

# CSE 140 Homework One

August 5, 2014

*Only **Problem Set Part B** will be graded. Turn in only **Problem Set Part B** which will be due on August 13, 2014 (Wednesday) at 4:30pm.*

## 1 Problem Set Part A

- Roth&Kinney, 6th Ed: 1.1
- Roth&Kinney, 6th Ed: 1.3
- Roth&Kinney, 6th Ed: 1.4
- Roth&Kinney, 6th Ed: 1.5
- Roth&Kinney, 6th Ed: 1.7
- Roth&Kinney, 6th Ed: 1.15
- Roth&Kinney, 6th Ed: 1.19
- Roth&Kinney, 6th Ed: 1.29
- Roth&Kinney, 6th Ed: 1.31
- Roth&Kinney, 6th Ed: 2.1
- Roth&Kinney, 6th Ed: 2.3
- Roth&Kinney, 6th Ed: 2.5
- Roth&Kinney, 6th Ed: 2.6
- Roth&Kinney, 6th Ed: 2.7
- Roth&Kinney, 6th Ed: 2.8
- Roth&Kinney, 6th Ed: 2.11
- Roth&Kinney, 6th Ed: 2.22
- Roth&Kinney, 6th Ed: 2.30

- Roth&Kinney, 6th Ed: 3.9
- Roth&Kinney, 6th Ed: 3.10
- Roth&Kinney, 6th Ed: 3.12
- Roth&Kinney, 6th Ed: 3.14
- Roth&Kinney, 6th Ed: 3.17
- Roth&Kinney, 6th Ed: 3.19
- Roth&Kinney, 6th Ed: 3.20
- Roth&Kinney, 6th Ed: 3.21
- Roth&Kinney, 6th Ed: 3.29
- Roth&Kinney, 6th Ed: 3.30
- Roth&Kinney, 6th Ed: 3.33
- Roth&Kinney, 6th Ed: 3.34

## 1 (Base Conversion)

This question concerns positional number systems and the conversion between systems with different positionally weighted values.

(Part A) Consider the following pseudocode:

```
procedure BaseConversion (a, b, c):  
    stack = an empty stack  
    print(a)  
    print(" base 10 is ")  
    while a > b:  
        push(stack, a % c)  
        a = a / c  
    while notEmpty(stack):  
        print( pop(stack) )  
    print(" when converted to base 6.")
```

Assume you want to print a base 6 representation of a base 10 number  $X$ . What should be the parameters of BaseConversion?

$a \leftarrow ( \quad )_{10}$

$b \leftarrow ( \quad )_{10}$

$c \leftarrow ( \quad )_{10}$

(Part B) Being the good computer science student that you are means that you enjoy messing around with code to see what happens. You wonder what will happen if you initialize  $c$  to 2 and increment  $c$  after the conversion of each digit. The program becomes the following:

```
procedure NewNumberSystemConversion (a, b):
    c = 2
    stack = an empty stack
    print(a)
    print(" base 10 is ")
    while a > b:
        push(stack, a % c)
        a = a / c
        c = c + 1
    while notEmpty(stack):
        print( pop(stack) )
    print(" in the new numbering system.")
```

The program now outputs numbers into a new, non-standard representation system. The largest digit value used in a number representation system usually has a relationship to the radix of that representation system. However, this new system does not have a traditional radix.

What is the largest digit that could be seen in each position after a conversion to this new number representation system? You may assume that the parameters of the procedure have the correct values from (Part A).

|                                   |   |   |   |   |   |
|-----------------------------------|---|---|---|---|---|
| Digit Number                      | 4 | 3 | 2 | 1 | 0 |
| Larget Possible Decimal Digit     | 9 | 9 | 9 | 9 | 9 |
| Largest Possible New System Digit |   |   |   |   |   |

(Part C) What are the positional weights of the digits in this new system?

|                         |      |     |    |   |
|-------------------------|------|-----|----|---|
| Digit Number            | 3    | 2   | 1  | 0 |
| Decimal Digit Weight    | 1000 | 100 | 10 | 1 |
| New System Digit Weight |      |     |    |   |

(Part D) Now that you are in love with the interesting properties of this new number representation system you have the chance to use them to do arithmetic! Perform the computation:

$$(4_{10} \times 3_{10} \times 2_{10}) \times (7_{10} \times 6_{10} \times 5_{10} + 1_{10}) = ( \quad )_{\text{new system}}$$

(Part E) In (Part B) you wrote the largest representable five digit number in this new system. Obviously, adding 1 to this number will generate the smallest six digit number which is  $100000_{\text{new system}}$ . Considering this, along with your result from (Part D), you notice that the carries seem to play nice during addition.

