Binary Integers

CS 350: Computer Organization & Assembler Language Programming

A. Why?

- Binary integers are one of the basic ways to store information in a modern computer.

B. Outcomes

At the end of today, you should:

- Know the three ways to represent signed binary integers.
- Know the pros and cons of each system.
- Know how to take the negative of a binary number in each system.
- Know how to do subtraction in two's complement.
- Know what overflow is, what it looks like, and when it occurs.

C. Unsigned Binary Integers

- Our hardware represents data using bits; we use bits to represent binary numbers.
- Given n bits, there are 2^n possible bit patterns.
- We can use them to name 2^n possible items.
- For unsigned binary integers, we read the n-bit string as a base 2 number that's ≥ 0.
  - E.g. for 3 bits: 000, 001, 010, 011, 100, 101, 110, 111 are 0 through 7.
  - The bit positions (left to right) are 2^{n-1}, …, 2^2, 2^1, 2^0.
  - The largest unsigned n-bit number (n 1's), represents $2^{n-1}+\ldots+2^1+2^0 = 2^n-1$.
- Unsigned binary addition is like decimal addition: Add right to left, add carry to left if necessary. Unsigned binary subtraction is like decimal subtraction (borrow from left). [See attached example.]

D. Signed Binary Integers

- With signed integers, the leftmost bit used as the sign bit (0 for nonnegative numbers, 1 for negative numbers).
- To get symmetry, we'd like to represent (for some k), the 2 k+1 numbers –k, –k+1, –k+2, …, –1, 0, 1, 2, …, k–1, k.
- This is an odd number, and n bits gives us 2n bit patterns.
- Any scheme for representing negative numbers is either going to have
- Unequal numbers of positive and negative integers
- Multiple representations of the same number. (Typically two versions of zero: "positive" zero and "negative" zero.
- We'll see three methods for representing negative numbers and for taking the negative of a number. (All three methods represent positive numbers the same way.)

E. Sign-Magnitude Negative Numbers

- To take the negative of a number, flip its sign bit. (Easy!)
- Example: 111 is 3-bit unsigned 7, so 0111 represents 7; In sign-magnitude, 1111 represents –7.
- In sign-magnitude, adding a positive and negative number is different from adding two unsigned numbers.
- A problem with sign-magnitude is that it has a "negative" zero: 1000 (in addition to "positive" zero: 0000).

F. One's Complement

- In one's complement, to take the negative of a number, flip all its bits.
- E.g. the negative of 7 is the bit-flip of 0111, which is 1000. Also, the negative of (–7) is the bit-flip of 1000 is 0111.
- Another way to think of taking the negative (in one's complement) involves subtracting each bit from 1.
- I.e., to calculate –k, let N be n one bits, and do an unsigned subtraction of N–k.

- E.g., to calculate –0111, do the unsigned subtraction 1111–0111 = 1000. To calculate –1000, do the unsigned subtraction 1111–1000 = 0111.
- –0101 is 1111–0101 = 1010, and –1010 is 1111–1010 = 0101.
- In decimal, "Nine's complement" corresponds to binary one's complement.
- E.g., the negative of 1234 = 9999–1234 = 8765 [unsigned decimal]. The negative of 8765 is 9999 – 8765 = 1234.
- In one's complement, adding a positive and negative number is different from adding two unsigned numbers.
- A problem with one's complement is that it has a "negative" zero: 1111 (in addition to "positive" zero: 0000).

G. Two's Complement

- In two's complement, to take the negative of a number, take the one's complement and add 1 (as unsigned addition).
- E.g., the negative of 7 is (bit-flip of 0111)+1 = 1000+1 = 1001. The negative of –7 is (bit-flip of 1001)+1 = 0110+1 = 0111.
- There's also a shortcut for taking the negative in two's complement:
- Take our bitstring; going right to left, skip over zeros until you reach a 1, then flip all bits to the left of that 1.
- E.g., for the negative of 0110, we keep the rightmost 10 and flip the 01 to its left, which gives us 1010. For the negative of 1010, we keep the rightmost 10 and flip the 10 to its left, which gives us 0110.
- Note for negative 0111, we skip over no zeros and flip to get 1001. For negative 1001, we skip over no zeros and flip to get 0111.
- Another way to think of taking the negative in two's complement is that $-k$ is represented as $N-k+1$ (where N is n one bits and the subtraction and addition are done as unsigned operations).
- Note: As an unsigned number, n one bits represents 2^n-1, so $N-k+1$ is $(2^n-1)-k+1$ (all using unsigned operations), which equals 2^n-k (using unsigned subtraction).
- The power 2^n is why this is called 2's complement.

- In decimal, "ten's complement" corresponds to binary two's complement.
- E.g., the negative of 1234 = 9999 – 1234+1 = 8765+1 = 8766. The negative of 8766 is 9999 – 8766+1 = 1233+1 = 1234.
- Two's complement only has one kind of zero.
- The negative of 0000 is 1111 – 0000+1 = 1111+1 = 0000 (we throw away the carry out of the leftmost position).
- The problem with two's complement is that it has one more negative number than positive number, and the bitstring 100…0 is its own negative.
- E.g., negative 1000 is 0111+1 = 1000.
- Two's complement 1000 represents decimal –8; 1001 represents –7 (and 0111 represents 7).
- **The really nice feature** of 2's complement (and the reason it's used in hardware) is that two's complement subtraction is the same as unsigned addition of the negative. I.e., in two's complement $x - y$ can be implemented as $x+(-y)$, where the + is unsigned.
- E.g., 6 – 4 is 0110 + (–0100) = 0110 + (1011+1) = 0110 + 1100 = 0010 (there's a carry in and carry out of the leftmost position). In the other direction, 4 – 6 = 0100 + (–0110) = 0100 + (1001+1) = 0100 + 1010 = 1110, which represents –2.

