CS1800
9/15 - Fri :)

Admin

- Hw1 out! due 9/22 11:59pm
- Rec1 quiz → due 9/18 9pm
- Tutoring → look on piazza / website
- Live Q+A for questions during lecture

Agenda

1. Repping signed numbers
2. Arithmetic with signed numbers
3. Modular arithmetic

0. Summary from Tues. → unsigned

other base → decimal

decimal → other base (Euclid's division)

arithmetic (just like the decimal)

1. Repping Signed Numbers

- numbers are positive / negative
- needed ...
  - sometimes we need negatives
  - computers do subtraction with addition

ex: 58 - 14 ✗
58 + -14 :)

- same amount of space for every number → "real life" 32 or 64 bits

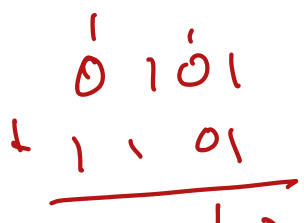
0 0 0 0 1 0 1

started with...

- sign and magnitude
- designate a sign bit (leftmost)
  - 0 = pos
  - 1 = neg

0 1 0 1 +5
1 1 0 1 -5

- everything else the same



should be zero

Today... two's complement

- sign bit (leftmost)
  0 = pos, 1 = neg
- value of the number is different when positive / negative
- Always have same # bits for each number
- Addition algorithm always the same
- 2's comp is for both pos/neg

ex: 3-bit two's complement
(# bits must be known!)

sign ↑ _ _ _

	dec
0 1 1	3
0 1 0	2
0 0 1	1
0 0 0	0
1 1 1	-1
1 1 0	-2
1 0 1	-3
1 0 0	-4

highest: 3
lowest: -4

Decimal ⇒ 2's Complement

0 0 0 1 0 1

1. number is positive
   - make leftmost bit zero
   - convert decimal → binary in usual way
   - fill in missing bits with zeroes

2. number is negative
   - pretend it's positive
   - convert to binary
   - flip the bits
   - add one

- what if number is too big? ⟶ Sorry :(
  - know highest possible #
  - if result of addition ... overflow

6 bit two's complement ...

- what's highest value I can rep?

0	1	1	1	1	1	$= 31_b$
	16	8	4	2	1	

6 bit two's complement

$9_{10} = \_\_\_\__2$

$29_{10} = \_\_\_\__2$

001001 (9)

011101 (29)

$-24_{10} = \_\_\_\__2$

011000 (+24)

100111 (flip)

+ 1

101000 (-24)

2. Arithmetic with Signed Numbers

(ex) 9 - 24 ... 9 + -24 — 6 bit

```
  001001   (9)
+ 101000
--------
  110001   --> done!
```

Sanity check: Is this -15? → Yes!

- subtract 1
- flip bits
- Is it correct if positive

```
  110000
  001111
```

(15) :)

