Representing Negative Numbers in a Ternary Computer

Kaleb Roggenkamp

SENIOR HONORS THESIS
Submitted In Partial Fulfillment of Requirements of the College Scholars Program
North Central College

May 27, 2012

Approved: ________________________________ Date: __________

Thesis Director Signature
Matthew Pons, Ph.D

Approved: ________________________________ Date: __________

Second Reader Signature
Stephen Renk, Ph.D
Abstract

Within current binary computers many different number representations systems have been developed to represent negative numbers, but will those systems continue to work as we develop computers which are ternary based? This thesis seeks to answer this question by first analyzing the current systems to find their strengths and weaknesses and then attempting to abstract two of those systems, Two’s Complement and Excess $2^{n-1}$ to the ternary world. In the process we discover a few unknown features of our binary systems and we discover that they do, in fact, abstract out to a ternary system quite effectively.

Table of Contents

I. Introduction

II. Binary
   i. Signed Bit
   ii. One’s Complement
   iii. Two’s Complement
   iv. Excess $2^{n-1}$

III. Ternary
   i. Complement
   ii. Excess

IV. Conclusions

V. Appendix
   i. Cayley Tables

VI. Bibliography
I Introduction:

In this Thesis we investigate the various systems used to represent integers within the hardware of computers. After exploring these systems, we will generalize them in an effort to determine if they are scalable to a future world where computers have evolved from binary machines to ternary machines.

The first step in this process is to determine what an ideal system would be like; then we compare the current systems against those standards to determine their effectiveness. The first note on an ideal system is that there are limitations inherent in current technology. Within computer hardware electrical current is used to represent data and so we only have two possible states for that current: present and not present. This restriction forces computers to use binary, or base 2, numeric systems to represent all data. In other words, all data must be represented by a collection of either 1’s or 0’s. These collections of 1’s and 0’s, individually referred to as bits, are commonly grouped together into 8 bit collections called a byte. However for our purposes using 8 bits is excessive and so we will use collections of 4 bits for our representations.

It is also desirable that our representation system be fast. Since everything in a computer must be broken down to collections of bits, we must have an algorithm that is as simple as possible to implement within the system so that it can be executed in a minimal amount of time. Then, since all data is a collection of bits, computers must group those bits together in a structured way in order to know where one collection begins and another collection ends. To that end a computer will always store leading 0’s of a given bit collection because in reality even though bits are numbers, a collection of bits is NOT a number.
Finally the most important limitation of a computer compared to human beings is something that we take for granted; our ability to understand what symbols mean. The best example of this is that we know that a “–” in front of a number means that the number is negative or less than zero. In other words we know that it means the number \(-x\) is \(x\) number of units less than 0. Furthermore we know that \(-4\) is greater than \(-5\), because \(-4\) is only 4 units less than zero but \(-5\) is one more unit less than zero. However 4 is not greater than 5, because 4 is only 4 units greater than zero, while 5 is 5 units greater than 0. Additionally when we see \(-5 + 4\) we know that the \(-5\) will become greater by 4 units for the sum of \(-1\). But when we see \(-5 + -4\) we know that the \(-5\) will become less by 4 units for the sum of \(-9\). These simple mental steps are done in the blink of an eye because we understand the implication of the “–” symbol, however it is not possible for a computer to come to these same conclusions because it simply cannot process the “–” symbol. It can only process bits.

Given the limitations of computers, now look at what our ideal system involves. First, the system should represent as many numbers as possible for the given size of the collection of bits. In other words we should not have a system that has duplicate representations for numbers, if we have 4 bits then we should be able to represent \(2^4 = 16\) distinct base 10 numbers. Second the system must be able to perform its addition algorithm in as few steps as possible, thereby making the process speed as fast as possible. Finally the system should be able to represent negative numbers within the binary computer structured collections of bits.

Given our ideal system, we will look to the current binary systems to find the closest two candidates. These candidates will then be expanded to a ternary, or base 3, world to find which of them is closer to our ideal system in the ternary world. This process will give us an
understanding of whether our binary systems will expand to the ternary based systems or to other bases. This is important to know because while we still live in a world with only binary computers, the world is ever changing and someday soon we may very well have that base 3 computer.
II Binary – Signed Bit

The first system studied is the Signed Bit representation system. Signed Bit attempts to represent positive and negative numbers using sets of bits within the computer. While there were some major issues and limitations, it was an important system to study because it helped to better illustrate and highlight the limitations of computers.

The first thing studied was the conversion process for converting base 10 numbers to their binary representation. This conversion process attempts to mimic the way positive and negative numbers are represented in the real world. To accomplish this, the left-most bit is set aside as a sign bit. Therefore when this bit is a 0, the number represented is positive, and when it is a 1 the number represented is negative. Given that the left-most bit is set aside, there are 3 remaining bits used to determine the magnitude of the number representation.

After defining the conversion process it is time to look at a few examples. First, take $1_{10}$, it is positive so our sign bit will be a 0, and the magnitude in binary is 001, so the Signed Bit representation is $0001_2$. Next look at $2_{10}$, which again is positive so 0, and 010 for the binary magnitude, so the representation is $0010_2$. Finally look at $4_{10}$ and $7_{10}$, which are both positive so a 0 sign bit for each of them, and binary magnitudes 100, and 111 respectively. Therefore $4_{10}$ becomes $0100_2$ and $7_{10}$ becomes $0111_2$ in Signed Bit representations.

