Binary Arithmetic II

CSCI 2050U - Computer Architecture

Randy J. Fortier
@randy_fortier

Outline

- Signed number representations
- Binary subtraction
- Overflow

Signed Binary Representations

CSCI 2050U - Computer Architecture

Signed Numbers

- Unsigned numbers are relatively simple, since we can just use the basic decimal to binary conversion process discussed in the last lecture
- Signed numbers could be implemented three ways:
  - Signed bit representation
  - 1s complement
  - 2s complement

Signed Bit Representation

- Use one of the bits of the binary representation to represent the sign
  - `0` - zero or positive value
  - `1` - negative value
- The rest of the number would represent the magnitude (value)
- e.g. `0110 1100`
  - `0` - this number is either zero or positive (non-negative)
  - `110 1100` - use normal binary to decimal conversion (`108`)
- Advantage: Easy to explain to CS students
- Disadvantage: No arithmetic works

Signed Bit Representation: Arithmetic

- How do we add numbers represented in this way?

```
    11  1
  1001 0101    -21
+ 0001 1100    +28
  ---------    ---
  1011 0001    -49 (incorrect)
```

1s Complement Representation

- Positive numbers have a leftmost bit `0` (just like in sign bit representation), and the rest of the number is normal binary
- Negative numbers are the positive number in binary, but with all bits flipped (complemented)
- e.g. `0110 1100 (positive, 108)`
  - `0` - this number is either zero or positive (non-negative)
  - `110 1100` - use normal binary to decimal conversion (`108`)
- e.g. `1110 1100 (negative, -19)`
  - `1` - this number is negative
  - Flip the remaining bits: `110 1100 → 001 0011` (which is `19` in decimal)
  - Therefore, this number is `-19`

1s Complement Representation

- Advantage: None
- Disadvantage: Arithmetic *almost* works

1s Complement Representation: Arithmetic

- How do we add numbers represented in this way?

```
                 Positive:
  1111
  1110 1010     0001 0101        -21
+ 0001 1100                      +28
  ---------                      ---
  0000 0110                        6 (incorrect, but almost)
```

2s Complement Representation

- Two-step process to negate a number:
  - Perform the 1s complement
  - Add 1 to the result
- Since this is complicated, to find out the value (magnitude) of a negative number, use these two steps (above) to make it positive to see its magnitude
- The magnitude of the original (negative) number will be the same

2s Complement Representation

- e.g. `0110 1100 (positive, 108)`
  - `0` - this number is either zero or positive (non-negative)
  - `110 1100` - use normal binary to decimal conversion (`108`)
- e.g. `1110 1100 (negative, -20)`
  - `1` - this number is negative
  - 1s complement: `1110 1100 → 0001 0011`
  - Add 1 to the result: `0001 0011 → 0001 0100 (20)`
  - Therefore, this number is `-20`

2s Complement Representation

- Advantage: Arithmetic works!
- Disadvantage: A bit tougher for CS students to learn

Alternative Twos Complement Technique

- This process produces identical results:
  - Start from the rightmost (least significant) digit
  - Copy all of the zeroes
  - Copy the first one
  - Invert all the remaining bits

```
0110 1100 (+108)
1001 0100 (-108)
```

```
1110 0101 (-27)
0001 1011 (+27)
```

2s Complement Representation: Arithmetic

- How do we add numbers represented in this way?

```
                    Complement:         Add one:
  1111
  1110 1011        0001 0100         0001 0101      -21
+ 0001 1100                                         +28
  0000 0111                                           7  (correct)
```

2s Complement Representation

- Simple way to remember:
  - The left-most bit still represents the same amount as before, but negative
  - For an 8-bit number, instead of `128`, the left-most bit represents `-128`

2s Complement Operation

- The operation that we just learned can be used to negate a number:

    1. Invert (complement) all of the bits
    2. Add 1 to the result

- Example (positive to negative):

    - `0110 1000` (`104`)
    - `1001 0111` (invert all of the bits)
    - `1001 1000` (add `1`, `-104`)