10:53

Arithmetic in two's complement

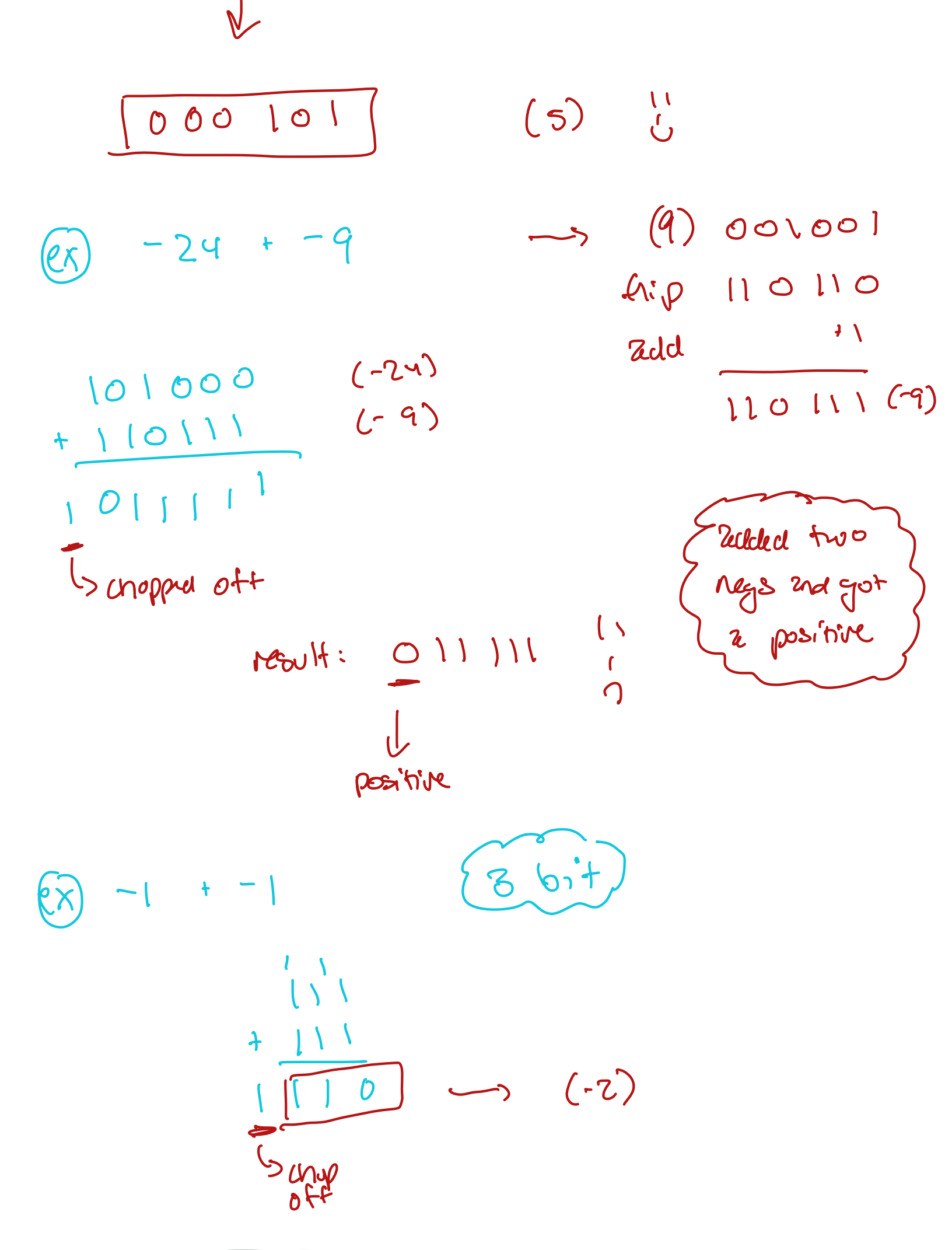
- normal binary addition (same steps as decimal)
- always have same # bits
- If after addition...
  - we get an extra bit → Chop it off! (left most bit)
  - after chopping
    - maybe fine
    - maybe result is crazy → overflow

- overflow: value of result is too big / small
  - too big: high pos
  - too small: low neg

(6 bit)

(ex) 29 - 24 ~~ 29 + -24

```
  011101   (29)
+ 101000   (-24)
--------
 100101   → too many bits! Chop off left
```


000 101
(5)
ex -24 + -9
(9) 001001
flip 110110
add +1
110111 (-9)
101000 (-24)
+ 110111 (-9)
101111
chopped off
added two negs and got a positive
result: 011111
positive
ex -1 + -1
8 bit
111
+ 111
1110
chop off
(-2)

3. Modular Arithmetic

- Back to decimal
- modulo is a mathematical operation
  - useful in many prog. languages
    - %, mod

Euclid for a minute...

$$n = p \cdot q + r$$

- r: remainder ($< p$)
- q: quotient
- p: non zero

(ex) $16 = 5 \cdot 3 + 1$

(n = 16, p = 5, q = 3, r = 1)

- what is remainder for $16 \div 5$?

$16 \bmod 5 = 1$

Real life

- credit card #s
- product code #s

} validated when typed in that it's a valid number

$n = p \cdot q + r$

$20 \bmod 6 = 2$

$15 \bmod 1 = 0$

$3 \bmod 5 = 3$

$-11 \bmod 4 = 1$

$-4 \bmod 7 = 3$

$-2 \bmod 2 = 0$

$20 = 6 \cdot 3 + 2$

$15 = 1 \cdot 15 + 0$

$3 = 5 \cdot 0 + 3$

$-11 = 4 \cdot -3 + 1$

$-4 = 7 \cdot -1 + 3$

$-2 = 2 \cdot -1 + 0$