Looking at examples of the conversion process for negative numbers is next. All of these examples will be negative so the sign bit will be a 1 for each of them. Then, $-1_{10}$ has magnitude 001, just as positive $1_{10}$ had the same magnitude. Therefore it follows that the Signed Bit representation for $-1_{10}$ is $1001_2$. Next $-2_{10}$ has magnitude 010, and so is represented
by 1010₂. Finally, -4₁₀ and -7₁₀ have magnitudes 100 and 111 respectively and are thus represented by 1100₂ and 1111₂ respectively.

At this time, it is important to observe that a positive and negative representation for each number is only separated by the Signed Bit. This perfectly mimics the real world number representations of numbers because the only difference between 5₁₀ and -5₁₀ is the presence of the negative sign which dictates positivity and negativity.

Finally, look at the representation for 0₁₀. Zero is not considered positive or negative, but since binary is a base 2 system, we must put either a 1 or a 0 in the signed bit position. Thus it was chosen to consider zero positive, for these purposes. Then the Signed Bit representation for 0₁₀ is sign bit 0, with magnitude 000, so together 0000₂. However, while it was decided to consider 0₁₀ positive, if 0₁₀ had been considered negative, a sign bit of 1 and 000 magnitude, so together 1000₂, would have been generated which is in fact a valid representation. This leads to a revelation that there are in fact two representations for 0₁₀: a positive zero, 0000₂, and a negative zero, 1000₂. Therefore this system has at least one major flaw in that it fails to represent as many numbers as possible.

Next the addition algorithm associated with Signed Bit representations was studied. Given that representations are designed after the real world equivalent numbers, the addition algorithm is defined to operate in the same fashion. For some examples, first look at 1₁₀ + 3₁₀ = 4₁₀, in Signed Bit 0001₂ + 0011₂, and performing addition to get 0100₂, which is the representation for 4₁₀. Next, look at 2₁₀ + 5₁₀ = 7₁₀, and 0010₂ + 0101₂ = 0111₂, and since 0111₂ is the representation for 7₁₀, once again the correct answer is generated. In fact for the addition of all positive numbers, whose sum is less than or equal to 7₁₀, this system will work. If
the sum of the two numbers is greater than 7₁₀, there is an issue because the sum cannot be represented with just 3 bits. This type of issue is called an overflow problem.

There are two possible solutions to this issue. First more bits can be added to account for the overflow, but it was already established that computers must have a structured way of organizing bits, so more bits cannot be added to a representation on the fly. The second solution doesn’t necessarily resolve the problem, but it does prevent the issue. This solution involves testing for the overflow and thus allowing the operation to be blocked or flagged if the operation would overflow. With the Signed Bit addition algorithm we can detect overflow by checking the carry out from the highest order magnitude bit, which in this instance is the 3^{rd} order bit. If this bit addition has a carry out, then the addition can be flagged as invalid.

Thus with the check for overflow, the addition algorithm is worked out for positive numbers, but what if negative numbers are involved? For example, what happens when we add 1₁₀ and -1₁₀. Well, since 1₁₀ is 0001₂, and -1₁₀ is 1001₂, adding the representations together becomes 1010₁₀, with no carry on the 3^{rd} order bits so no overflow. But is 1010₂ the correct answer? No, because 1010₂ is the representation for -2₁₀ and 1₁₀ + -1₁₀ should result in 0₁₀. Where did the algorithm go wrong? Well the issue is really that when humans see a negative number in the addition process we know to subtract it from the positive number, if a positive number is present. But a computer doesn’t inherently know this, so add additional rules must be added. These rules state that if two numbers are being added and one of them is negative, subtract the magnitude of the negative number from the magnitude of the positive number. Then, we have 1₁₀ + -1₁₀ represented as 0001₂ + 1001₂ becoming 0001₂ − 1001₂ which results in 1000₂, which is a representation for 0₁₀ and so this situation is now resolved.
But what if we try to add $1_{10}$ represented as $0001_2$ and $-2_{10}$ represented as $1010_2$. Subtracting $2_{10}$ from $1_{10}$, requires the concept of negative numbers since we are subtracting a greater magnitude number from a smaller magnitude number. However, we know that computers have no concepts of negative numbers, and thus subtracting $2_{10}$ from $1_{10}$ will not be easily programmable. Instead, additional rules are added to alter the addition algorithm so that when positive and negative numbers are added the two magnitudes are compared and the magnitude of the smaller number is subtracted from the magnitude of the greater number, then the sign bit is set to the value of the number with greater initial magnitude. Then it can be seen that taking $010_2 - 001_2 = 001_2$, and so the sign bit is set to 1, since the $-2_{10}$ had a greater magnitude, resulting in $1001_2$ which is $-1_{10}$, the desired result.

Now for the addition of 2 negative numbers, simply add the magnitudes of the numbers together and leave the sign bit set to 1. So $-1_{10} + -3_{10} = -4_{10}$ becomes $1001_2 + 1011_2 = 1100_2$, which is representing $-4_{10}$, and so this system works. In fact this system is quite similar to the system used to add positive numbers. The similarity is due to the fact that in both the cases we want to just add the magnitudes and leave the sign bits set to the original value. Additionally in both cases we know that we have overflow if the 3\textsuperscript{rd} bit carries over. Therefore, adding two positive or two negative numbers, can be covered by the same operation.

