

CSE 140 Homework One

June 29, 2009

*Only **Problem Set Part B** will be graded. Turn in only **Problem Set Part B** which will be due on July 13, 2009 (Monday) at 3:00pm.*

1 Problem Set Part A

- textbook 1.3
- textbook 1.5(c)(d)
- textbook 1.6(c)(f)
- textbook 2.1(c)
- textbook 2.4(a)
- textbook 2.5(c)
- textbook 2.6(b)(d)
- textbook 2.7(b)
- textbook 2.8(a)(c)
- textbook 2.13(a)(c)
- textbook 2.14(a)
- textbook 2.15(a)(b)
- textbook 2.16(a)(c)

2 Problem Set Part B

1 (Number Representation, Addition and Subtraction)

(Part A) Specify whether an overflow can occur in the operations on the incomplete 2's complement numbers listed in the following table where the x 's represent unknown values. Fill in the blanks with a "Yes" if you think the corresponding operation will generate an overflow; fill it in with a "No", otherwise.

Operations	Overflow?
$011xx0x1 + 1x1111x1$	
$01x10xxx - 100xxx1x$	
$111xx1xx + 1x1xx1x1$	
$0x11x00x + 01x1x001$	
$1x1x10xx - 1xxxx1x1$	

(Part B) Decimal numbers are usually encoded in the BCD format. For example, 57_{10} can be represented by two 4-bit BCD digits 0101 0111. To complete the addition/subtraction operations on these decimal numbers, a possible correction step might need to be applied to the result to attain the correct BCD code for the digit of the sum/difference being computed. The candidate correction operations include the following:

A Do nothing

B Add 3

C Add 6

D Add 10

The table below contains several arithmetic operations on 1-digit BCD codes where the x 's represent unknown values. For each case, fill in the box with the index of the appropriate correction operation.

	Correction operation
$0100 + 0111$	
$0xxx - 1xxx$	
$xx1x + 1xxx$	
$01xx - 00x1$	

(Part C) If $\alpha 1_{r^3} = r 2 \beta_{r^2}$, what are the values for the integers α , β and r ? (**Hint:** perhaps expanding the two sides of the given equation to a partial radix r representation may help you reason about this question more effectively.)

2 (Binary Addition)

Addition of **two** binary numbers ($x = x_{n-1}...x_0$ and $y = y_{n-1}...y_0$) is performed by adding each pair of bits, x_i and y_i , and the carry bit, c_i , from the previous bit position. Addition of each bit position results in a sum bit, s_i , and an output carry bit, c_{i+1} . Figure 1 represents the outlined process.

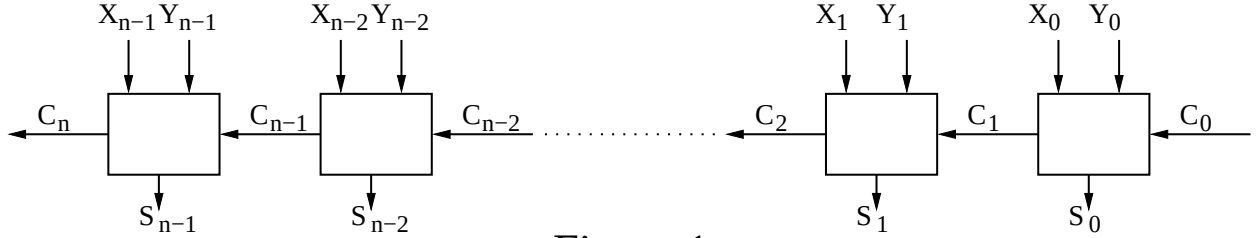


Figure 1

Given the functions (1), (2) and (3),

odd(variable list) : **True** if an odd number of variables is logic 1 (1)

even(variable list) : **True** if an even number of variables is logic 1 (2)

gte(m, variable list) : **True** if at least **m** of the variables is logic 1 (3)

one can write the *carry logic* (c_{i+1}) and the *sum logic* (s_i) as defined in (4) and (5).

$$c_{i+1} = \begin{cases} 1 & \text{if } gte(2, x_i, y_i, c_i) \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

$$s_i = \begin{cases} 1 & \text{if } odd(x_i, y_i, c_i) \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

The addition of **three** binary numbers ($x = x_{n-1}...x_0$, $y = y_{n-1}...y_0$ and $z = z_{n-1}...z_0$) can be represented as in Figure 2.

Notice that block i has **2** input carry signals (c_i and k_{i-1}) and produces 2 output carry signals (c_{i+1} and k_{i+1}).

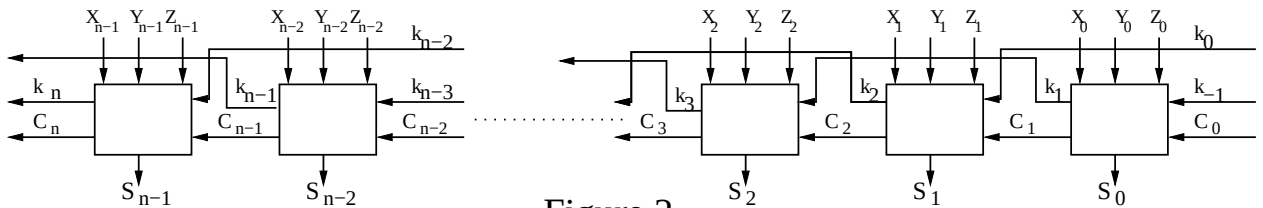


Figure 2

(Part A) Please fill the following truth table for the given input combinations.

