Computer Arithmetic 2016-2017 | **Homework 2 Solutions**

1 Residues for the range ±32

The range ±32 requires $2 \times 32 + 1 = 65$ different representations. We add one to count the number '0'. If only moduli of the form 2^k and $(2^k) - 1$ are allowed then the solution that gives the minimum time delay (minimum number of bits in the largest modulus) and minimizes the total number of bits is:

$$7 \times 4 \times 3 = 84.$$

I am not going to write down all the tables needed here. The complete solution should include them.

The operation is carried by first finding the representations in the tables:

$$\begin{aligned} 3 &= (3,3,0) \\ -3 &= (4,1,0) \\ 2 &= (2,2,2) \\ 7 &= (0,3,1) \end{aligned}$$

then the multiplication is performed as:

$$-3 \times 2 = (8\mathbf{mod_7}, 2, 0) = (1,2,0).$$

Finally, the addition

$$\begin{aligned} -3 \times 2 + 7 &= (1,2,0) + (0,3,1) \\ &= (1, 5\mathbf{mod_4}, 1) \\ &= (1,1,1). \end{aligned}$$

By checking the tables, the last representation corresponds to 1 which is the result of $-3 \times 2 + 7$.

2 Parity and error checking

1. Any addition of numbers that generates an odd number of carries works as a counter example.

	Operand	Parity	
	0101	0	
+	0001	1	
	0110	0	$\neq$ P(01) = 1

2. One method is to use a residue check. In this method, $P(A)$, $P(B)$, and $P(S)$ are the residue $\mathbf{mod}_\mu$ of the m bit numbers. For all m bits to affect the residue we choose μ to be relatively prime to the radix. Therefore, to get the best coverage using n bits for each residue we should choose $\mu = 2^n - 1$. Residue checks work in general because:

$$(A[+,-,*]B)\mathbf{mod}_\mu = (\,(A\mathbf{mod}_\mu)[+,-,*](B\mathbf{mod}_\mu)\,)\mathbf{mod}_\mu$$

3. Since S is an m bit number there are 2^m total representations. Assume that there are no errors in the calculation of the checksum. There are at most $\lceil \frac{2^m}{\mu} \rceil$ values which give the same $\mathbf{mod}_\mu$ value. There is only 1 correct solution. Therefore, the probability of an undetected error is as follows:

$$\text{Prob} = \frac{\lceil \frac{2^m}{\mu} \rceil - 1}{2^m} \approx \frac{1}{\mu} \text{ (for } m \gg n)$$

4. We notice from the first part that $P(P_A + P_B) \neq P_S$ only for the cases where $A + B$ produces an odd number of carries. Therefore, we create a sum check that works properly by including the parity of the carry bits.

$$P[P(A), P(B), P(Carry)] = P(S)$$

Example:

		Parity	
Carry	0010	1	
	0101	0	
+	0001	1	
	0110	0	= P(101)

---

3 Table-lookup versus logic

1. A lookup table for z uses the two 8 bit values for x and y as address bits, so that $L = 16$ address lines.

$$\begin{aligned} \text{ROM delay} &= 2 + \lceil \log_r L/2 \rceil + \left\lceil \log_r 2^{L/2} \right\rceil \\ &= 2 + \lceil \log_4 8 \rceil + \lceil \log_4 2^8 \rceil \\ &= 2 + 2 + 4 \\ &= 8 \text{ gate delays} \end{aligned}$$

2. A table to find $1/x$ has $L = 8$ address lines only.

$$\begin{aligned} \text{ROM delay} &= 2 + \lceil \log_r L/2 \rceil + \left\lceil \log_r 2^{L/2} \right\rceil \\ &= 2 + \lceil \log_4 4 \rceil + \lceil \log_4 2^4 \rceil \\ &= 2 + 1 + 2 \\ &= 5 \text{ gate delays} \end{aligned}$$

We must add the delay of an 8 bit adder:

$$\begin{aligned} t &= 4 \lceil \log_r 2n \rceil \\ t &= 4 \lceil \log_4 2(8) \rceil \\ t &= 8 \text{ gate delays.} \end{aligned}$$

Total delay = 5 + 8 = 13 gate delays

3. Now it is possible to compare the delay functions from the first two parts ignoring the ceiling functions to decide on when to use a lookup table. A single table lookup is better when

$$\begin{aligned} 2 + \log_r(n) + \log_r(2^n) &< 2 + \log_r\left(\frac{n}{2}\right) + \log_r(2^{n/2}) + 4\log_r(2n) \\ \log_4(n) + n\log_4(2) &< \log_4(n) + \log_4\left(\frac{1}{2}\right) + \frac{n}{2}\log_4(2) + 4\log_4(2) + 4\log_4(n) \\ \frac{n}{2}\log_4(2) &< 4\log_4(n) + \log_4\left(\frac{1}{2}\right) + 4\log_4(2) \\ \frac{n}{4} &< 4\log_4(n) + 1.5 \\ n &< 16\log_4(n) + 6 \\ n &\leq 51 \text{ bits.} \end{aligned}$$