To conclude the Signed Bit system is a system which represents any negative or positive number, but there are several major drawbacks to this system. First, there are 2 zeros, so there are not as many representations as possible. Second, the addition algorithm is complex involving several steps to determine whether to add, or subtract, the numbers together and what sign bit to pass through to the result. Furthermore it then becomes obvious that a second
operation, subtraction, has been defined. Therefore, this system is a nightmare when it comes to actually implementing it within the computer. However, this study was not a waste, because it lead to a greater and deeper understanding of the limitations of computers which helped to frame the study of the other binary based systems.
II Binary – One’s Complement

One’s Complement was the next system of binary representation of base 10 numbers studied. This system does in fact alleviate one of the two key issues present in the Signed Bit system, the complexity of the addition algorithm. The complexity is alleviated as a result of the way the system generates representations of negative numbers, and so this study will begin with a look at that process.

One’s Complement represents positive numbers in their normal binary equivalent. In other words, $1_{10}$ is $0001_2$ and $4_{10}$ is $0100_2$. Once again, because only 4 bits are being used to represent numbers, there are only 16 total numbers represented. Thus only half of these representations are positive and so 7 is the greatest positive number, because 0 is considered positive for these purposes. Then our positive numbers $0_{10}$ through $7_{10}$ are defined to be $0000_2$ through $0111_2$ (a complete listing can be found in the One’s Complement Cayley Table in the Appendix).

To generate negative numbers a process of “flipping” of the bits of the corresponding positive number is used. Flipping the bits, gets its name from the process used to generate the replacement bit. Essentially take a list of the available digits, so (0, 1) in this case, and flip the list around to get [1, 0]. Then the bits in the normal list are replaced by the corresponding bit in the flipped list. So (0) becomes [1], and (1) becomes [0]. For example, to determine the One’s representation of the $-1_{10}$, first take $1_{10}$ and then generate the corresponding One’s binary representation of $0001_2$. Next, flip the bits of positive one, so the bits of $0001_2$ become $1110_2$. Therefore the One’s representation of $-1_{10}$ is $1110_2$. 
Next, look at a few more examples of representations of negative numbers in One’s Complement. First for \(-2_{10}\), generate \(2_{10}\) in One’s which is \(0010_2\), so flipping the bits results in \(1101_2\). Thus the One’s representation of \(-2_{10}\) is \(1101_2\). Next, for \(-4_{10}\) we can see that \(4_{10}\) is \(0100_2\), and so flipping the bits becomes \(1011_2\). Finally \(-7_{10}\), we know \(7_{10}\) is \(0111_2\), and so by flipping it becomes \(1000_2\). Hence the One’s Complement binary representation of \(-2_{10}\) is \(1101_2\), \(-4_{10}\) is \(1011_2\), and \(-7_{10}\) is \(1000_2\).

There is one last negative number representation to address. Negative numbers in One’s Complement are generated by flipping the bits of the positive corresponding number, so what happens if we flip the bits of zero? Since \(0000_2\) is considered a positive number, flipping its bits results in \(1111_2\) which is the One’s binary representation of \(-0\). This representation confirms the existence of a negative zero within the One’s Complement system. Therefore one issue from the Signed Bit system still remains.

The addition algorithm in One’s Complement is straightforward. Simply add the two desired numbers to generate their sum, no need to check for positive or negative before performing the operation. Additionally, if there is a carry from the left-most (highest order bit) column, it is added back into the number in the right-most (lowest order bit) column. There is also a way to test for overflow. This test is performed by comparing the carry out from the two left-most columns. If they are the same it is a valid representation, if they are different, overflow has occurred and it is an invalid representation.

Thus, the addition algorithm has been fully defined and it time for some examples. First, look at adding \(1_{10}\) and \(2_{10}\), so \(0001_2 + 0010_2\) becomes \(0011_2\) which is \(3_{10}\) and the left most carries are both 0 so there is nothing to add and the result is valid. Next look at \(3_{10} + 4_{10}\), which
is $0011_2 + 0100_2 = 0111_2$ and once again there are zeros in the left most column’s carry out, so there is nothing to add and a valid result of $0111_2$, which represents $7_{10}$, is generated. Finally, look at $4_{10} + 4_{10}$, $0100_2 + 0100_2$, which results in $1000_2$. However, there is a carry of 1 from the third column and zero from the last column, and so there is nothing to add, but the result is invalid. The invalidity of the result is verified because $4_{10} + 4_{10} = 8_{10}$, but there is no representation for $8_{10}$ in One’s Complement.

Moving this algorithm to the negative world it follows that $-1_{10} + -2_{10}$ is $1110_2 + 1101_2 = 1011_2$, however there were carries of 1 in both the third and fourth columns. Thus a valid result is generated, and the 1 must be added back into the first column. Therefore $1011_2$ becomes $1100_2$, which is the representation for $-3_{10}$, the desired result. Next, add $-5_{10} + -4_{10}$, which is $1010_2 + 1011_2 = 0101_2$, with a carry of 0 from the third column and 1 from the fourth, which indicates the result is invalid. It can be verified that the result should be invalid again because there is no representation for $-9_{10}$.

