Lecture 3 In-Class Worksheet

1 Learning Objectives

This worksheet is based on Lumetta course notes section 1.2 and Patt & Patel textbook sections 2.2–2.5. After completing this lesson, you will know how to://

- [S03-1] Determine the number of bits needed to represent a number in unsigned representation.
- [S03-2] Convert between unsigned and decimal representation.
- [S03-3] Add numbers using unsigned representation.
- [S03-4] Determine the number of bits needed to represent a number in two's complement representation.
- [S03-5] Convert between two's complement and decimal representation.
- [S03-6] Sign-extend a number in two's complement representation.
- [S03-7] Negate a number in two's complement representation.
- [S03-8] Add numbers using two's complement representation.
- [S03-9] Subtract numbers in two's complement and unsigned representation.

2 Unsigned Representation

In practice, computers operate on a fixed number of bits grouped together. *Unsigned representation* is a way of representing a range of consecutive non-negative integers starting from zero using a group of bits. With n bits, we can represent 2^n values, which means that using n bits, unsigned representation allows us to represent the integer range $[0, 2^n - 1]$. We call this *n-bit unsigned representation*.

A number represented using n-bit unsigned representation looks the same as the number in binary representation, except that the number of bits is fixed. Numbers that require fewer than n bits in binary representation include leading zeroes so that the representation has exactly n bits.

The formula for the numeric value of the number is the same as for a binary numeral:

Numeric value of a number in n-bit unsigned representation

$$\text{Value} = b_{n-1}2^{n-1} + \cdots + b_1 2^1 + b_0 2^0 \tag{1}$$

The process for converting to and from this representation is the same as when working with binary numerals that we studied in the previous lesson.

Addition of numbers in unsigned representation works the same way as binary number addition. Recall that adding to n-bit binary numbers may produce a $(n + 1)$-bit result.

However, because the unsigned representation works with a fixed number of bits, to get a result that has the same number of bits, we keep only the n least-significant bits. For example, if we add the binary numbers 1101_2 and 0011_2, we get 10000_2. However, this value not representable using 4-bit unsigned representation. Instead, the 4-bit unsigned result is 0000. However, 0000 is the correct result *modulo* 2^n. In fact, *arithmetic using n-bit unsigned representation is arithmetic modulo* 2^n.

3 Two's Complement Representation

Two's complement representation allows us to represent a range of integers around zero. Like unsigned representation, it also used a fixed number of bits, so to fully specify the representation, it is necessary to specify the number of bits. We say that a number is represented using *n-bit two's complement representation* when it is represented using exactly n bits. The range of integers we can represent with n bits is given below.

Integer range of n-bit two's complement representation

$$[-2^{n-1},\ 2^{n-1}-1]$$

The two's complement representations of zero and positive integers are same as their unsigned representation, however, the *number* of non-negative integers we can represent using n bits is half the number we can represent using unsigned representation; the other half of the bit patterns encode negative integers.

> **Q1.** How many bits are needed to represent -100 using two's complement representation? to represent 100? to represent -8? to represent 8?

The smallest n that includes -100 in the range $[-2^{n-1}, 2^{n-1}-1]$ is 8, which gives integer range $[-128, 127]$. This is also the smallest n needed to represent 100. To represent -8, the smallest n is 4, which gives integer range $[-8, 7]$. However, this range does *not* include 8. To represent 8, we need $n = 5$, which gives range $[-16, 15]$.

3.1 Representing Numbers In Two's Complement Representation

Positive integers have the same representation in two's complement representation as in unsigned representation. We will look at two ways of finding the two's complement representation of a negative integer. Given a negative integer, the first way to find its two's complement representation is to add 2^n to the negative integer and use the *unsigned* representation of the result.

The second way to find the two's complement representation of a negative integer is to negate the absolute value of a negative integer, as described in subsection, Negating by Inverting the Bits and Adding One.

> **Q2.** Find the 4-bit two's complement representation of −8 and the 5-bit two's complement representation of 8. Find the 8-bit two's complement representation of 100, −100, 8, and −8. Use the first approach.

To find the 4-bit two's complement representation of −8, we add 2^4 to get 8, and use the unsigned representation of this result, which is `1000`.
The 5-bit two's complement representation of 8 is the same as its unsigned representation (because 8 is not negative), which is `01000`. Note that `1000` is *not* the same as `01000`. The integer 8 cannot be represented using 4-bit two's complement representation, because `1000` is the representation of −8.
The 8-bit two's complement representation of 100 is the same as its unsigned representation: `01100100` $(64+32+4)$. To find the 8-bit two's complement representation of −100, we add $2^8 = 256$ and use the unsigned representation of the result (156) to get: `10011100` $(128+16+8+4)$.
The two's complement representation of 8 is the same as its unsigned representation: `00001000`. Note that this is the same as the 5-bit representation, but with leading zeroes.
To find the 8-bit two's complement representation of −8, we add $2^8 = 256$ and use the unsigned representation of the result (248) to get: `11111000` $(128+64+32+16+8)$. Note that this is the same as the 5-bit representation, but with leading ones. This is called *sign extension*, which we discuss in Section 3.4.

