Data representation

College of Saint Benedict & Saint John’s University

decimal refresher

58036

decimal refresher

5		8		0		3		6
50000	+	8000	+	0	+	30	+	6

decimal refresher

5		8		0		3		6
50000	$+$	8000	$+$	0	$+$	30	$+$	6
5×10000	$+$	8×1000	$+$	0×100	$+$	3×10	$+$	6×1

decimal refresher

5		8		0		3		6
50000	$+$	8000	$+$	0	$+$	30	$+$	6
5×10000	$+$	8×1000	$+$	0×100	$+$	3×10	$+$	6×1
5×10^4	$+$	8×10^3	$+$	0×10^2	$+$	3×10^1	$+$	6×10^0

binary refresher

10110

binary refresher

1		0		1		1		0
1×2^4	$+$	0×2^3	$+$	1×2^2	$+$	1×2^1	$+$	0×2^0

binary refresher

1		0		1		1		0
1×2^4	$+$	0×2^3	$+$	1×2^2	$+$	1×2^1	$+$	0×2^0
1×16	$+$	0×8	$+$	1×4	$+$	1×2	$+$	0×1

binary refresher

1		0		1		1		0
1×2^4	$+$	0×2^3	$+$	1×2^2	$+$	1×2^1	$+$	0×2^0
1×16	$+$	0×8	$+$	1×4	$+$	1×2	$+$	0×1
16	$+$	0	$+$	4	$+$	2	$+$	0

- this is the representation for unsigned binary integers
- so how to represent signed integers?
  - why not use the leftmost bit to store the sign?
    - what is the range of values if we choose this? — 0 is represented twice, so our range has one less value — not the end of the world
    - what happens if we add to -5 to +5? — the result is -10?

unsigned addition

$$
\begin{aligned}
0 + 0 &= 0 \\
0 + 1 &= 1 \\
1 + 0 &= 1 \\
1 + 1 &= 10
\end{aligned}
$$

	0	0	0	1	0	1	= 5
ADD	0	0	0	1	0	1	= 5

unsigned addition

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 10$$

	0	0	0	1	0	1	= 5
ADD	0	0	0	1	0	1	= 5
C = 0	0	0	1	0	1	0	= 10

- the hardware has a special bit known as the *carry* bit, denoted by C, which stores a 1 if the result of the addition was a carry, and 0 otherwise.

signed addition

	0	0	0	1	0	1	$= +5$
ADD	1	0	0	1	0	1	$= -5$

signed addition

	0	0	0	1	0	1	$= +5$
ADD	1	0	0	1	0	1	$= -5$
$C = 0$	1	0	1	0	1	0	$= -10$

- what is the problem
  - in this case, we had two additional symbols, $+$ and $-$, and we were making some assumptions about their behavior.
  - For example, we know that $5 ADD 5 = 10$, but what does $+ADD-$ equal? — we have no rule for that in our definition of decimal.
  - we are trying to apply the addition algorithm, when we should be applying a different algorithm, called *subtraction*
  - can we choose a different representation that we can directly use the addition algorithm with?

one’s complement

NOT	0	0		0	1	0	1

one’s complement

NOT	0	0	0	1	0	1
	1	1	1	0	1	0

- one’s complement is known as logical not

one's complement

NOT	0	0	0	1	0	1
	1	1	1	0	1	0

	0	0	0	1	0	1
ADD	1	1	1	0	1	0

- one's complement is known as logical not

one's complement

NOT	0	0	0	1	0	1
	1	1	1	0	1	0

	0	0	0	1	0	1
ADD	1	1	1	0	1	0
$C = 0$	1	1	1	1	1	1

- one's complement is known as logical not
- adding the one's complement will always result in all 1s

one's complement

NOT	0	0	0	1	0	1
	1	1	1	0	1	0

	0	0	0	1	0	1
ADD	1	1	1	0	1	0
$C = 0$	1	1	1	1	1	1
ADD	0	0	0	0	0	1

- one's complement is known as logical not
- adding the one's complement will always result in all 1s

one's complement

NOT	0	0	0	1	0	1
	1	1	1	0	1	0

	0	0	0	1	0	1
ADD	1	1	1	0	1	0
$C = 0$	1	1	1	1	1	1
ADD	0	0	0	0	0	1
$C = 1$	0	0	0	0	0	0

- one's complement is known as logical not
- adding the one's complement will always result in all 1s
- so two's complement is NOT + 1