Finally, the addition algorithm must be tested with the zeros. First, adding $0000_2$ to any other representation will not change any bits in that representation, so adding positive zero is verified as valid. To test negative zero, it is added to $1_2$, $7_2$, $-1_2$, and $-7_2$. Therefore $1111_2 + 0001_2 = 0000_2$ with a carry out of 1 in every column. Thus we need to add one to the right-most column, results in $0001_2$ which is $1_{10}$. Next, add negative zero to $7_{10}$, so $1111_2 + 0111_2 = 0110_2$ with carries of 1 in the third and fourth column, thus it is valid. After adding the one back in the result is $0111_2$, which is $7_{10}$. Then, looking at $-1_{10}$ and $-7_{10}$ respectively, so $1111_2 + 1110_2 = 1110_2$ and $1111_2 + 1000_2 = 1000_2$ after applying the algorithm, with both results valid. Hence, it has been verified that adding positive and negative zero provides valid results.
Therefore, One’s Complement is a system in which there are representations for positive and negative numbers, with a straightforward addition algorithm. Furthermore, the algorithm has a built in test for the validity of the produced sums. Thus, given the progress we have made from the Signed Bit system, another question is left to answer: is One’s Complement a group under addition?

First One’s cannot be a group with both zero’s since there cannot be two zero’s under the same operation in the same group. Thus the positive zero is removed. This decision is made by looking at the One’s Complement Cayley table in the Appendix. After scanning through the table with both zero’s it is apparent that positive zero can be removed without affecting any other rows or columns. Looking at the Cayley table with positive 0 removed, it follows that One’s Complement is an Abelian group under addition.

One’s Complement is an Abelian group with 15 elements, and so it must be isomorphic to $\mathbb{Z}_{15}$. Therefore the last task is to determine the isomorphism. To determine that begin by looking at the conversion from base 10 numbers to their representation in One’s Complement binary. First note that positive numbers from $1_{10}$ through $7_{10}$ are simply converted to their binary equivalent and so the function can just do that conversion. For the negative numbers, the conversion has an additional step. Since $0_{10}$ becomes $1111_2$ has magnitude value 15 and $-1_{10}$ becomes $1110_2$, which has magnitude value 14, and so on until $-7_{10}$ which becomes $1000_2$ which has magnitude value 8. From this it becomes apparent that to convert numbers $-7_{10}$ through $-0_{10}$, add 15 then apply the conversion to binary. One last detail to note, which can be seen in the Cayley table, is that addition wraps around just as in $\mathbb{Z}_{15}$ addition and so the conversion can mod the base 10 inputs by 15 to get the desired range of number $-7_{10}$ through
Thus the isomorphism from base 10 to base 2 is:

\[
f(x_{10}) = \begin{cases} 
(x_{10})_2, & 1 \leq (x_{10} \mod 15) \leq 7 \\
(x_{10} + 15)_2, & -7 \leq (x_{10} \mod 15) \leq 0 
\end{cases}
\]
Binary – Two’s Complement

The next system studied is Two’s Complement. This system is immensely similar to One’s Complement but improves upon it by eliminating the second representation for zero. Two’s Complement is then a system that has eliminated both major issues from Signed Bit, the algorithm complications and the negative zero.

To eliminate the negative zero, a step is added to the generation of negative numbers. This step is the addition of 1 to the negative representation of the positive number. To better illustrate this look at the following example: -1₁₀. To get -1₁₀ in Two’s Complement follow the same steps from One’s Complement and get 1110₂, then add 1 to that negative representation, and so 1110₂ becomes 1111₂. Thus -1₁₀ in Two’s Complement is represented by 1111₂.

Now to explore the conversion process further. First, -2₁₀ becomes 1101₂ in One’s and so it becomes 1110₂ in Two’s. Next, -4₁₀ becomes 1100₂ in Two’s, and -7₁₀ becomes 1001₂ in Two’s Complement. It then becomes clear a pattern has emerged. 1111₂ is -1₁₀, 1110₂ is -2₁₀, and so on until 1001₂ is -7₁₀. Then according to the pattern 1000₂ is representing -8₁₀. However, there is no positive 8₁₀ in this system. This situation is a product of the fact that the base is even and so there will be an even number of representations. Therefore, with 0 being represented, we cannot have an equal number of positive and negative numbers, however since 0 is considered a positive number then there are 8 positive representations and so there need to be 8 negative representations. Thus -8₁₀, represented by 1000₂ is a valid representation in Two’s Complement.

Finally, take a closer look at zero. It follows that 0₁₀ is represented by 0000₂, but what happens when it is converted to a negative representation. First, flip all the bits and get 1111₂,
then add one and get 0000₂ with a carry out of the fourth column. The carry is ignored and so there is only one zero represented in Two’s Complement. Thus Two’s Complement is a system which represents -8₁₀ through 7₁₀ in binary with a single unique representation for each number.

Now, addition algorithm is found to be virtually identical to the One’s Complement algorithm as well, with one minor tweak; do not add the carry from the fourth column back into the first column. To illustrate the system add 1₁₀ + 2₁₀, which is 0001₂ + 0010₂ = 0011₂ the representation for 3₁₀. Next add 5₁₀ + 2₁₀ and so 0101₂ + 0010₂ = 0111₂ the representation for 7₁₀. In both of these tests the carries in the third and fourth columns have been zeros and so we know that our results are valid. However, now try 6₁₀ + 2₁₀, and so 0110₂ + 0010₂ = 1000₂ with a carry of 0 in the fourth and 1 in the third columns. Thus the result of 1000₂, representing -8₁₀, is invalid.

