CS 354
Machine Organization and Programming
Lecture 09

Michael Doescher
Summer 2020

Low Level C Programming
Number Representations

Number Representations

Assume for these next few examples that sizeof(int) -> 1 byte (8 bits)

What is the maximum integer we could represent with these bits?

Number Representations

Assume for these next few examples that sizeof(int) -> 1 byte (8 bits)

What is the maximum integer we could represent with these bits?

Put 1 in every bit position
1111 1111

Number Representations

Assume for these next few examples that sizeof(int) -> 1 byte (8 bits)

What is the maximum integer we could represent with these bits

Put 1 in every bit position
1111 1111

-> $2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$

Number Representations

Assume for these next few examples that sizeof(int) -> 1 byte (8 bits)

What is the maximum integer we could represent with these bits

Put 1 in every bit position
1111 1111

-> $2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$
-> 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1

Number Representations

Assume for these next few examples that sizeof(int) -> 1 byte (8 bits)

What is the maximum integer we could represent with these bits

Put 1 in every bit position
1111 1111

-> $2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$
-> 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1
-> 255

Number Representations

What if we wanted to represent negative numbers?
Consider a 3-bit example:

000 -> 0
001 -> 1
010 -> 2
011 -> 3
100 -> 4
101 -> 5
110 -> 6
111 -> 7

Number Representations

What if we wanted to represent negative numbers?

- One solution is to use one of the bits as a sign bit
- Use the most significant bit (msb) the left most one to represent the sign
- And the rest of the bits to represent the magnitude

000 -> 0
001 -> 1
010 -> 2
011 -> 3
100 -> 4
101 -> 5
110 -> 6
111 -> 7

1. Signed Magnitude Representation

What if we wanted to represent negative numbers?

- One solution is to use one of the bits as a sign bit
- Use the most significant bit (msb) the left most one to represent the sign
- And the rest of the bits to represent the magnitude

000 -> +0
001 -> +1
010 -> +2
011 -> +3
100 ->
101 ->
110 ->
111 ->

1. Signed Magnitude Representation

What if we wanted to represent negative numbers?

- One solution is to use one of the bits as a sign bit
- Use the most significant bit (msb) the left most one to represent the sign
- And the rest of the bits to represent the magnitude

000 -> +0
001 -> +1
010 -> +2
011 -> +3
100 -> -0
101 -> -1
110 -> -2
111 -> -3

1. Signed Magnitude Representation

What if we wanted to represent negative numbers?

- One solution is to use one of the bits as a sign bit
- Use the most significant bit (msb) the left most one to represent the sign
- And the rest of the bits to represent the magnitude

000 -> +0

001 -> +1

010 -> +2

011 -> +3

100 -> -0

101 -> -1

110 -> -2

111 -> -3

Downside

We have two ways to represent 0?

1. Signed Magnitude Representation

What if we wanted to represent negative numbers?

- One solution is to use one of the bits as a sign bit
- Use the most significant bit (msb) to represent the sign
- And the rest of the bits to represent the magnitude

000 -> +0

001 -> +1

010 -> +2

011 -> +3

100 -> -0

101 -> -1

110 -> -2

111 -> -3

Downside

We have two ways to represent 0?

- Difficult to do in hardware!
- What about ==, <, <=, >, >=, !=

2. One’s Complement

What if we wanted to represent negative numbers?

- Complement means that we just flip the bits for negative numbers

000 -> +0 :: complement -> 111
001 -> +1 :: complement -> 110
010 -> +2 :: complement -> 101
011 -> +3 :: complement -> 100

2. One’s Complement

What if we wanted to represent negative numbers?

- Complement means that we just flip the bits for negative numbers

000 -> +0 :: complement -> 111 -> -0
001 -> +1 :: complement -> 110 -> -1
010 -> +2 :: complement -> 101 -> -2
011 -> +3 :: complement -> 100 -> -3

2. One's Complement

What if we wanted to represent negative numbers?

- Complement means that we just flip the bits for negative numbers

000 -> +0 :: complement -> 111 -> -0
001 -> +1 :: complement -> 110 -> -1
010 -> +2 :: complement -> 101 -> -2
011 -> +3 :: complement -> 100 -> -3
100 -> -3
101 -> -2
110 -> -1
111 -> -0

Same Downside

We have two ways to represent 0?

- Difficult to do in hardware!
- What about ==, <, <=, >, >=, !=

2. One's Complement

What if we wanted to represent negative numbers?

- Complement means that we just flip the bits for negative numbers
- The subtraction algorithm for calculating one's complement
- Subtract the positive representation from the all 1s bit pattern
- E.g. -1 and -3

$$\begin{array}{r} 111 \\ -001 \\ \hline 110 \end{array} \qquad \begin{array}{r} 111 \\ -011 \\ \hline 100 \end{array}$$

3. Two's Complement

What if we wanted to represent negative numbers?

