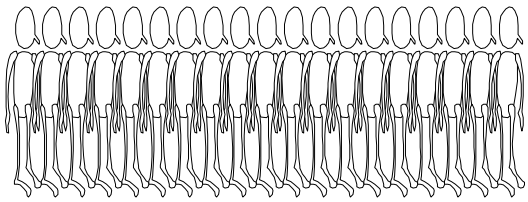
Great Theoretical Ideas In Computer Science		
Anupam Gupta		CS 15-251 Spring 2005
Lecture 19	Mar 22, 2005	Carnegie Mellon University

Grade School Again:
A Parallel Perspective



Plus/Minus Binary (Extended Binary)

Extended Base 2:

Each digit can be -1, 0, 1

Example:

1 -1 -1 0 1 -1

32 + -16 + -8 + 0 + 2 + -1 = 9

Similar to Egyptian Base-3

One weight for each power of 3.
Left = "negative". Right = "positive"

Plus/Minus Binary (Extended Binary)

Base 2:

Each digit can be -1, 0, 1

Example:

1 -1 -1 0 1 -1

32 + -16 + -8 + 0 + 2 + -1 = 9

Note: 1 0 0 1 = 9 as well

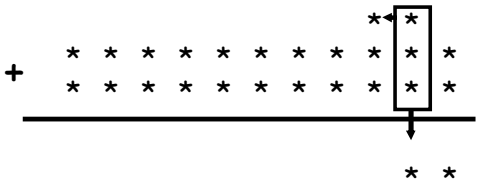
How to add 2 n-bit numbers.

\+ * * * * * * * * * * * *
* * * * * * * * * * * *

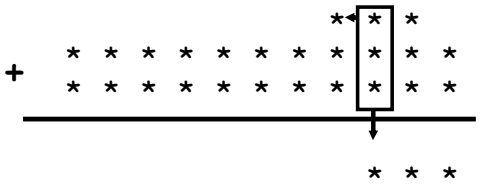
How to add 2 n-bit numbers.

\+ * * * * * * * * * * * *
* * * * * * * * * * * *
* *
*

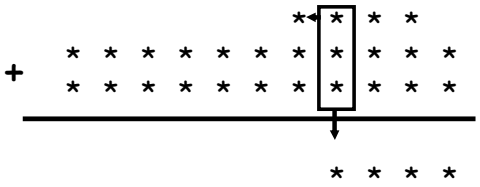
How to add 2 n-bit numbers.



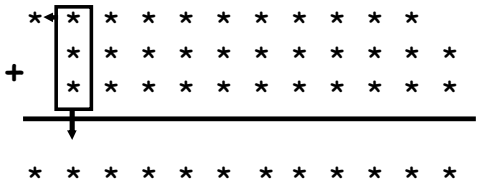
How to add 2 n-bit numbers.



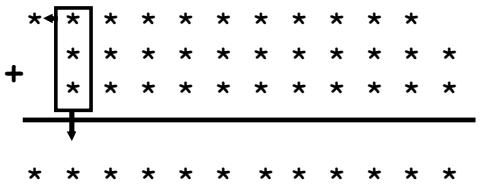
How to add 2 n-bit numbers.



How to add 2 n-bit numbers.

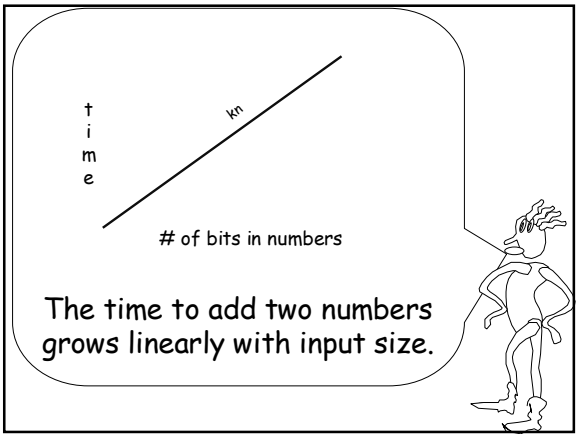


How to add 2 n-bit numbers.