List all possible values for each result digit, where  $a$  denotes the same unknown digit value:

$$31a11_{\text{new system}} + 31a01_{\text{new system}} = (d_5d_4d_3d_2d_1d_0)_{\text{new system}}$$

$$d_5 =$$

$$d_4 =$$

$$d_3 =$$

$$d_2 =$$

$$d_1 =$$

$$d_0 =$$

(Part F) Now that you are confident in your ability to manipulate numbers in this new system, subtraction is trivial. Perform the computation:

$$111\ 111\ 111_{\text{new system}} - 101\ 200_{\text{new system}} = ( \hspace{10em} )_{\text{new system}}$$

## 2 (Addition and Subtraction)

(Part A) Specify whether overflow can occur in the operations on the incomplete 2's complement numbers listed in the following table where the  $x$ 's represent distinct unknown values (for example,  $0xx1$  can be any of  $0001$ ,  $0011$ ,  $0101$ , or  $0111$ ). Fill in the blanks with "Always", "Sometimes", or "Never" depending on if you think the operation will overflow in all cases, some cases, or no cases.

| Operation             | Overflow? |
|-----------------------|-----------|
| $0x11x00x + 01x1x001$ |           |
| $011xx0x1 + 1x1111x1$ |           |
| $1x1x10xx + 1xxxx1x1$ |           |
| $01x10xxx - 100xxx1x$ |           |
| $x1111111 + x1111111$ |           |
| $111xx1xx + 1x1xx1x1$ |           |

(Part B-D) After passing CSE 140 with an A+, you begin your career at a local hardware design company. Knowing about your impressive grade, your boss has asked you to evaluate the use of a new adder chip in the company's *excess-3* addition logic. This new chip is a standard 4-bit adder with one twist, namely while adding two numbers it simultaneously subtracts 0011 from the result. We denote this operation as  $X \hat{+} Y = X + Y - 3$ .

For example:

$$\begin{array}{r}
 0 \ 0 \ 1 \ 1 \\
 \hat{+} \ 0 \ 1 \ 1 \ 1 \\
 \hline
 0 \ 1 \ 1 \ 1
 \end{array}$$

(Part B) As a preliminary step, your boss asks for a technical report describing existing techniques for decimal addition. Although you have found an intern to do the writing, he lacks your illustrious education and needs your guidance. He asks you to fill in the following table with the offset (the value to be added/subtracted on correction) and the logic for when the offset is added and subtracted. Code tables for BCD and Excess-3 codes are on the back of the homework.

| Method   | Offset | When to Add | When to Subtract |
|----------|--------|-------------|------------------|
| BCD      |        |             |                  |
| Excess-3 |        |             |                  |

(Part C) Although she is excited by the prospect of two operations in one clock cycle, your boss is unsure if the new chip can provide any improvement on the existing excess-3 addition. Explain how the new chip speeds up the correction step needed in excess-3 addition.

(Part D) After you gain all the glory from these results, your intern assistant becomes jaded and decides to throw you under the bus for personal gain. He starts spreading rumors that although you are right about the improvement, you failed to account for the complication of deciding when the correction is needed. Provide an explanation about when the correction logic must be applied to avoid these rumors from killing your career.

### 3 (Error Correction and Hamming Codes)

You are hired to work at a prestigious physics laboratory studying high-energy particles. You are hired for your experience with circuit design. So far, you have designed a chip that will be used in a sophisticated geiger counter. Since the device is expected to operate in highly irradiated environments, the memories on its chip need to be resistant to the corrupting influence of high-energy particles. Thus, your manager instructs you to implement these memories using Hamming Encoding.

(Part A) Oddly, yet conveniently, the memory cells in the circuit each seem to contain 4 bits of information. Thus, all that is required to maintain memory integrity is to implement the familiar Hamming Encoding scheme you learned in an undergraduate class. Demonstrate your knowledge of this scheme to your manager by filling in the parity equations and correction equations that make up Hamming Encoding.

|       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|
| $m_3$ | $m_2$ | $m_1$ | $p_2$ | $m_0$ | $p_1$ | $p_0$ |
|-------|-------|-------|-------|-------|-------|-------|

|         |       |         |       |
|---------|-------|---------|-------|
| $p_0 =$ | _____ | $C_0 =$ | _____ |
| $p_1 =$ | _____ | $C_1 =$ | _____ |
| $p_2 =$ | _____ | $C_2 =$ | _____ |

(Part B) Your manager is relieved to know that he has not hired a complete idiot, but decides to test your understanding of Hamming Codes more thoroughly. Below, he has presented some modifications to the original parity equations. Using the table of single-bit errors and their corresponding correction bit error codes, explain how the table changes as a result of the modification.