3.2 Negating by Inverting the Bits and Adding One

When starting with a number already in two's complement representation, it is easier to calculate $2^n - x$ as $(2^n - 1) - x + 1$. This is because the binary representation of $2^n - 1$ is just n ones: $2^1 - 1$ is 1_2, $2^2 - 1$ is 11_2, $2^3 - 1$ is 111_2, and so on. Subtracting a number from $2^n - 1$ is easy: we just turn zeroes in the subtrahend to ones, and ones to zeroes. For example:

$$\begin{array}{cccccccc} & 1 & 1 & 1 & 1 & 1 & 1 \\ - & 1 & 0 & 1 & 0 & 1 & 1 \\ \hline & 0 & 1 & 0 & 1 & 0 & 0 \end{array}$$

After we invert the bits (flip ones to zeroes and zeroes to ones), we need to add 1. In the example above, we would have `010101`, which is the two's complement (and unsigned) representation of 21. Using Formula (2), we know that `101011` is the two's complement representation of $-2^5 + 2^3 + 2^1 + 2^0 = -32 + 8 + 2 + 1 = -21$.

Q3. Negate the number represented in 5-bit two's complement as `01111`.

Inverting the bits from zero to one and one to zero, we get `10000`, and adding one gives `10001`. Using Formula (2), `01111` represents 15, and `10001` represents $-16+1=-15$.

3.3 Converting from Two's Complement Representation

We will look at two ways to find the numeric value of a number in two's complement representation: the reverse of the method described above, and another using a formula.

The first way is to examine the most significant bit, which is the left-most bit when written in the standard way. This bit is called the *sign bit*, because it tells us if the number is negative. If this bit is 1, then the number is negative. If it is 0, then the number is zero or positive. If the number is zero or positive, then we can treat it as a number in unsigned representation and convert using the method discussed in the previous lesson. If the number is negative, we determine its value by proceeding as if it were in unsigned representation. This will give a large positive number. We then subtract 2^n from this number to get the negative number—this is the two's complement value.

Q4. What number has the 4-bit two's complement representation `0011`? the 4-bit two's complement representation `1101`? the 8-bit two's complement representation `11111111`?

Using the method above, we note that the sign bit of `0010` is 0, so `0010` represents a non-negative number. Using the formula from the previous lesson, we find that `0011` represents 3.
The sign bit of `1101` is 1, so we know this is a negative number. Interpreting it as a number in unsigned representation, we get 13. Subtracting 2^4 gives -3.
The sign bit of `11111111` is 1, so this is a negative number. The value of `11111111` as an unsigned number is 255. Subtracting 2^8 gives its two's complement value: -1.

The second way to convert from two's complement is using the following formula.

Numerical value of a number in n-bit two's complement representation

$$\text{Value} = -b_{n-1}2^{n-1} + \cdots + b_1 2^1 + b_0 2^0 \tag{2}$$

Note the similarity and difference between this formula the formula for the value of a number is unsigned representation. (The difference is the sign of the coefficient of 2^{n-1}.)

Q5. What number has the 4-bit two's complement representation `0010`? the 4-bit two's complement representation `1100`? the 8-bit two's complement representation `10000000`?

Using Formula (2) above, the unsigned two's complement value of `0010` is:

$$-0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 2.$$

The two's complement value of `1100` is:

$$-1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = -8 + 4 = -4.$$

The two's complement value of `10000000` is:

$$-1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = -128.$$

★ **Q6.** Show why interpreting the bits as an unsigned value, and then subtracting 2^n, is equivalent to Formula (2).

Let $b_{n-1}, b_{n-2}, \ldots, b_1, b_0$ be the bits of the representation. When $b_{n-1} = 1$, the unsigned interpretation of these bits gives:

$$U = 2^{n-1} + b_{n-2} \cdot 2^{n-2} + \cdots + b_1 \cdot 2^1 + b_0.$$

Subtracting 2^n gives:

$$U - 2^n = -2^n + 2^{n-1} + b_{n-2} \cdot 2^{n-2} + \cdots + b_1 \cdot 2^1 + b_0 \quad = -2^{n-1} + b_{n-2} \cdot 2^{n-2} + \cdots + b_1 \cdot 2^1 + b_0.$$

But this is exactly Formula (2) when $b_{n-1} = 1$. Therefore, the two procedures are equivalent.