2s Complement Operation

- The operation that we just learned can be used to negate a number:
  1. Invert (complement) all of the bits
  2. Add 1 to the result
- Example (negative to positive):
  - `1001 1000` (`-104`)
  - `0110 0111` (invert all of the bits)
  - `0110 1000` (add `1`, `104`)

Binary Subtraction

CSCI 2050U - Computer Architecture

Ontario Tech University

Binary Subtraction

- Half subtractor (HS) - subtracts two bits
- Full subtractor (FS) subtracts two bits with a possible borrow bit
- One way to design a subtraction circuit:
  - Design half subtractors and full subtractors
  - Combine them to subtract multi-bit numbers
- In practice, we don't have to do it this way

Binary Subtraction

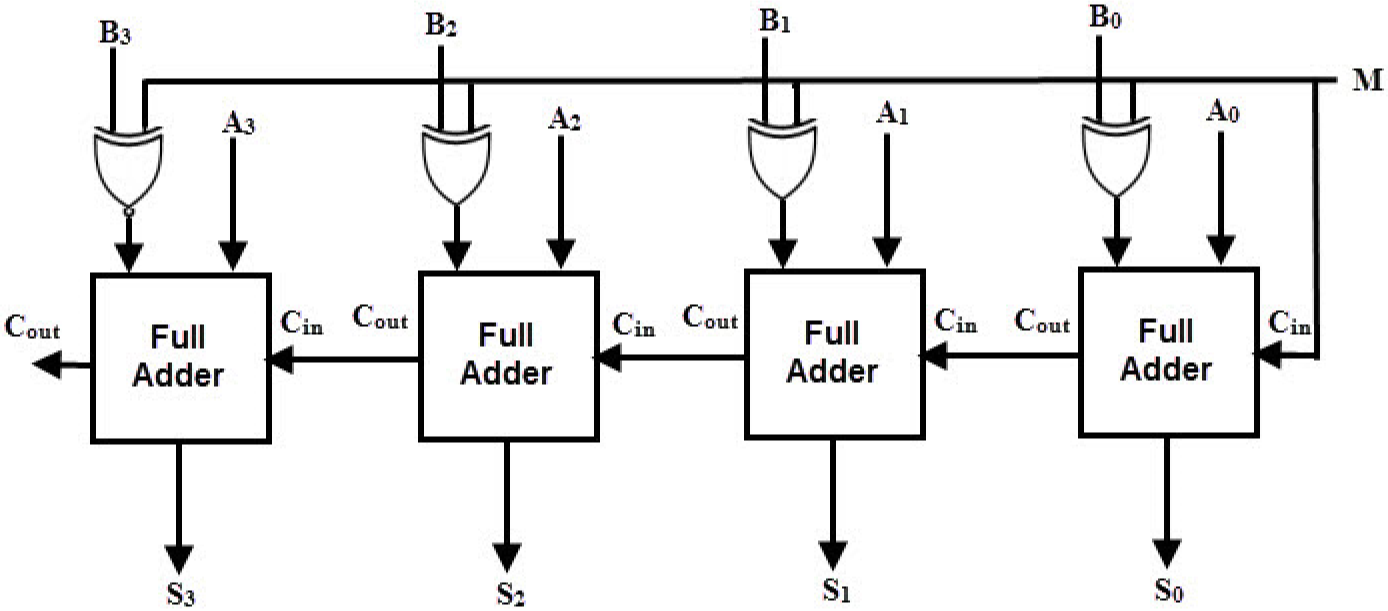
- In decimal, subtracting A-B is the same as adding A+(-B)
  - The same is true in binary
- So, subtracting A-B could be done as follows:
  - Negate B (i.e. apply the twos complement operation)
  - Add A and -B

Binary Subtraction - Implementation

- How to negate B?
- Recall that to negate B:
    1. Complement all of the bits
    2. Add one

