Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.313 Contemporary Computer Design

Problem Set #5 Solutions Fall, 1995, Wed 18 October

1. The design of the CL-1 has several problems. The full, non-redundant representation is required for a number of important tasks including:

- Conditional branches. A conditional branch requires a comparison between two quantities. If a magnitude comparison is necessary, a non-redundant representation must be obtained.
- Address generation. The CL-1 cannot manipulate pointers effectively.
- Overflow detection. During an arithmetic operation, overflow can occur and not be detected, since it will become obvious only when the carry is computed.

Now, the CL-1 could be used in a number of very specific computations without a worry. For example, if it were used only for numeric computation with no branches, and if the sizes of the operands were known and guaranteed not to cause overflows, the computer would do fine. This can occur in image processing applications. For example, when adding 8-bit numbers (pixel values) to a 16-bit sum, we know that overflow can not result until 256 numbers are summed. If the CL-1 had a capability to compute carries, even if that was very slow, the rarity of carry computation would not affect performance much. Nevertheless, these types of specific computation are rare, and the CL-1 is doomed as a general purpose computer.

2. a) 001011011100_2 =1 -1 -1 2 -1 0_4 = 732_10

Recoding requires delay of order N since borrows must "ripple" through the recoding operation.

b) The encoding is the same, 1 -1 -1 2 -1 0_4, but it may now be done in constant time.

c)

1011011100	1 -1 -1 2 -1 0	732
x 0001000101	0 0 1 0 1 1	x 69
1011011100	1 -1 -1 2 -1 0	6,588
101101110000	1 -1 -1 2 -1 0 0	43,920
1011011100000000	1 -1 -1 2 -1 0 0 0 0	
1100010101001100	1 -1 0 1 1 1 1 -1 0	50,508

d) Nothing need be changed to support two's complement multiplication, as long as both inputs are arithmetically sign extended.

3. If we use one log table to find the log of both inputs, we require a maximum of 256 entries. Precision of 10^{-5} is required on the input logs (to perform 2×128 accurately) which requires 17 bits of precision to the right of the binary point and two bits (log 255 = 2.406...) for the integer part, for 19 bits of precision total.

Multiplying two 8-bit inputs produces 65,536 (256x256, or 2^{16}) possible results, including duplicates, setting the maximum size of the inverse-log table. We must be able to resolve differences of 256 between products (since we're only interested in the 8 MSBs), so we want to know the smallest difference in logs between two numbers in our range which differ by 256. For a lower bound, take the difference in logs between 255^2 and 255×254. The products differ by 255, just below our detection threshold, and the difference in their logs is the smallest in our range ($\log 255^2 - \log(255 \times 254) = \log \frac{255^2}{(255 \times 254)} = \log(255/254)$). The difference is 0.001706. This requires ten bits of precision past the binary point. Using two bits for the integer part ($\log(255) = 4.813\ldots$) requires a total of 12 bits of precision for the inverse log table.

Overflows will not occur. To handle signed numbers, one technique is to determine the signs of the input numbers and calculate what the sign of the output should be. Then perform the lookup and log addition with on the positive input numbers and negate the output if necessary.