When N = 32

- If we use the two's complement method
  - 0000 0000 0000 0000 0000 0000 0000 0000 = 0
  - 0000 0000 0000 0000 0000 0000 0000 0001 = +1
  - 0000 0000 0000 0000 0000 0000 0000 0010 = +2
  - ...
  - 0111 1111 1111 1111 1111 1111 1111 1110 = +2,147,483,646
  - 0111 1111 1111 1111 1111 1111 1111 1111 = +2,147,483,647
  - 1000 0000 0000 0000 0000 0000 0000 0000 = - 2,147,483,648
  - 1000 0000 0000 0000 0000 0000 0000 0001 = - 2,147,483,647
  - 1000 0000 0000 0000 0000 0000 0000 0010 = - 2,147,483,646
  - ...
  - 1111 1111 1111 1111 1111 1111 1111 1101 = - 3
  - 1111 1111 1111 1111 1111 1111 1111 1110 = - 2
  - 1111 1111 1111 1111 1111 1111 1111 1111 = - 1

Addition

- We are quite familiar with adding two numbers in decimal
  - What about adding two binary numbers?

- If we use the two’s complement method to represent binary numbers, addition can be done in a straightforward way

Suppose:
N=8
a=20
b=30

What is result and carry out?
Addition

- N=8, a=20, b=30
- Do binary addition to get result and carryout
- Convert A and B to binary? How?
  - a=20=4+16=2^2+2^4 => a is 0001 0100b
  - b=30=16+8+4+2=2^4+2^3+2^2+2^1 => b is 0001 1110b

```
0001 0100b
+ 0001 1110b
-------------------
0011 0010b
```

NOTE: N=8 in this example

Addition

- N=8, a=80, b=50
- Do binary addition to get result and carryout
- Convert A and B to binary? How?
  - A=80=64+16=2^6+2^4 => a is 0101 0000b
  - b=50=32+16+2=2^5+2^4+2^1 => b is 0011 0010b

```
0101 0000b
+ 0011 0010b
-------------------
1000 0010b
```

Result is NEGATIVE!
Overflow

- Because we use a limited number of digits to represent a number, the result of an operation may not fit
- No overflow when result remains in expected range
  - We add two numbers with different signs
  - We subtract a number from another number with the same sign
- When can overflow happen?

\[
\begin{array}{ccc}
  a & b & \text{overflow possible?} \\
  + & + & \text{yes} \\
  + & - & \text{no} \\
  - & + & \text{no} \\
  - & - & \text{yes}
\end{array}
\]

Overflow

- What is special about the cases where overflow happened?
  - The input values signs are the same; so, can go outside range
- Overflow detection
  - Adding two positive numbers yields a negative number
  - Adding two negative numbers yields a positive number

Check signs
a is positive
b is positive
result isn't!
Overflow

- Can detect by inspecting sign bits of inputs and output
- Alternatively, can also detect by watching “carries”

\[
\begin{align*}
01110 000 & \quad \text{(carries from previous bit add)} \\
0101 0000 & \quad \text{a} \\
+ 0011 0010 & \quad \text{b} \\
\hline
01000 0010 & \\
\end{align*}
\]

Notice the carry into sign bit is different than the final carryout
When carry into sign bit doesn’t equal carryout implies overflow

What happens on overflow?

- The CPU can
  - Generate an exception (what is an exception?)
  - Set a flag in the status register (what is the status register?)
  - Do nothing

- Languages may have different notions about overflow

- Do we have overflows in the case of unsigned, always positive numbers?
  - Example: addu, addiu, subu
MIPS example

- I looked at the MIPS32 instruction set manual
- ADD, ADDI instructions generate an exception on overflow
- ADDU, ADDIU are silent

```assembly
li $t0, 0x40000000
add $t1, $t0, $t0
li $t0, 0x40000000
addu $t1, $t0, $t0
```

MARS give error
MARS doesn't give error

Subtraction

- We know how to add
- We know how to negate a number
- We will use the above two known operations to perform subtraction

- \[ A - B = A + (-B) \]
- The hardware used for addition can be extended to handle subtraction!
Subtraction

- N=8, a=90, b=20
- Do binary subtraction (A+(-B)) to get result and carryout
- Convert A and B to binary? How?
  - a=90 is 0101 1010\(_b\)
  - b=20 is 0001 0100\(_b\)

\[
\begin{array}{c}
\text{find } -b \\
\text{invert } 0001 \text{ 0100}_b \\
= 1110 \text{ 1011}_b \\
+ 0000 \text{ 0001}_b \\
\hline
1110 \text{ 1100}_b
\end{array}
\]

NOW, add a
\[
\begin{array}{c}
0101 \text{ 1010}_b \\
+ 1110 \text{ 1100}_b \\
\hline
1 \text{ 0100} \text{ 0110}_b
\end{array}
\]

1-bit adder

- We will look at a single-bit adder
  - Will build on this adder to design a 32-bit adder
- 3 inputs
  - A: 1\(^{st}\) input
  - B: 2\(^{nd}\) input
  - \(C_{\text{in}}\): carry input
- 2 outputs
  - S: sum
  - \(C_{\text{out}}\): carry out
1-bit adder

- What are the binary addition rules?
  - $0 + 0 = 0$, Cout = 0
  - $0 + 1 = 1$, Cout = 0
  - $1 + 0 = 1$, Cout = 0
  - $1 + 1 = 0$, Cout = 1

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>S</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input Values</th>
<th>Output Values</th>
</tr>
</thead>
</table>

- What about Cin?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>S</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1-bit adder

- What about Cin?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>S</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

N-bit adder

- An N-bit adder can be constructed with N single-bit adders
  - A carry out generated in a stage is propagated to the next ("ripple-carry adder")

- 3 inputs
  - A: N-bit, 1st input
  - B: N-bit, 2nd input
  - Cin: carry input

- 2 outputs
  - S: N-bit sum
  - Cout: carry out
N-bit ripple-carry adder

carry in at each stage

carry out at each stage