Another solution that requires less steps is:

$$\begin{aligned} 2+\log_r(n)+\log_r(2^n) &< 2+\log_r\left(\frac{n}{2}\right)+\log_r(2^{n/2})+4\log_r(2n) \\ \log_r(n\times 2^n) &< \log_r\left(\frac{n}{2}\times 2^{n/2}\times(2n)^4\right) \\ n\times 2^n &< 8n^5\times 2^{n/2} \\ 2^{n/2} &< 8n^4 \\ n &\leq 51 \text{ bits.} \end{aligned}$$

Despite the above calculation, from a practical point of view, when n is in the range of 8 to 10 bits many designers will switch to logic gates instead of using a table. Beyond that range the large size of the memory (2^{2n} entries) is prohibitive.

In the field of computer arithmetic in general, tables are better for operands represented by a small number of bits. As the operand sizes get larger the use of logic to perform the operations becomes more efficient.

4 RNS and clocks

The required range to represent the hours is 24 while that for the minutes and seconds is 60. If any integer modulus is permitted then the sets are chosen as:

Hours	8×3	$=24$
Minutes	$5\times 4\times 3$	$=60$
Seconds	$5\times 4\times 3$	$=60$

These sets represent the required range exactly, use the minimum total number of bits for each range, and minimize the longest carry propagation delay.

The operation

	hours	minutes	seconds
	13	10	55
+	10	12	04
	23	22	59

is performed as:

	hours	minutes	seconds
	$(5,1)$	$(0,2,1)$	$(0,3,1)$
+	$(2,1)$	$(2,0,0)$	$(4,0,1)$
	$(7,2)$	$(2,2,1)$	$(4,3,2)$

The results of the RNS operation correspond to $23h\ 22m\ 59s$. If large numbers are added then an overflow may occur and a carry from the seconds to the minutes or the minutes to the hours might be needed. This overflow is not easily detected in residue number systems.

In CMOS circuits, any node that switches its value consumes power. The limited carry propagation in RNS reduces the amount of node switching and hence is suitable for low power applications. From another point of view, a system that is inherently faster than another one (RNS faster than binary) can be run at a lower clock frequency to achieve the same results while consuming less power.

Note: Some of you suggested the use of $7\times 4=28$ for the hours. This set uses the minimum total number of bits and minimizes the longest carry propagation delay. However, it does not represent the required range exactly. This complicates the circuits slightly since we must implement a block for modulo M=24. Otherwise, $15+10=25$ and not 1 as expected in the hours of a clock. I accepted this solution from those who wrote it but I would like to remind you that the exact representation of the range has its own merit.

5 Modulus is $\mu = 2^k + 2^{k-1}$

1. Basically, when $Y < \mu$ (i.e. $y_k y_{k-1} \neq 11$) the result of $Y\mathbf{mod}_\mu = Y$. When $Y \geq \mu$ (i.e. $y_k y_{k-1} = 11$) the result of $Y\mathbf{mod}_\mu = Y - \mu = 00y_{k-2} \dots y_0$.

   This means that in both cases the bits $y_{k-2} \dots y_0$ are not changed while if $y_k y_{k-1} = 11$ both of them are turned to zeros. A simple AND gate detects that they are both 1 and its output is XORed with the values of the two bits to give the required result.

2. For the case $i = 0$, $2^0\mathbf{mod}_\mu = 1\mathbf{mod}_\mu = 1$ which is equal to $(2^{k-1})^0$ and the relation is verified.

   For $i = 1$, $2^{k+1}\mathbf{mod}_\mu = (2^k + 2^{k-1} + 2^{k-1})\mathbf{mod}_\mu = (\mu + 2^{k-1})\mathbf{mod}_\mu = (2^{k-1})\mathbf{mod}_\mu$ which validates the relation as well.

   For $i > 1$,

$$\begin{aligned} 2^{i(k+1)}\mathbf{mod}_\mu &= ((2^{k+1})^i)\mathbf{mod}_\mu & (1)\\ &= (((2^{k+1})\mathbf{mod}_\mu)^i)\mathbf{mod}_\mu & (2)\\ &= (2^{k-1})^i\mathbf{mod}_\mu. & (3) \end{aligned}$$

3. We divide the bits of X to groups each with $k+1$ bits such that

$$X = \sum Y_i 2^{i(k+1)}.$$

   Now,

$$\begin{aligned} X\mathbf{mod}_\mu &= \sum (Y_i 2^{i(k+1)})\mathbf{mod}_\mu \\ &= \sum (Y_i\mathbf{mod}_\mu (2^{k-1})^i\mathbf{mod}_\mu)\mathbf{mod}_\mu \end{aligned}$$

   If the number of bits n within X is not a multiple of $k+1$, we pad X with zeros on the MSB side if it is unsigned to make the total number of bits a multiple of $k+1$. If X is a signed number then slight modifications are needed at the most significant $k+1$ block of bits.