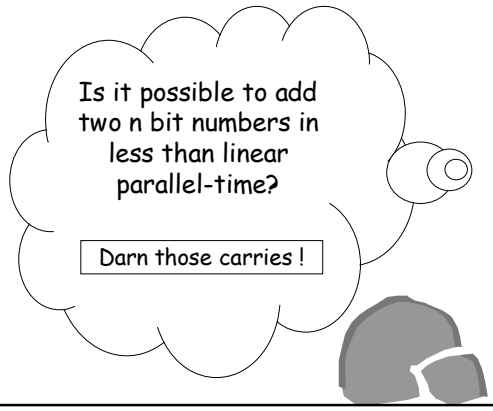
Let c be the maximum time that it takes you to do

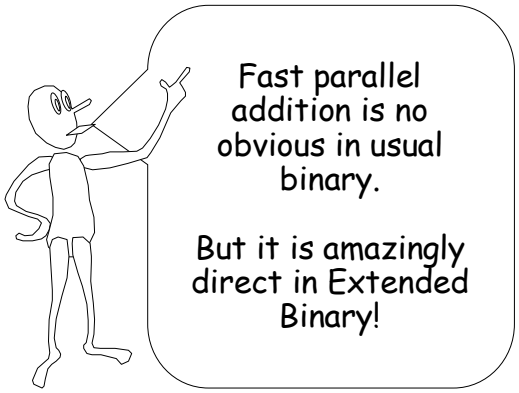
$T(n) \leq cn$ is proportional to n

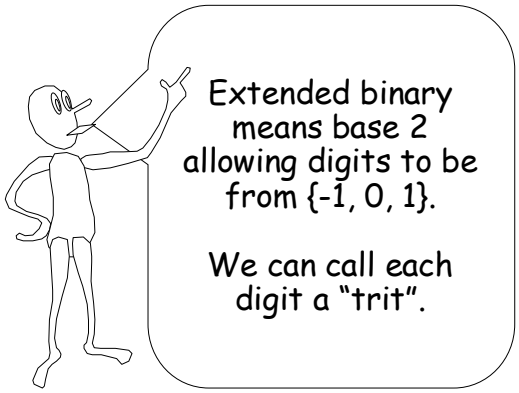


The time to add two numbers grows linearly with input size.

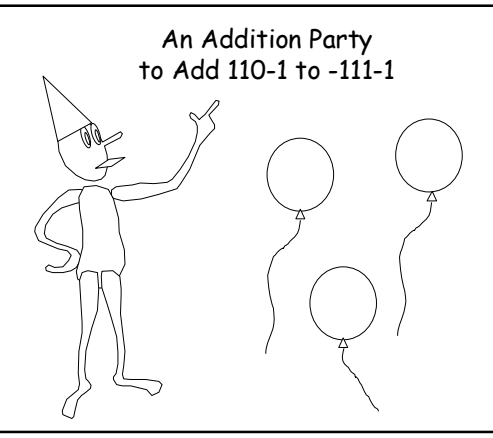
If n people agree to help you add two n bit numbers, it is not obvious that they can finish faster than if you had done it yourself.



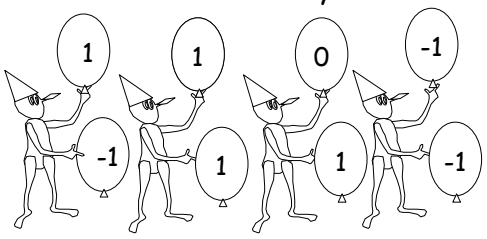




Theorem: n people can add two n-trit, plus/minus binary numbers in constant time!

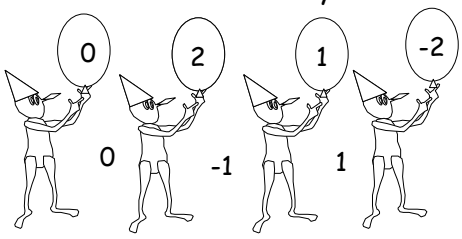


An Addition Party



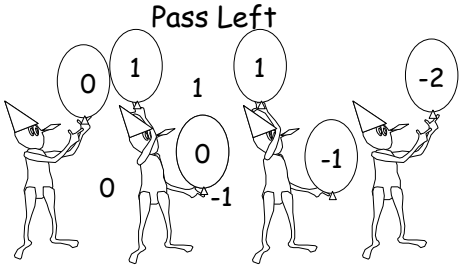
Invite n people to add two n-trit numbers.
Assign one person to each trit position.