$x_i + y_i + z_i + c_i + k_{i-1}$	c_{i+1}	k_{i+1}	s_i
0 1 1 0 0			
1 0 1 1 1			
1 1 0 0 1			
1 1 1 1 1			

(Part B) Please perform the following addition operation and specify the carry bits.

Diagram illustrating a 3-bit ripple-carry adder circuit. The inputs are c and k , and the output is s .

The inputs are represented by three boxes each:

- c : [] [] []
- k : [] [] []

The output s is represented by four boxes:

- s : [] [] [] []

The carry-in of 1 is shown entering the first box of k . The carry-out of the first box of s is connected to the second box of k , and so on, illustrating the ripple effect.

(Part C) Describe the sum logic for the 3-bit adder by **only** using the functions (1), (2) and (3). Do not forget to specify the **variable list** (and **m** if needed).

$$s_i = \begin{cases} 1 & \text{if } \dots\dots\dots \\ 0 & \text{otherwise} \end{cases}$$

(Part D) Describe the carry logic for the 3-bit adder by **only** using the functions (1), (2) and (3). Do not forget to specify the **variable list** (and **m** if needed).

$$c_{i+1} = \begin{cases} 1 & \text{if } \dots\dots\dots \\ 0 & \text{otherwise} \end{cases}$$

$$k_{i+1} = \begin{cases} 1 & \text{if } \dots\dots\dots \\ 0 & \text{otherwise} \end{cases}$$

3 (Floating Point Number)

The IEEE754 floating point number system is used extensively in computer arithmetic, but is not the only one that has been employed in processor implementations. Actually, IBM has used in their IBM System/360 computers a **hexadecimal floating point number system** that uses a similar approach to IEEE 754, but with the following differences.

- The 32-bit number is stored in a binary form, with **1 bit sign**, **7 bit exponent** and **24 bit mantissa**.
- The radix is **16**.
- There is **no** hidden bit in the mantissa.
- In the normalized form, the first (leftmost) digit of the mantissa should be a **nonzero hexadecimal** digit.
- The bias of the exponent is **64**.

We provide you below as an example the representation for the number -64 . This number would need to be expressed as -0.25×16^2 . The normalized hexadecimal radix floating point representation of this number is as follows:

Sign	Exponent	Mantissa
1	1000010	0100 0000 0000 0000 0000 0000

The arithmetic on this system is also similar to that of IEEE754 in that the last step of arithmetic operations is always a possible *normalization* of the exponent (E) and mantissa (M). We list two sets of candidate normalization operations¹ to E and M for the last normalization step of the algorithms. However, only some of them can really happen in the computation.

Operations for M	Operations for E
A. $M \ll 1$	a. $E + 1$
B. $M \ll 2$	b. $E + 2$
C. $M \ll 3$	c. $E + 3$
D. $M \ll 4$	d. $E + 4$
E. $M \ll n$, where $n > 4$	e. $E + n$, where $n > 4$
F. $M \gg 1$	f. $E - 1$
G. $M \gg 2$	g. $E - 2$
H. $M \gg 3$	h. $E - 3$
I. $M \gg 4$	i. $E - 4$
J. $M \gg n$, where $n > 4$	j. $E - n$, where $n > 4$
K. Do nothing	k. Do nothing

¹In the table, “ $\ll$ ” means left shift and “ $\gg$ ” means right shift.

(Part A) Specify which operations can happen in the final normalization step of the **Add/Sub** algorithm and which operations cannot, by filling in each box of the table with some letter combination of the operations on M and E . *Please note that multiple answers may hold for each part, in which case for full credit, please supply **all correct** answers.*

Operations on M		Operations on E	
Can happen	Never happen	Can happen	Never happen

(Part B) Given a pair of floating point numbers N_1 and N_2 in this representation, we are going to perform some operations on them. Unfortunately, only a part of information is available with these two numbers.

1. The signs of these two numbers are identical, denoted by u .
2. The exponents are known exactly.
3. The first eight bits of the mantissa are already known, but the last 16 bits are unknown.

The following table shows the available information.

	Sign	Exponent	Mantissa
N_1	u	1001001	1111 0010 ???? ???? ???? ???? ?
N_2	u	1001000	0110 0100 ???? ???? ???? ???? ?

For the various cases specified below, fill in each box of the table with some combination of the operations on M (Mantissa) and E (Exponent) which exactly describes all the possibilities that can occur in that case.

	Operations on M	Operations on E
$N_1 + N_1$		
$N_2 + N_2$		
$N_1 + N_2$		
$N_1 - N_2$		
$N_1 * N_2$		
N_1 / N_2		