits.

There are three data flag bits, T, U and V, for each floating-point number. It is found by actual programming experience that the power of the machine is indeed greatly enhanced by the availability of one flag bit. There are occasional needs for a second bit, but there is actually no case in actual computation that three bits are really needed. It would seem desirable to turn the unused bit(s) to better use.

h. Absence of progressive indexing facilities.

All 7030 VFL instructions have progressive indexing facilities. Progressive indexing is hard to learn, mainly due to the unnatural divorcement from effective address creation. Once learned, however, it is an extremely powerful tool, making bug-free programming much easier. It is obvious that the hardware installed for progressive indexing is equally useful to floating point computations, if there is a way of specifying progressive indexing in floating point instructions.

Of the eight progressive indexing options, 6 are concerned with altering of the index register being used, one is for normal indexing, and one is to specify an immediate operand. It is easy to speculate the power of an instruction set which allows immediate operands. "Multiply the accumulator by 17" is obviously more direct and to the point than "multiply the accumulator by the contents of location 12000. (I hope it still has the number 17 I put there 5 seconds ago)".

i. <u>Incomplete use of power inherent in the design.</u>

All the criticisms above can actually be summarized by this sentence.

The floating point arithmetic was built chiefly for speed, of course. Accompanying this investment is potentialy for much greater power and flexibility. Both of the latter are already quite apparent in X-1, much more so in K-2. However, even in the K-2 the computing potential of the machine hardware remains largely untapped, and the machine as much fulfills the needs only of somewhat specialized market.

3. Making Full Use of 7030 Hardware.

Great harm can be caused by assuming that asynchronous designs yield mostly improvements in speed, and are destined only for specialized markets; that there is no good way to improve the performance of the 7030 design without exorbitant additional cost.

The implementation of the 7030 high speed design in fact, already has introduced hardware features to allow large performance enhancement without serious additional cost.

The points raised in the last section are answered below through modest redesign of the instruction set and improvement of hardware linkages.

a. An easily implemented three-accumulator system consisting of the left accumulator, the right accumulator and the multiplier register will not only resolve the third temporary storage problem, but will greatly enhance the power and flexibility of the machine.

b. Operations referring to different accumulators can now coexist in the E-box with no fear of interference. This increases the performance of operations in such a manner as to reduce the shifting cost. The relative imbalance of instruction speeds will be redressed. Unnormalized loads and stores can be done in the lookahead at the rate of 0.3 μs each.

c. One new instruction, KL (compare load: load low order, compare with high order and exchange to insure low order is smaller) greatly enhances "bubble sorting".

One new interruptible indicator bit $KC (comparision status change), which is turned on only if the present compare instruction execution changes the compare type indicator bits, greatly facilitates automatic merging, and incidentally removes the need for "compare for range" type of instructions.

d. More vigilance will be paid to the "unnormalized" modifier bit. The V flag will be used as fraction overflow bit, and U flag will be used as "fraction double overflow" trigger.

e. Remainders will be put in the multiplier register.

f. The independent status of the low order accumulator (see (a)), automatically reduces the time for double precision load and stores to reasonable values.

g. By (d), the U, V flags now sees proper usage.

h. Basic reshuffling of floating point instruction set allows every instruction the freedom of being "postgressively" indexed. The postgressive indexing scheme is fast, easy to learn, and just as powerful as progressive indexing (see an earlier Memo on Progressive Indexing, July 28, 1961.)

i. The result is a three-accumulator machine, each one is capable of doing independent arithmetic, yet for special instructions (*+ and D*, etc.) can cooperate with and complement each other. With a 64-instruction set, each instruction being further postgressively indexable, the machine will have power worthy of the hardware investment. The additional hardware investment is very small, and is dependent on the amount of overlap in (b) to speed up the execution of instructions. A more systematic account of the design will be presented in a subsequent memo.

a Discussions with Don Gibson (Kingston) has been instrumental in arriving at the new design, particularly the three accumulator concept and speed-up possibilities. Stimulations also have been received from C. T. Apple, N. Hardy, R. Rockefeller (Kingston), G. Zook (Kingston) and G. Hira.

[illegible]
Tien Chi Chen
Special Studies

TCC:tm