An Addition Party



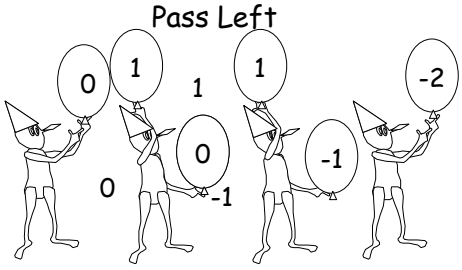
Each person should add the two input trits in their possession.
Problem: 2 and -2 are not allowed in the final answer.

Pass Left



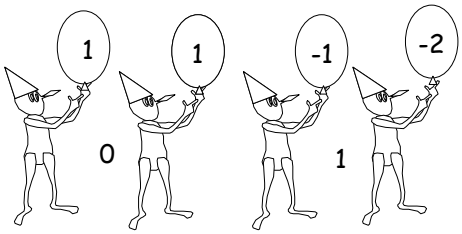
If you have a 1 or a 2
subtract 2 from yourself and pass a 1 to the left.
(Nobody keeps more than 0)

Pass Left



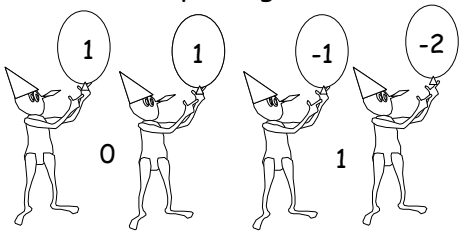
Add in anything that is given to you from the right.
(Nobody has more than a 1)

Pass Left



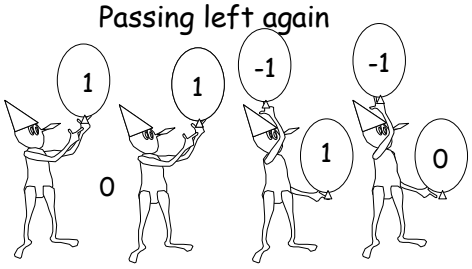
Add in anything that is given to you from the right.
(Nobody has more than a 1)

After passing left



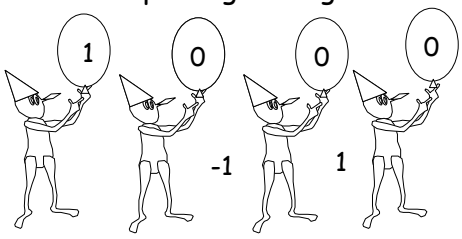
There will never again be any 2s because
everyone had at most 0
and received at most 1 more

Passing left again

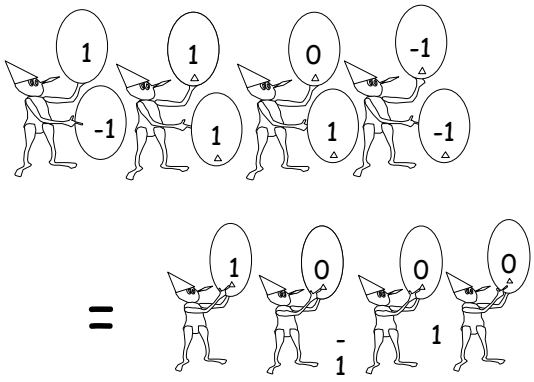


If you have a -1 or -2
add 2 to yourself and pass a -1 to the left
(Nobody keeps less than 0)

After passing left again



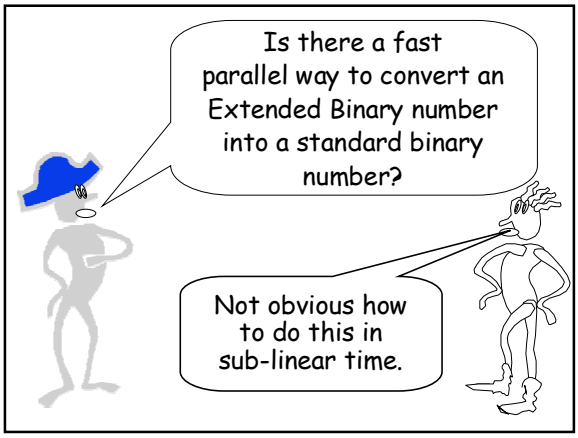
No -2s anymore either.
Everyone kept at least 0 and received at most -1.