| Wrong Bit | $C_2$ | $C_1$ | $C_0$ |
|-----------|-------|-------|-------|
| $p_0$     |       |       |       |
| $p_1$     |       |       |       |
| $m_0$     |       |       |       |
| $p_2$     |       |       |       |
| $m_1$     |       |       |       |
| $m_2$     |       |       |       |
| $m_3$     |       |       |       |

- i) The equation for  $p_1$  is XORed with  $m_3$ , so that if the previous equation is  $p_1 = x \oplus y \oplus z$ , the new equation is  $p_1 = x \oplus y \oplus z \oplus m_3$ .

| Wrong Bit | $C_2$ | $C_1$ | $C_0$ |
|-----------|-------|-------|-------|
| $p_0$     |       |       |       |
| $p_1$     |       |       |       |
| $m_0$     |       |       |       |
| $p_2$     |       |       |       |
| $m_1$     |       |       |       |
| $m_2$     |       |       |       |
| $m_3$     |       |       |       |

- ii) The equation for  $p_0$  is XORed with  $m_2$ .

(Part C) Seeing that you have demonstrated significant capability, your manager decides that you are competent to solve a problem that the other engineers have been having significant trouble with for some time. Field tests of the geiger counter that your team is working on show that, although the standard Hamming Encoding scheme from (Part A) has been correctly implemented in its chip's memories, the chip is suffering from significant memory corruption, presumably due to high-energy particle interactions with the memory cells.

After a few minutes thinking about the problem, a flash of inspiration comes to your mind. You propose to your manager that perhaps the memories are suffering from multiple-bit errors. Your manager dismisses the idea, arguing that the probability of multiple corruption incidents on the same memory cell is too low to justify the observed corruption rates. Although a sound argument, you suggest that perhaps a single high-energy particle may be consistently **flipping two adjacent bits** (we'll call it an **Adjacent 2-Bit Error**). Intrigued, your manager instructs another technician to run some tests. The results show that not only is this error indeed occurring, but it seems to be occurring exclusively – single-bit errors are not occurring at all.

For each Adjacent 2-Bit Error in the table below, show the error code that will be generated by the standard Hamming Encoding scheme, and explain why the results make Hamming Encoding unsuited to correcting this class of errors.

| Adjacent 2-Bit Error | $C_2C_1C_0$ |
|----------------------|-------------|
| $p_0p_1$             |             |
| $p_1m_0$             |             |
| $m_0p_2$             |             |
| $p_2m_1$             |             |
| $m_1m_2$             |             |
| $m_2m_3$             |             |

(Part D) Looking at the table you generated in (Part C), another flash of inspiration comes to your mind. It is possible to remove some bits from the original parity equations to enable the correction of these Adjacent 2-Bit Errors. Each original equation should have one bit removed, and the right hand side of two of the resulting equations will be the same. (Hint: there need not be any particular order or meaning in the error codes; error correction requires only that every error result in a unique code. Also, keep in mind the implications from making the modifications in (Part B), and how the fact that we are examining Adjacent 2-Bit Errors instead of single-bit errors impacts the nature of the changes.)

(Part E) Although your scheme in (Part D) may work to correct Adjacent 2-Bit Errors, it is defective in that not only can it not correct single-bit errors, which is acceptable given the rareness of this error in our case, but it cannot even detect such errors. Which single-bit error(s) cannot be detected by your scheme? Explain.

(Part F) There is a way to modify the equations generated in (Part D) which enables the *detection* of all single-bit errors without compromising the ability of the scheme to *correct* all Adjacent 2-Bit Errors. Describe the modifications that are necessary to your equations in order to pull this off. (Hint: try to think of a modification that solves the problem which should be apparent from (Part E) which, as a side-effect, exchanges the error codes for two of the Adjacent 2-Bit Errors. You may need to modify more than one equation.)

BCD Codes:

| Decimal | 2421    |
|---------|---------|
| 0       | 0 0 0 0 |
| 1       | 0 0 0 1 |
| 2       | 0 0 1 0 |
| 3       | 0 0 1 1 |
| 4       | 0 1 0 0 |
| 5       | 0 1 0 1 |
| 6       | 0 1 1 0 |
| 7       | 0 1 1 1 |
| 8       | 1 0 0 0 |
| 9       | 1 0 0 1 |

Excess-3 Codes:

| Decimal | Excess-3 |
|---------|----------|
| 0       | 0 0 1 1  |
| 1       | 0 1 0 0  |
| 2       | 0 1 0 1  |
| 3       | 0 1 1 0  |
| 4       | 0 1 1 1  |
| 5       | 1 0 0 0  |
| 6       | 1 0 0 1  |
| 7       | 1 0 1 0  |
| 8       | 1 0 1 1  |
| 9       | 1 1 0 0  |