It has then been shown that the Two’s Complement addition algorithm works for positive numbers, so now it is time to test on representations for negative numbers. Begin with -1₁₀ + -4₁₀, and so 1111₂ + 1100₂ = 1011₂, with carries of 1 in both the third and fourth columns. Thus the result of 1011₂, representing -5₁₀, is valid. Next try adding -5₁₀ + -3₁₀, so 1011₂ + 1101₂ = 1000₂, with carries of 1 in both the third and fourth columns again, so the result is valid. Finally try -5₁₀ + -4₁₀, which is 1011₂ + 1100₂ = 0111₂, with carries of 1 in the third column and 0 in the fourth. Therefore the result of 0111₂, representing 7₁₀, is invalid.

Therefore Two’s Complement is a system for representing -8₁₀ through 7₁₀ with a straightforward addition algorithm. Additionally looking at the Cayley Table in the appendix, shows that Two’s Complement is an Abelian group of 16 elements., which is isomorphic to $\mathbb{Z}_{16}$. 

17
Since positive representations are simply the binary equivalents like in One’s Complement, with the addition of $0_{10}$ also being represented by its literal base 2 equivalent, the One’s Complement isomorphism can be mimicked for the positive portion of this isomorphism. Next, take a look at representations for negative numbers. So $-1_{10}$ becomes $1111_{2}$ which is $15_{10}$, then $-2_{10}$ becomes $1110_{2}$, which is $14_{10}$, and so on until $-8_{10}$ becomes $1000_{2}$ which is $8_{10}$. It is then apparent that that a similar process to One’s can be implemented, except adding 16 instead of 15. Since there are 16 elements the isomorphism should mod by 16 as opposed to 15. Thus, the isomorphism is: 

$$f(x_{10}) = \begin{cases} 
(x_{10})_{2} & , \ 0 \leq (x_{10} \ % \ 16) \leq 7 \\
(x_{10} + 16)_{2} & , \ -8 \leq (x_{10} \ % \ 16) \leq -1 
\end{cases}$$
II Binary – Excess $2^{n-1}$

The final binary system is Excess $2^{n-1}$, or Excess for short. This system is developed in an entirely different fashion than Signed Bit, One’s Complement, or Two’s Complement, but it retains the advantages from Two’s Complement of having a simple addition algorithm, and only one zero. However, it does maintain one advantage over Two’s Complement. It has a more straightforward isomorphic function, and as a result is easier to read in its binary form.

To generate numbers in the Excess system begin by taking note of the fact that there 16 representations. This indicates that there should be 8 positive numbers and 8 negative numbers. So the base 10 numbers could be shifted up by 8 such that they are then all positive values to directly convert to binary numbers. So for example to convert $-8_{10}$ to Excess binary add 8 to the base ten number and get $0_{10}$ then convert to binary and so $-8_{10}$ is represented by $0000_2$. Next look at $-1_{10}$ and so $7_{10}$ after adding 8, which becomes $0111_2$. Further, $0_{10}$ becomes $8_{10}$ then correspondingly $1000_2$, $1_{10}$ becomes $9_{10}$ then $1001_2$, and finally $7_{10}$ becomes $15_{10}$ then $1111_2$. Therefore it is apparent that the numbers $-8_{10}$ through $7_{10}$ can be represented with 4 bit binary representations, with a unique representation for each.

Next look at the addition algorithm, which once again involves adding the two numbers, with one small exception. After adding the numbers, the highest order (leftmost) bit is flipped. Additionally, just as in One’s and Two’s there is a check for the validity of the result. However, this check involves comparing the highest order bit, after it is flipped, with the carry out from that same column. If those two digits are the same, the result is valid, and if they are different result is invalid.
Now to test the addition algorithm, first add \( 1_{10} + 6_{10} \), which \( 1001_2 + 1110_2 = 0111_2 \), then flip the lead bit and get \( 1111_2 \). So 1 is the highest order bit and there was a 1 in the carry from the fourth column. Thus the result is valid, which can be verified since \( 1111_2 \) represents \( 7_{10} \).

Next add \( 3_{10} + 5_{10} \), so \( 1011_2 + 1101_2 = 1000_2 \), and after flipping it is \( 0000_2 \), however this is an invalid result because the lead bit is 0 and the carry from the fourth column was 1. To verify, look at \( 3_{10} + 5_{10} = 8_{10} \) which is not in the representation range. Now add \( -3_{10} + -4_{10} \), so \( 0101_2 + 0100_2 = 1001_2 \), then \( 0001_2 \) after flipping which is valid because the lead bit is 0 and the carry was 0. Finally check \( -4_{10} + -6_{10} \), which is \( 0100_2 + 0010_2 = 0110_2 \), so \( 1110_2 \), however this is invalid because the lead bit is 1 and the carry out was 0.

Now that the algorithm has been tested, it can be seen that the Excess system is a group under addition. Once again this can be seen from looking at the Cayley table. Furthermore, it becomes apparent that this group is \( Z_{16} \), because the numbers represented are shifted up to the representation range of 0 through 15 by adding 8 in the conversion from base 10 to binary. Finally there is one final question to consider, what is the isomorphism? In truth this is apparent from the conversion algorithm, so the isomorphism from base 10 to binary is:

\[
f(x_{10}) = (x_{10} + 8)_{2}, \quad -8 \leq (x_{10} \ % \ 16) \leq 7
\]
III Ternary – Complement

The first Ternary system studied is an abstraction of Two’s Complement, however it will be called Complement for the sake of clarity. So in the binary world the defined Complement systems were built upon the idea of defining the positive numbers to be their binary equivalent then defining the negative numbers to be the complement of the positive numbers. Thus in the ternary world the representations will be defined the same way.