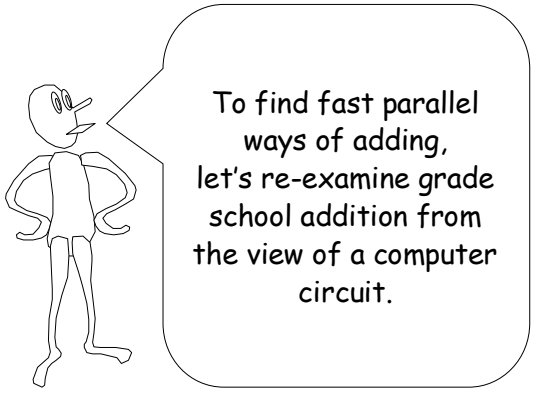


How about standard binary?

X in Extended Binary	X in Binary
Y in Extended Binary	Y in Binary
(X+Y) in Ext. Bin.	(X+Y) in Ext. Bin.
	???
	(X+Y) in Binary

Can we convert $(X+Y)_{ExtBin}$ into Binary fast?





Grade School Addition

$$\begin{array}{r} 1011111100 \\ + \quad 1011111101 \\ \hline 10100000011 \end{array}$$

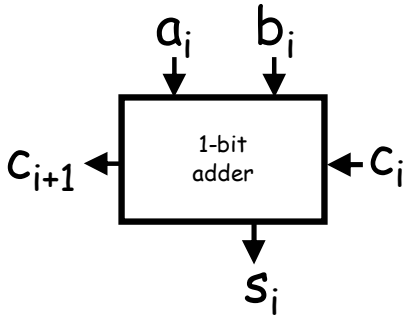
Grade School Addition

$$\begin{array}{r} c_5\ c_4\ c_3\ c_2\ c_1 \\ + \quad a_4\ a_3\ a_2\ a_1\ a_0 \\ \quad\ b_4\ b_3\ b_2\ b_1\ b_0 \\ \hline \end{array}$$

Grade School Addition

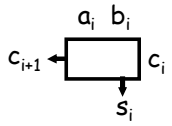
$$\begin{array}{r} c_5\ c_4\ c_3\ c_2\ \boxed{c_1} \\ + \quad a_4\ a_3\ a_2\ \boxed{a_1}\ a_0 \\ \quad\ b_4\ b_3\ b_2\ \boxed{b_1}\ b_0 \\ \hline \qquad\qquad\ \ s_1 \end{array}$$

Adder



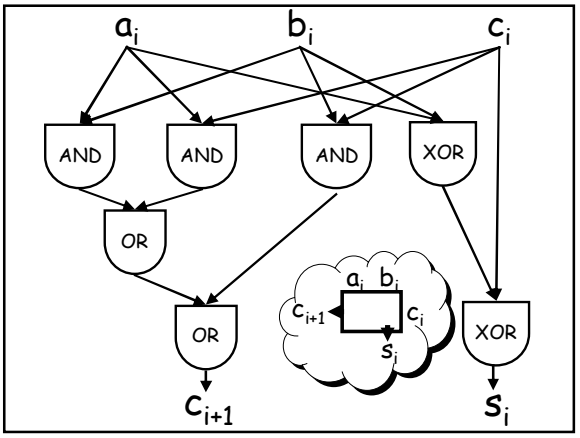
Logical representation of binary:
0 = false, 1 = true

$$\begin{array}{r} c_5c_4c_3c_2\ \boxed{c_1} \\ + \quad a_4a_3a_2\ \boxed{a_1}\ a_0 \\ \quad\ b_4b_3b_2\ \boxed{b_1}\ b_0 \\ \hline \qquad\qquad\ \ s_1 \end{array}$$

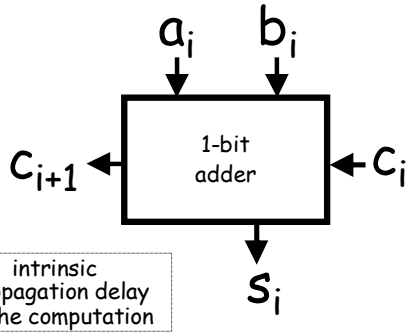


$s_1 = (a_1 \text{ XOR } b_1) \text{ XOR } c_1$ — odd number of bits are 1.