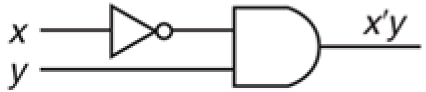
- How to complement bits in a digital circuit?
    1. XOR with `0`: no effect on the bit value
    2. XOR with `1`: the bit value is complemented
        - We could XOR each bit with `1`

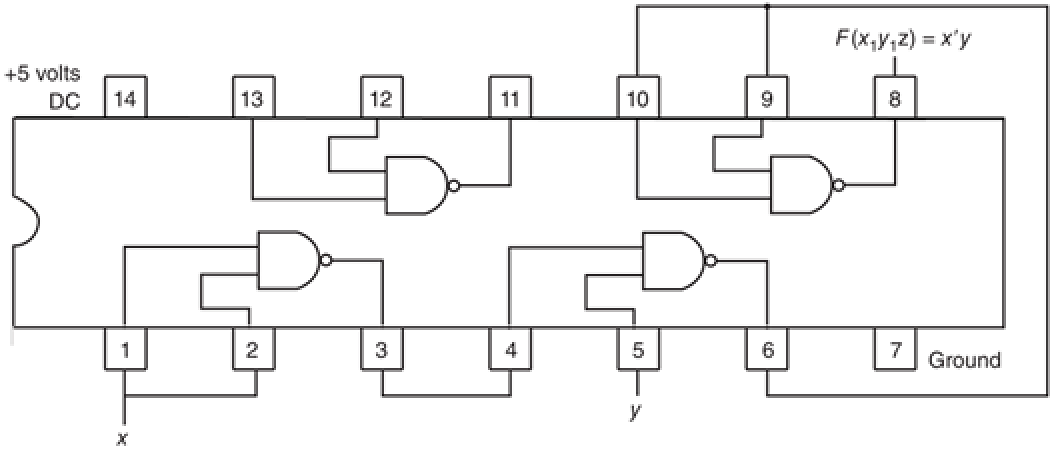
Adder-Subtractor Circuit



Implementing Circuits

- If you don’t happen to have your own multi-billion dollar foundry, you can still build your own circuits using TTL chips (e.g. TI 7426)





Overflow

CSCI 2050U - Computer Architecture

Ontario Tech University

Overflow - Unsigned Integers

- Overflow means that the result of an arithmetic operation cannot be correctly represented in the number of bits available
  - For addition, this means that we've exceeded the bounds of our representation
- With unsigned addition, overflow happens when we go beyond the limits of our representation
  - This is easily recognized by a *carry out*

```
1 111  111
  1011 0101   (181)
+ 0101 0111   (87)
  ---------
  0000 1100   (12)
```

Overflow - Signed Integers

- With signed integers, a carry out doesn't indicate overflow
  - Can adding one positive and one negative signed integers ever result in overflow?

Overflow - Signed Integers

- With signed integers, a carry out doesn't indicate overflow
  - Can adding one positive and one negative signed integers ever result in overflow? No
  - Can adding two negative (or two positive) signed integers ever result in overflow?

Overflow - Signed Integers

- With signed integers, a carry out doesn't indicate overflow
  - Can adding one positive and one negative signed integers ever result in overflow? No
  - Can adding two negative (or two positive) signed integers ever result in overflow? Yes
    - Adding two negative signed integers should produce a negative result
    - Adding two positive signed integers should produce a positive result

Overflow - Signed Integers

- Detecting overflow when adding signed integers:
  - Does the sign of the result match the sign of both of the input numbers?
    - No → overflow

```
1 111     1
  1011 0101
(-75)
+ 1101 1001     (-39)
  1000 1110    (-114)
```

```
1         111
  1011 0101
(-75)
+ 1000 0011    (-125)
  0011 1000     (+56)
```

Wrap-up

- Signed number representations
  - Sign bit representation
  - 1s complement
  - 2s complement
- Binary subtraction
- Overflow
  - Unsigned overflow
  - Signed overflow

What is next?

- Shift and rotation
- Booth's algorithm