To begin take note that we will be using three trits and so we have $3^3 = 27$ representations to utilize. So there will be 13 representations for positive numbers, 13 for negative numbers, and one for 0. First $0_{10}$ is represented by $000_3$, and then $1_{10}$ is represented by $001_3$. Next it follows that $2_{10}$ is $002_3$, $3_{10}$ is $010_3$, and so on until $13_{10}$ is $111_3$. Thus the representations for positive numbers and our 0 have been defined.

Now, take a look at the negative number representations. To determine the pairings for flipping the trits, note that the list of digits is (0,1,2) and when flipped that list becomes [2,1,0], and so individually it follows that: 0 becomes 2, 1 remains 1, and 2 becomes 0. Therefore representations for negative numbers can now be built. First generate $-1_{10}$, by taking $1_{10}$ and getting $001_3$, then flipping the trits results in $221_3$, and finally adding 1 to get $222_3$. So the representation for $-1_{10}$ is $222_3$.

Now look at a few other representations. First, $-2_{10}$, so 2 is $002_3$, which becomes $220_3$, and finally adding 1 results in $221_3$. Next, $-3_{10}$, so $3_{10}$ is $010_3$, which becomes $212_3$, and finally adding one results in $220_3$. Third, $-9_{10}$ so $9_{10}$ is $100_3$, which becomes $122_3$, and finally $200_3$. Finally, $-13_{10}$, so $13_{10}$ is $111_3$ which becomes $111_3$, and finally $112_3$. (a complete listing can be found in the Two’s Complement Cayley Table in the Appendix)
Next look at the addition algorithm of our ternary Complement system. In fact as it turns out the algorithm is the same algorithm as Two’s Complement binary. To illustrate this look to a few examples: \(1_{10} + 2_{10}\), is \(001_3 + 002_3 = 010_3\) which is \(3_{10}\) the desired result. Then: \(6_{10} + 7_{10}\), so \(020 + 021_3 = 111_3\) which is \(13_{10}\), once again the desired result. Now for some negative examples, first, \(-1_{10} + -2_{10}\), so \(222_3 + 221_3 = 120_3\), which represents \(-3_{10}\). Finally, take a look at \(-13_{10} + 0_{10}\), so \(112_3 + 000_3 = 112_3\), the representation for \(-13_{10}\). Therefore it can be seen that the algorithm holds for valid additions.

Given that the algorithm holds for valid additions, what happens with invalid additions? First try adding \(10_{10} + 4_{10}\), so \(101_3 + 011_3 = 112_3\), which is the representation for \(-13_{10}\), not \(14_{10}\) the desired result. However, there were carry outs of 0 from all columns. Thus checking carry outs to determine the validity of the result cannot be used, since adding \(10_{10} + 3_{10}\), so \(101_3 + 010_3 = 111_3\) the desired representation for \(13_{10}\) also had carry outs of 0 in all columns. In fact the only way to check the validity of the addition results in ternary based Complement is to determine the signs of numbers before performing the addition and deciding what the sign should be after addition. Then verifying that, that post addition sign is valid. This process is not efficient so in the ternary complement system the validity checking has been lost, while the rest of the algorithm and representation process remains valid.

Once again taking a look at the Cayley confirms that ternary Complement is a group under addition. Furthermore, it can be seen that this group is isomorphic to \(Z_{27}\). To determine the isomorphism, look at the Two’s Complement isomorphism and adapt it to this situation, and so it follows that: \(f(x_{10}) = \begin{cases} \ (x_{10})_3, & 0 \leq (x_{10} \mod 27) \leq 13 \\ (x_{10} + 27)_3, & -13 \leq (x_{10} \mod 27) \leq -1 \end{cases}\)
III Ternary – Excess

Finally, this study looked at the Excess system abstracted into the ternary world. This system is built upon sliding the base 10 numbers to create positive base 3 representations. So to follow the process from binary simply shift the numbers by half of the total representations possible. Thus the Excess system will represent $-13_{10}$ through $13_{10}$ using three trits.

First find the representation for $0_{10}$, so adding 13 to $0_{10}$ results in $13_{10}$ which is $111_3$. Next, to find the representations for $1_{10}$, $2_{10}$, $3_{10}$, $9_{10}$, and $13_{10}$, that process will be repeated. Then $1_{10} + 13 = 14_{10}$ which is $112_3$, and so $1_{10}$ is represented by $112_3$. Next we will look at $2_{10}$, which is $15_{10}$ after adding 13. Then $15_{10}$ is $120_3$ and so $2_{10}$ is represented by $120_3$. It follows that $3_{10} + 13 = 16_{10}$, and so $16_{10}$ is $121_3$, which means $3_{10}$ is represented by $121_3$. Finally $9_{10}$ and $13_{10}$ are $22_3$ and $26_{10}$ after adding 13 to each of them. Thus $9_{10}$ and $13_{10}$ are represented by $211_3$ and $222_3$ respectively.