$c_2 = (a_1 \text{ AND } b_1)$ OR $(a_1 \text{ AND } c_1)$ OR $(b_1 \text{ AND } c_1)$ — at least two of the bits are 1.

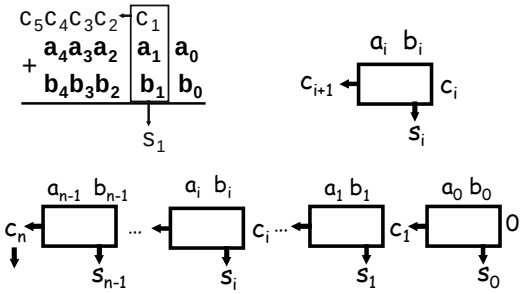


Adder

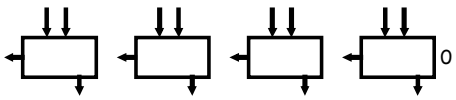


intrinsic propagation delay in the computation

Ripple-carry adder

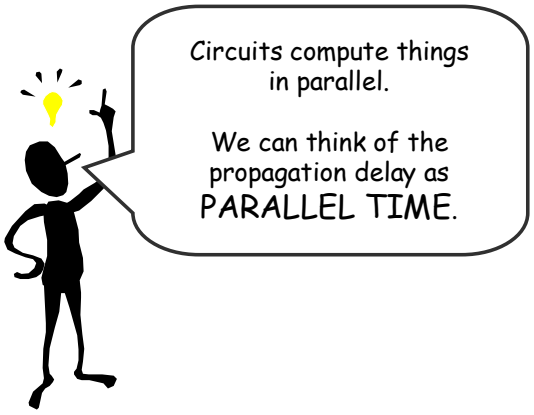


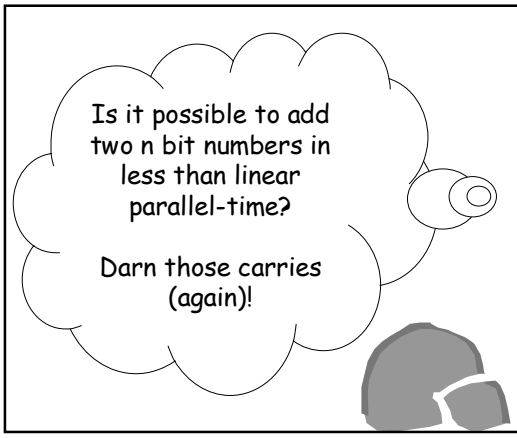
Ripple-carry adder



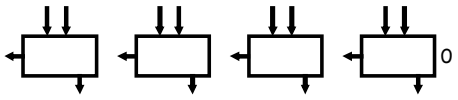
How long to add two n bit numbers?

Propagation time through the ripple-carry adder will be $\Theta(n)$



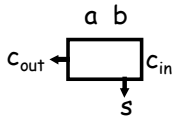


If we knew the carries it would be very easy to do fast parallel addition



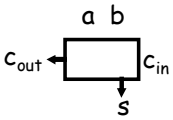
So how do we figure out the carries fast?

What do we know about the carry-out before we know the carry-in?



a	b	C_{out}
0	0	0
0	1	C_{in}
1	0	C_{in}
1	1	1

What do we know about the carry-out before we know the carry-in?



a	b	C_{out}
0	0	0
0	1	←
1	0	←
1	1	1

This is just a function of a and b. We can do this in parallel.

a	b	C_{out}
0	0	0
0	1	←
1	0	←
1	1	1

Idea #1: do this calculation first.

$$\begin{array}{r} 10\leftarrow\leftarrow\leftarrow\leftarrow1\leftarrow0 \\ + \quad 1011111101 \\ \hline 1000000110 \end{array}$$

a	b	C_{out}
0	0	0
0	1	←
1	0	←
1	1	1

Note that this just took one step!

