EE486—Computer Arithmetic
Homework #2 Solutions (35 pts)

(4) Problem 2.2 - Choosing RNS Moduli

The range 0 to 10 000 requires 10 001 distinct representations.

(2) a) If any integer modulus is permitted.
Simply selecting the product of primes gives:

$$2 \times 3 \times 5 \times 7 \times 11 \times 13 = 30\,030 > 10\,001$$

However, we can reduce the total number of bits needed by using powers of the lower order primes.

$$2^2 \times 3^2 \times 5 \times 7 \times 11 = 13\,860 > 10\,001$$

(2) b) If only moduli of the form 2^k and $(2^k) - 1$ are allowed then there are several possible solutions that give minimum delay where

$$\text{Delay} \propto \lceil \log_2 \alpha(M) \rceil = 5\,\text{carries}$$

The best of these solutions choose moduli so that a total of 14 bits are need to represent a number in this system. Two such solutions are:

$$3 \times 7 \times 16 \times 31 = 10\,416 > 10\,001$$
$$15 \times 31 \times 32 = 14\,880 > 10\,001$$

(7) Problem 2.5 - Error Checking

(2) a) Any addition of numbers that generates an odd number of carries will work as a counter example.

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

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

$$(A[+,-,*]B) \bmod X = (\,(A \bmod X)[+,-,*](B \bmod X)\,) \bmod X$$

(1) c) 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 $\lceil 2^m/X \rceil$ values which will give the same mod X value. There is only 1 correct solution. Therefore, the probability of an undetected error is as follows:

$$\text{Prob} = (\left\lceil \frac{2^m}{X} \right\rceil - 1)/2^m \approx \frac{1}{X} \quad (\text{for } m \gg n)$$

(2) d) We notice from part a that $P(P_A + P_B) \neq P_S$ only for the cases where $A + B$ produces an odd number of carries. Therefore, we can 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) Problem 2.6 - More Moduli

The next seven factors are as follows:

$$37 \times 41 \times 43 \times 47 \times 7^2 \times 53 \times 59$$

and therefore

$$M' = 9.69 \times 10^{24} \approx 2^{83} \qquad \alpha(M') = 59$$

(8) Problem 2.7 - Addition Delay

(3) a) The lower bound for M is

$$\begin{aligned} t &\geq \lceil \log_r 2 \lceil \log_d \alpha(M) \rceil \rceil \\ t &\geq \lceil \log_4 2 \lceil \log_2 32 \rceil \rceil \\ t &\geq \lceil \log_4 2(5) \rceil \\ t &\geq 2 \text{ gate delays} \end{aligned}$$

The lower bound for M' is

$$\begin{aligned}
t &\geq \lceil \log_r 2 \lceil \log_d \alpha(M') \rceil \rceil \\
t &\geq \lceil \log_4 2 \lceil \log_2 59 \rceil \rceil \\
t &\geq \lceil \log_4 2(6) \rceil \\
t &\geq 2 \text{ gate delays}
\end{aligned}$$

(3) b) The lower bound for M is

$$\begin{aligned}
t &\geq \lceil \log_r 2 \lceil \log_d \beta(M) \rceil \rceil \\
t &\geq \lceil \log_r 2 \lceil \log_d \beta(17) \rceil \rceil \\
t &\geq \lceil \log_4 2 \lceil \log_2 16 \rceil \rceil \\
t &\geq \lceil \log_4 2(4) \rceil \\
t &\geq 2 \text{ gate delays}
\end{aligned}$$

The lower bound for M' is

$$\begin{aligned}
t &\geq \lceil \log_r 2 \lceil \log_d \beta(M') \rceil \rceil \\
t &\geq \lceil \log_r 2 \lceil \log_d \beta(59) \rceil \rceil \\
t &\geq \lceil \log_4 2 \lceil \log_2 29 \rceil \rceil \\
t &\geq \lceil \log_4 2(5) \rceil \\
t &\geq 2 \text{ gate delays}
\end{aligned}$$

(2) c) The number of gate delays for a ROM lookup for either set of moduli is determined by the number of bits required to represent the largest residues of the two numbers to be added. 5 bits are needed to represent the largest residue for M, and 6 bits are needed for M'. Therefore, L = 10 and 12 address lines for M and M', respectively. In both cases, we get the same ROM delay.

$$\begin{aligned}
\text{ROM delay} &= 2 + \lceil \log_r L/2 \rceil + \left\lceil \log_r 2^{L/2} \right\rceil \\
&= 2 + 2 + 3 \\
&= 7 \text{ gate delays}
\end{aligned}$$

(8) Problem 2.8 - ROM vs Adder

(3) a) A lookup table for z will use the two 8 bits values for x and y as address bits, so that L=16 address lines.

$$\text{ROM delay} = 2 + \lceil \log_r L/2 \rceil + \left\lceil \log_r 2^{L/2} \right\rceil$$

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

(3) b) Table look-up to find $1/x$ will have only L=8 address lines.

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

And we must add the delay for 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

(2) c) Compare the delay functions from part a and b ignoring ceiling function effects.

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

(5) Problem 2.11 - Cast Out 8's

This algorithm is really just casting out 9's in disguise. Assume we are adding two decimal numbers A and B with digits a_i and b_i to form the sum S. First we show that this algorithm for adding up the digits of the numbers and replacing 8's by -1 gives the same result as taking the mod 9 of the numbers.

$$
\begin{aligned}
A \bmod 9 &= (\Sigma a_i 10^i) \bmod 9 = [\Sigma(a_i 10^i \bmod 9)] \bmod 9 \\
A \bmod 9 &= [\Sigma((a_i \bmod 9)(10^i \bmod 9))] \bmod 9 \\
A \bmod 9 &= [\Sigma((a_i)(10 \bmod 9)^i)] \bmod 9 \\
A \bmod 9 &= [\Sigma a_i] \bmod 9 = [\Sigma(a_i - 9)] \bmod 9
\end{aligned}
$$

Therefore, this algorithm gives the mod 9 of each number and since:

$$(A \bmod X + B \bmod X) \bmod X \equiv S \bmod X$$

This checksum algorithm will always work.