3.4 Sign Extension

Sometimes we want to convert a number represented using n bits to a wider representation using m bits, where $m > n$. It's obvious how to do this using unsigned representation: just add leading zeroes! In unsigned representation, `10`, `010`, and `0000010` all represent the same number. As you saw in Q2, however, `1000` and `01000` represent -8 and 8, respectively.

As we saw in Section 3.3, if the sign bit is 0, the number is non-negative, and has the same value in two's complement and unsigned representations. Thus, if the sign is 0, we can extend the number to a larger width by adding leading zeros.

As you saw in Q2, the two's complement representation of -8 is `1000` using 4 bits, and `11111000` using 8 bits. In fact, any representation using more than 4 bits is just the 4-bit representation with leading ones. This is true in general: a negative number in n-bit two's complement representation can be extended to $m > n$ bits by adding $m - n$ leading ones.

> ★ **Q7.** Show why the above is true using Formula (2).

Consider a negative number whose n-bit two's complement representation is $1b_{n-2}\cdots b_1 b_0$ and value given Formula (2) is

$$-2^{n-1} + b_{n-2}2^{n-2} + \cdots + b_1 2^1 + b_0 2^0$$

Adding $m-n$ leading ones gives the value

$$-2^{m-1} + 2^{m-2} + \cdots + 2^{n-1} + b_{n-2}2^{n-2} + \cdots + b_1 2^1 + b_0 2^0$$

The leading terms of the sum correspond to the leading ones and the old sign bit. Simplifying gives

$$\begin{aligned} -2^{m-1} + 2^{m-2} + \cdots + 2^{n-1} &= \left(-2^{m-n} + 2^{m-n-1} + \cdots + 1\right) \cdot 2^{n-1} \\ &= \left(-2^{m-n} + (2^{m-n} - 1)\right) \cdot 2^{n-1} \\ &= -2^{n-1}. \end{aligned}$$

Therefore, the n and m bit representations have the same value, and therefore represent the same number.

Combining the two observations, we see that we can extend a number in n-bit two's complement representation to $m > n$ bits by adding leading zeroes if the sign bit is 0 and leading ones if the sign bit is 1. This is called *sign extension*, because we are just repeating the sign bit $m-n$ times to get m bits.

4 Addition and Subtraction Using Two's Complement

Using two's complement representation, we can add and subtract numbers using the same circuit as unsigned representation. To subtract, we negate the subtrahend (the number being subtracted) as discussed above, and add it to the minuend (the number from which we are subtracting). For example, to subtract `0010` from `0101`, we first negate `0010` to obtain `1110`, and then add `1110` to `0101` to obtain `0011`. Check: $101_2 - 10_2 = 5 - 2 = 3 = 11_2$.

> **Q8.** Perform the following operations using 5-bit two's complement representation: $5-12$, $0-1$, $-1-0$.

Start by converting the numbers to two's complement representation: $5 =$ `00101` and $12 =$ `01100`. To get -12, we invert the bits and add one to obtain `10100`. Adding `00101` to `10100`

without carry-out gives `11001`. This is the two's complement representation of $-16+8+1 = -7$.
To compute $0-1$, we need to negate 1, which gives `11111`. We then add `00000` to `11111`, which gives `11111`.
To compute $-1-0$, we need to negate 0, which gives `00000`. (We ignore the carry-out when adding 1.) Then add `11111` to `00000` to obtain `11111`.

Subtracting by adding the negative also works with numbers that are in unsigned representation. This is because negation gives us the additive inverse for modular arithmetic, and this additive inverse works even when working with unsigned representations. Of course, the result is only a meaningful integer value when the result is positive. When the result of subtracting one number in unsigned representation from another would be negative, the result is still correct modulo 2^n (where n is the number of bits we're working with), but not over the integers.

> **Q9.** Perform the following operations using 5-bit unsigned arithmetic: $31-1$, $13-13$.

The unsigned representation of 31 and 1 is `11111` and `00001`. Inverting `00001` and adding 1 gives `11111`! Note that since we're working with 5-bit unsigned representation, `11111` *is* 31. It so happens that 31 is the additive inverse of 1 modulo 32, because $31+1 = 32$, which is congruent to 0 modulo 32. Adding `11111` to itself and dropping the carry-out bit gives `11110`, which is the 5-bit unsigned representation of 30.
The unsigned representation of 13 is `01101`. Inverting `01101` and adding 1 gives `10011`. Adding `01101` and `10011` and dropping the carry-out bit gives `00000`.