Now if we could only replace the ← by 0/1 values…

Idea #1: do this calculation first.

$$\begin{array}{r} 10\leftarrow\leftarrow\leftarrow\leftarrow1\leftarrow00 \\ + \quad 1011111101 \\ \hline 1000000110 \end{array}$$

Idea #1: do this calculation first.

$$\begin{array}{r} 10\leftarrow\leftarrow\leftarrow\leftarrow1000 \\ + \quad 1011111101 \\ \hline 1000000110 \end{array}$$

Idea #1: do this calculation first.

$$\begin{array}{r} 10_{\leftarrow}{}_{\leftarrow}{}_{\leftarrow}{}_{\leftarrow}11000 \\ 1011111101 \\ + \quad 1000000110 \\ \hline \end{array}$$

Idea #1: do this calculation first.

$$\begin{array}{r} 10_{\leftarrow}{}_{\leftarrow}{}_{\leftarrow}111000 \\ 1011111101 \\ + \quad 1000000110 \\ \hline \end{array}$$

Idea #1: do this calculation first.

$$\begin{array}{r} 10_{\leftarrow}{}_{\leftarrow}1111000 \\ 1011111101 \\ + \quad 1000000110 \\ \hline \end{array}$$

Idea #1: do this calculation first.

$$\begin{array}{r} 10_{\leftarrow}11111000 \\ 1011111101 \\ + \quad 1000000110 \\ \hline \end{array}$$

Idea #1: do this calculation first.

$$\begin{array}{r} 10111111000 \\ 1011111101 \\ + \quad 1000000110 \\ \hline \end{array}$$

Idea #1: do this calculation first.

$$\begin{array}{r} 10111111000 \\ 1011111101 \\ + \quad 1000000110 \\ \hline 10100000011 \end{array}$$

Once we have the carries, it takes only one more step:
$s_i = (a_i \text{ XOR } b_i) \text{ XOR } c_i$

Steps to add two numbers

Compute the carry in terms of 0, 1, ←.

Convert this into regular binary carry.

Add X, Y and carry in one step.

$10\leftarrow\ \leftarrow\leftarrow\leftarrow 1\leftarrow 0$

So, everything boils down to: can we find a fast parallel way to convert each position to its final 0/1 value?

Called the "parallel prefix problem"

Idea #2:

Can think of $10\leftarrow\ \leftarrow\leftarrow\leftarrow 1\leftarrow 0$ as all partial results in:

(1 M (0 M (← M (← M (← M (← M (← M (1 M (← M 0)))))))))

for the operator **M**:

←**M** x = x

1 **M** x = 1

0 **M** x = 0

M	0	1	←
0	0	0	0
1	1	1	1
←	0	1	←

Idea #2 (cont):

And, the **M** operator is associative.

$$10\leftarrow\ \leftarrow\leftarrow\leftarrow 1\leftarrow 0$$

$$(\leftarrow \mathbf{M} (\leftarrow \mathbf{M} (\leftarrow \mathbf{M} (1 \mathbf{M} (\leftarrow \mathbf{M}\ 0)))))$$
$$=$$
$$(\leftarrow \mathbf{M} \leftarrow) \mathbf{M} (\leftarrow \mathbf{M}\ 1) \mathbf{M} (\leftarrow \mathbf{M}\ 0)$$
$$=$$
$$\leftarrow \mathbf{M}\ 1 \mathbf{M}\ 0 \ = \ 1$$

We'll just use fact that we have an Associative, Binary Operator

Binary Operator: an operation that takes two objects and returns a third.

- A♠B = C

Associative:

- (A ♠ B) ♠ C = A ♠ (B ♠ C)

Examples

- Addition on the integers
- Min(a,b)
- Max(a,b)
- Left(a,b) = a
- Right(a,b) = b
- Boolean AND
- Boolean OR
- **M**

In what we are about to do "+" will mean an arbitrary binary associative operator.

Prefix Sum Problem

Input: $X_{n-1}, X_{n-2},\ldots,X_1, X_0$
Output: $Y_{n-1}, Y_{n-2},\ldots,Y_1, Y_0$
where
$Y_0 = X_0$
$Y_1 = X_0 + X_1$
$Y_2 = X_0 + X_1 + X_2$
$Y_3 = X_0 + X_1 + X_2 + X_3$

$\vdots$

$Y_{n-1} = X_0 + X_1 + X_2 + X_3 + \ldots + X_{n-1}$

Prefix Sum Problem

Input: $X_{n-1}, X_{n-2},\ldots,X_1, X_0$
Output: $Y_{n-1}, Y_{n-2},\ldots,Y_1, Y_0$
where
$Y_0 = X_0$
$Y_1 = X_0 + X_1$
$Y_2 = X_0 + X_1 + X_2$
$Y_3 = X_0 + X_1 + X_2 + X_3$