Next to take a look at the representations of negative numbers. We will generate $-1_{10}$, $-3_{10}$, $-9_{10}$, and $-13_{10}$. First $-1_{10}$, which becomes $12_{10}$ after adding 13 and so $12_{10}$ is $110_3$, thus $-1_{10}$ is represented by $110_3$. Next, $-3_{10}$ becomes $10_{10}$ after adding 13 which is $101_3$, so $-3_{10}$ is represented by $101_3$. Finally $-9_{10}$ and $-13_{10}$ become $3_{10}$ and $0_{10}$ respectively, and thus $-9_{10}$ is represented by $010_3$ and $-13_{10}$ is represented by $000_3$.

After completing the analysis of the conversion process, the research to an investigation of the addition algorithm, but this time the algorithm cannot be based on the binary Excess equivalent. For example try adding $1_{10} + 2_{10} = 3_{10}$, so it follows that $112_3 + 120_3 = 002_3$ and then flipping the leading trit and results in $202_3$. However, $202_3$ is the representation for $7_{10}$, but the
desired result is $121_3$, the representation for $3_{10}$. Therefore, the addition algorithm must be redefined.

To determine what the issue is the most simplistic example of addition is used, $0_{10} + 0_{10}$. In Excess then it is $111_3 + 111_3$, and the desired result is $111_3$. So, adding $111_3 + 111_3$ results in $222_3$. Attempting to flip the lead trit once again causes issues because there is an erroneous result of $022_3$. Taking a different approach shows that to get $111_3$ from $222_3$ we could subtract $111_3$, however this would involve defining the subtraction algorithm, which is an undesirable situation. It also follows that adding $112_3$ to $222_3$ results in $111_3$. Therefore we have found an addition that will work in this specific instance. So now to expand the algorithm, take a look at $1_{10} + 2_{10}$ again and see if the new modification still holds. First, $112_3 + 120_3 = 002_3$, and so adding $112_3$ we get $121_3$, the desired result. Thus, this addition algorithm defined by adding the operands, then adding $112_3$ has worked in our limited testing, so we will expand that testing.

To expand first, try $6_{10} + 7_{10}$, so we have $201_3 + 202_3 = 110_3$. Then add $112_3$ and get $222_3$ which is the representation for $13_{10}$ the desired result. Next add $-13_{10} + 13_{10}$, so we have $000_3 + 222_3 = 222_3$ and adding $112_3$ results in $111_3$ which represents $0$, once again the desired result. Next try $-1_3 + -2_3$ and so $110_3 + 102_3 = 212_3$ and then adding $112_3$ results in $101_3$, which represents $-3_{10}$. Finally adding $-1_{10} + -12_{10}$, so $110_3 + 001_3 = 111_3 + 112_3 = 000_3$, the representation for $-13_{10}$.

Then given that the ternary Excess addition algorithm has been defined, it is important to determine why it is so different from the binary Excess binary algorithm. Looking closer at the binary excess algorithm it can be seen that flipping the left-most bit is equivalent to adding $1000_2$. This is important because adding $1000_2$ to the binary Excess zero representation, which
is $1000_2$ results in $0000_2$. Then in ternary adding $112_3$ to $111_3$, the ternary Excess representation for $0_{10}$, results in $000_3$. Therefore, it would appear that the two systems are accomplishing the same process in both binary and ternary, and that process is equalizing for the shifted $0_{10}$ representation.

Next it can be seen that the ternary based addition algorithm has an efficient error checking system built into it like the binary systems. Before the test of our ternary additions take a closer look at the binary validity check after the revelation that the true purpose of flipping the leading bit is to account for the shifted zero. When adding $1001_2 + 1110_2$ it was valid because $1001_2 + 1110_2 = 0111_2$ with a carry of 1 from the left most column. Then after accounting for the shifted zero by adding $1000_2$, so $0111_2 + 1000_2 = 1111_2$, which we previously said was valid because the carry of 1 matched the highest order bit, 1, in the final result of $1111_2$. However the carry in the left most column from adding $1000_2$ in the second step is 0, so perhaps there is a valid result when the two carries are different. To verify try adding $1001_2 + 1111_2 = 1000_2$ and a carry of 1. Then adding $1000_2$, so $1000_2 + 1000_2 = 0000_2$, an invalid result, with a carry of 1, which should indeed be an invalid result because both carries were the same. Therefore the validity check can be rewritten as: the result is valid if the carries from the leftmost columns in the normal addition step and the second addition to account for the shifted zero step, are different.

Given the clarification of the addition validation check in the binary Excess system, now check if it works for the ternary Excess system. First add $112_3$ the representation for $1_{10}$, and $222_3$ the representation for $13_{10}$, which will sum to an invalid result. So $112_3 + 222_3 = 1113_3$ with a carry of 1, and then adding $112_3$ to account for the shifted zero, $1113_3 + 112_3 = 0000_3$, with a
carry of 1, which means there is an invalid result because both carries were 1’s. Next try adding $001_3 + 110_3 = 111_3$ with a carry of 0, and then to correct for the zero, so $111_3 + 112_3 = 000_3$ with a carry of 1, and thus it is a valid result since the two carries were different. Thus it follows that the ternary Excess system has an efficient check for validity of the addition algorithm.