H. Overflow

- Overflow: Result of operation on n bits doesn't fit into n bits.
- In general, if x and y have different signs, then $x + y$ won't overflow.
- Overflow occurs when you try to go too far from zero.
- When can overflow occur in two's complement?
- Taking the negative of the most negative number.
- Symptom: k is negative and so is –k.
- Adding together two positive or two negative numbers and getting a result of the other sign.
- Symptom: positive + positive = negative result or negative + negative = positive result.

- More generally, (if you add two numbers of the same sign), if the carry in to the leftmost position doesn't equal the carry out of the leftmost position, then overflow has occurred.
- If you carry in a 1 and carry out a 0, the sign has changed from + to –.
- If you carry in a 0 and carry out a 1, the sign has changed from – to +.

Binary Integers

CS 350: Computer Organization & Assembler Language Programming

A. Why?

- Binary integers are one of the basic ways to store information in a modern computer.

B. Outcomes

After this activity, you should

- Be able to represent signed binary integers in sign-magnitude, 1's complement, and 2's complement
- Be able to take the negative of a binary number in each of the 3 systems.
- Be able to do subtraction in two's complement.
- Be able to recognize overflow and know when and why it occurs.

C. Questions

1. Complete the following table of representations of 4-bit bitstrings as unsigned, sign-magnitude, 1's complement, and 2's complement binary numbers: Fill in each entry with the decimal number represented by the bitstring, using the given representation technique.

Bitstring	Unsigned	Sign-Magnitude	1's Complement	2's Complement
0111	7	7	7	7
0110	6	6	6	6
0101	5	5	5	5
0100	4	4	4	4
0011	3	3	3	3
0010	2	2	2	2
0001	1	1	1	1
0000	0	0	0	0
1111				
1110				
1101				

Bitstring	Unsigned	Sign-Magnitude	1's Complement	2's Complement
1100				
1011				
1010				
1001				
1000				

2. Use your table to answer the following questions:
   a. What decimal number does 0010 represent in sign-magnitude, 1's complement, and 2's complement?
   b. What is the bit-flip of 0010 (i.e., the bitstring that results from flipping the bits of 0010)?
   c. What (decimal number) does the answer from (b) represent in sign-magnitude?
   d. … in 1's complement?
   e. … in 2's complement?
3. Rewrite the following subtractions in binary, using 5-bit 2's complement. (E.g., 3–1 is rewritten as $00011 - 00001 = 00011 + (-00001) = 00011 + (11110 + 1) = 00011 + 11111 = 00010$.)
   a. $7 - 5 = 2$:
   b. $6 - 8 = -2$:
   c. $-5 - 10 = -15$:
   d. $-12 - 4 = -16$:
   e. $-9 - 9 =$????
4. With an 8-bit number
   a. What is the largest positive number that can be represented? (unsigned? signed?)
   b. What is the largest negative number (i.e., most negative) that can be represented in sign-magnitude?
   c. … in 1's complement?
   d. … in 2's complement?

5. With an n-bit number
   a. What is the largest positive number that can be represented? (unsigned? signed?)
   b. What is the largest negative number (i.e., most negative) that can be represented in sign-magnitude?
   c. … in 1's complement?
   d. … in 2's complement?

Solutions

Bitstring	Unsigned	Sign-Magnitude	1's Complement	2's Complement
0111	7	7	7	7
0110	6	6	6	6
0101	5	5	5	5
0100	4	4	4	4
0011	3	3	3	3
0010	2	2	2	2
0001	1	1	1	1
0000	0	0	0	0
1111	15	-7	-0	-1
1110	14	-6	-1	-2
1101	13	-5	-2	-3
1100	12	-4	-3	-4
1011	11	-3	-4	-5
1010	10	-2	-5	-6
1001	9	-1	-6	-7
1000	8	-0	-7	-8

2 (a) 2, 2, 2; (b) 1101; (c) -5, -2, -3

3a.
```
  7     00111
- 5    +11011 = -00101
----   ----------------
  2     00010 = 2
```

3b.
```
  6     00110
- 8    +11000 = -01000
----   ----------------
- 2     11110 = -00010 = -2
```

3c.
```
 -5     11011 = -00101 = -5
-10    +10110 = -01010 = -10
----   --------------------
-15     10001 = -01111 = -15
```


3d.
```
-12     10100 = -01100 = -12
 - 4   +11100 = -00100 = -4
----   ----------------------
-16     10000 = -16
```

3e.
```
 -9     10111 = -01001 = -9
 - 9   +10111 = -01001 = -9
----   ----------------------
 14     01110 = 14 (overflow!)
```

4a. unsigned: 1111 1111 = 255; signed: 0111 1111 = 127

4b. 1111 1111 = -127

4c. 1000 0000 = -127

4d. 1000 0000 = -128

5a. unsigned: $2^n - 1$; signed: $2^{n-1} - 1$

5b. 11…11 = $-(2^{n-1} - 1) = -2^{n-1} + 1$

5c. 100…00 = $-(2^{n-1} - 1) = -2^{n-1} + 1$

5d. 100…00 = -2^{n-1}