- Currently used in modern computers
- The Most Significant Bit is the sign bit (true for all 3 representations)
- Just change the formula for converting unsigned binary to decimal by including the negative sign

000 -> +0
001 -> +1
010 -> +2
011 -> +3

$$-b_2 2^2 + b_1 2^1 + b_0 2^0$$

3. Two's Complement

What if we wanted to represent negative numbers?

$$-b_2 2^2 + b_1 2^1 + b_0 2^0$$

000 -> +0
001 -> +1
010 -> +2
011 -> +3
100 -> $-1*2^2 + 0*2^1 + 0*2^0 = -4 + 0 + 0 = -4$
101 ->
110 ->
111 ->

3. Two's Complement

What if we wanted to represent negative numbers?

$$-b_2 2^2 + b_1 2^1 + b_0 2^0$$

000 -> +0

001 -> +1

010 -> +2

011 -> +3

100 -> $-1*2^2 + 0*2^1 + 0*2^0 = -4 + 0 + 0 = -4$

101 -> $-1*2^2 + 0*2^1 + 1*2^0 = -4 + 0 + 1 = -3$

110 ->

111 ->

3. Two's Complement

What if we wanted to represent negative numbers?

000 -> +0
001 -> +1
010 -> +2
011 -> +3
100 -> -4
101 -> -3
110 -> -2
111 -> -1

$$-b_2 2^2 + b_1 2^1 + b_0 2^0$$

3. Two's Complement

What if we wanted to represent negative numbers?
Still works for positive numbers!!

$$-b_2 2^2 + b_1 2^1 + b_0 2^0$$

000 -> +0

001 -> +1

010 -> +2 $= -0*2^2 + 1*2^1 + 0*2^0 = 0 + 2 + 0 = 2$

011 -> +3

100 -> -4

101 -> -3

110 -> -2

111 -> -1

3. Two's Complement

What if we wanted to represent negative numbers?
Another algorithm

000 -> +0
001 -> +1
010 -> +2
011 -> +3
100 -> -4
101 -> -3
110 -> -2
111 -> -1

If we get n bits (3 bits in this example)
If b = 2 :: 010
How can we represent -2

2^n = 2^3 = 1000
Subtract b $\underline{-010}$

3. Two’s Complement

What if we wanted to represent negative numbers?
Another algorithm

000 -> +0
001 -> +1
010 -> +2
011 -> +3
100 -> -4
101 -> -3
110 -> -2
111 -> -1

Remember subtraction in decimal
151 Borrow from the decimal to the left
- 85 and rearrange

$$\begin{array}{r} 14\ (11) \\ -\ \ 8\ \ \ 5 \\ \hline 6\ \ \ 6 \end{array}$$

3. Two's Complement

What if we wanted to represent negative numbers?
Another algorithm

000 -> +0
001 -> +1
010 -> +2
011 -> +3
100 -> -4
101 -> -3
110 -> -2
111 -> -1

If we get n bits (3 bits in this example)
If b = 2 :: 010
How can we represent -2

$2^n = 2^3 = 1000$ 120
Subtract b -010 -010

110

3. Two's Complement

What if we wanted to represent negative numbers?
Yet Another algorithm

000 -> +0
001 -> +1
010 -> +2
011 -> +3
100 -> -4
101 -> -3
110 -> -2
111 -> -1

Consider a 1-byte integer (8 bits)
3 -> 00000011
How can we get directly to -3?

Step 1: Takes the ones complement
Step 2: Add 1

```
11111100
+      1
11111101
```

Number Range

000 -> +0
001 -> +1
010 -> +2
011 -> +3
100 -> -4
101 -> -3
110 -> -2
111 -> -1

For a 3-bit example the allowed number range
Goes from -4 to +3

In general for n bits.
-2^{n-1} to $+2^{n-1}-1$

For unsigned numbers
0 to +7
0 to $2^n - 1$

Bit Patterns

000 -> +0
001 -> +1
010 -> +2
011 -> +3
100 -> -4
101 -> -3
110 -> -2
111 -> -1

1 as a signed number vs 1 as an unsigned number
Signed -> 001 and Unsigned -> 001

Bit Patterns

000 -> +0
001 -> +1
010 -> +2
011 -> +3
100 -> -4
101 -> -3
110 -> -2
111 -> -1

1 as a signed number vs 1 as an unsigned number
Signed -> 001 and Unsigned -> 001

What about -1?
Signed -> 111 and Unsigned -> 7

Bit Patterns

000 -> +0	000 -> 0
001 -> +1	001 -> 1
010 -> +2	010 -> 2
011 -> +3	011 -> 3
100 -> -4	100 -> 4
101 -> -3	101 -> 5
110 -> -2	110 -> 6
111 -> -1	111 -> 7

1 as a signed number vs 1 as an unsigned number
Signed -> 001 and Unsigned -> 001

What about -1?
Signed -> 111 and Unsigned -> 7

Bit Patterns

Number Range for signed 1-byte int (char)
-2^7 to 2^7-1 = -128 to 127

Number Range for unsigned 1-byte int (unsigned char)
0 to 2^8-1 = 0 to 255

Bit Patterns

Number Range for signed 1-byte int (char)
-2^7 to $2^7-1 = -128$ to 127

Number Range for unsigned 1-byte int (unsigned char)
0 to $2^8-1 = 0$ to 255

Same example representing -1 in 8 bits
signed int -> -1 -> 11111111 -> 0xFF
unsigned int 0xFF -> 255