Now the addition algorithm has been determined, it can be seen in the Cayley Table that the ternary Excess system is an Abelian group under that algorithm. Finally, like in binary Excess the isomorphism becomes apparent from the process used to generate the ternary representations, which is: $f(x_{10}) = (x_{10} + 13)_3$, $-13 \leq (x_{10} \% 27) \leq 13$
IV Conclusions

Now that we have completed the study of the representation systems we need to determine which system is the most conducive for expansion to base 3 and beyond. From looking at binary we know that the only two viable candidates are the Two’s Complement and Excess $2^{n-1}$ system. They both maximized their representation of numbers and they had the fewest steps inherent in their addition algorithm. Additionally as a bonus feature both were able to perform a validity check on the sums of two numbers. In the binary world both systems were approximately equivalent in their advantages and disadvantages, but when we expanded to the ternary world, the answer was not as straightforward.

The issue is that in the ternary world each system had a clear advantage over the other system. Complement has the cleaner addition algorithm because there is no correction for zero, but the Excess system had the ability to efficiently check the validity of its addition algorithm’s results. Furthermore from a readability standpoint they each have their advantages and disadvantages. Complement provides positive representations that are linked directly to the number they represent, but the Excess system places all numbers in order. Both of these advantages can be seen from the isomorphisms associated with each system. The truth is that from an approachability perspective there is again no clear favorite, it just depends on the designers preferences. This leads us to another analysis factor, and that is flexibility.

While flexibility was not one of our original criteria, it is nonetheless something we can address now. The truth is that what makes the Excess system so approachable also makes flexible. What we mean by flexible is that the system does not have to shift by half of the available numbers. Instead of shifting by 13 in the ternary setting we could instead shift by 9
and then have 100₃ be defined as our 0₁₀. Thus we would only need to add 200₃ to account for our shifted zero. Then the adjustment step for the shifted zero step would be just as simple to implement as it was in binary. Furthermore the advantages we would gain from the ability to shift by a variable amount are important. Sometimes in a computer science situation we only need a handful of negative numbers, but we would like to maximize our positive numbers. This system would allow for that situation. This system would allow for a more flexible implementation based upon the specific situation the designers found themselves in and so flexibility an advantage that cannot be ignored.

Therefore when looking at all these factors to determine the recommendation for expanding to base 3, we can see that the recommendation depends upon the use case. If the designers want speed then they should utilize the Complement system, but if they want flexibility or validity checking they should instead look to the Excess system.
## V Appendix – Cayley Tables

### Binary- One’s Complement

#### Addition

<table>
<thead>
<tr>
<th>base 10</th>
<th>-7</th>
<th>-6</th>
<th>-5</th>
<th>-4</th>
<th>-3</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ones</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
</tr>
<tr>
<td>1001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td>0001</td>
</tr>
<tr>
<td>1010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td>0001</td>
<td>0010</td>
</tr>
<tr>
<td>1011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td>0001</td>
<td>0010</td>
<td>0100</td>
</tr>
<tr>
<td>1100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td>0001</td>
<td>0010</td>
<td>0100</td>
<td>0101</td>
</tr>
<tr>
<td>1101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td>0001</td>
<td>0010</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
</tr>
<tr>
<td>1110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td>0001</td>
<td>0010</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>1111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td>0001</td>
<td>0010</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
</tr>
</tbody>
</table>

### Binary- One’s Complement (Group)

#### Addition

<table>
<thead>
<tr>
<th>base 10</th>
<th>-7</th>
<th>-6</th>
<th>-5</th>
<th>-4</th>
<th>-3</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ones</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
</tr>
<tr>
<td>1001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td>0001</td>
</tr>
<tr>
<td>1010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td>0001</td>
<td>0010</td>
</tr>
<tr>
<td>1011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td>0001</td>
<td>0010</td>
<td>0100</td>
</tr>
<tr>
<td>1100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td>0001</td>
<td>0010</td>
<td>0100</td>
<td>0101</td>
</tr>
<tr>
<td>1101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td>0001</td>
<td>0010</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
</tr>
<tr>
<td>1110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td>0001</td>
<td>0010</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>1111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td>0001</td>
<td>0010</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
</tr>
</tbody>
</table>

29
### Binary - Two's Complement

#### Addition

<table>
<thead>
<tr>
<th>base 10</th>
<th>-8</th>
<th>-7</th>
<th>-6</th>
<th>-5</th>
<th>-4</th>
<th>-3</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twos</td>
<td>0000</td>
<td>1000</td>
<td>0001</td>
<td>0010</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Binary - Excess 2^{2-1}

#### Addition

<table>
<thead>
<tr>
<th>base 10</th>
<th>-8</th>
<th>-7</th>
<th>-6</th>
<th>-5</th>
<th>-4</th>
<th>-3</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excess</td>
<td>0000</td>
<td>1000</td>
<td>0001</td>
<td>0010</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Ternary-Excess

<table>
<thead>
<tr>
<th>base 10 Excess</th>
<th>000</th>
<th>001</th>
<th>002</th>
<th>010</th>
<th>011</th>
<th>012</th>
<th>020</th>
<th>021</th>
<th>022</th>
<th>100</th>
<th>101</th>
<th>102</th>
<th>110</th>
<th>111</th>
<th>112</th>
<th>120</th>
<th>121</th>
<th>122</th>
<th>200</th>
<th>201</th>
<th>202</th>
<th>210</th>
<th>211</th>
<th>212</th>
<th>220</th>
<th>221</th>
<th>222</th>
</tr>
</thead>
<tbody>
<tr>
<td>-13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
VI Bibliography