$\vdots$

$Y_{n-1} = X_0 + X_1 + X_2 + X_3 + \ldots + X_{n-1}$

6, 9, 2, 3, 4, 7
31, 25, 16, 14, 11, 7
+ is Addition

Prefix Sum Problem

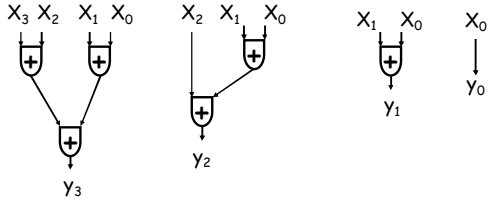
Input: $X_{n-1}, X_{n-2},\ldots,X_1, X_0$
Output: $Y_{n-1}, Y_{n-2},\ldots,Y_1, Y_0$
where
$Y_0 = X_0$
$Y_1 = X_0 + X_1$
$Y_2 = X_0 + X_1 + X_2$
$Y_3 = X_0 + X_1 + X_2 + X_3$

$\vdots$

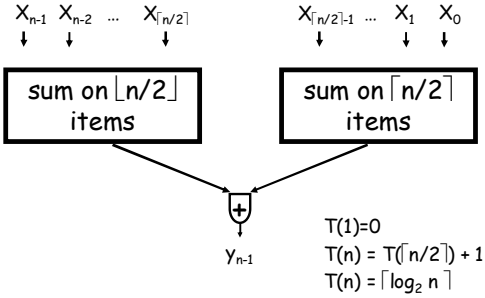
$Y_{n-1} = X_0 + X_1 + X_2 + X_3 + \ldots + X_{n-1}$

0, ←, ←, 1, ←, ←, 0
0, 1, 1 , 1, 0, 0, 0
+ is **M**

Example circuitry (n = 4)



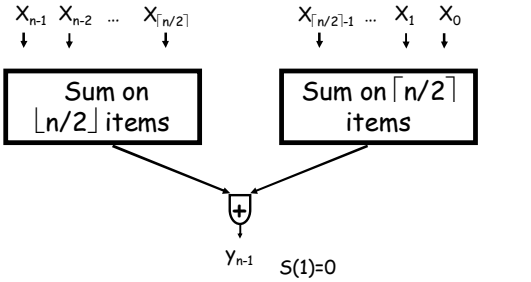
Divide, conquer, and glue for computing y_{n-1}



The parallel time taken is $T(n) = \lceil \log_2 n \rceil$!

But how many components does this use? What is the size of the circuit?

Size of Circuit (number of gates)

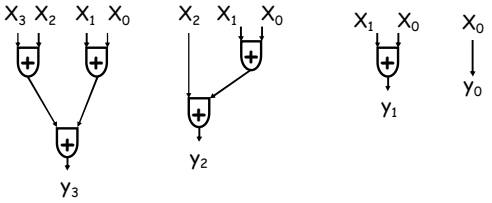


$S(1)=0$

$S(n) = S(\lceil n/2 \rceil) + S(\lfloor n/2 \rfloor) + 1$

$S(n) = n-1$

Sum of Sizes



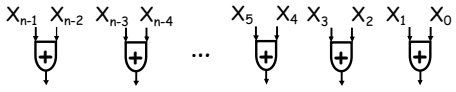
$S(n) = 0 + 1 + 2 + 3 + \ldots + (n-1) = n(n-1)/2$

This algorithm is fast, but it uses too many components!

Modern computers do something slightly different.

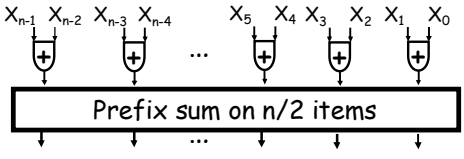
Recursive Algorithm
n items (n = power of 2)

If $n = 1$, $Y_0 = X_0$; else



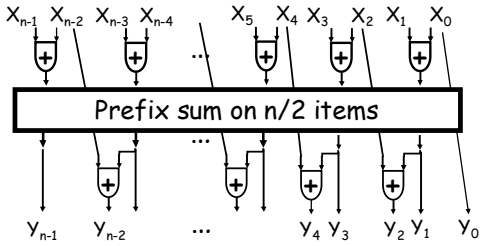
Recursive Algorithm
n items (n = power of 2)

If $n = 1$, $Y_0 = X_0$; else



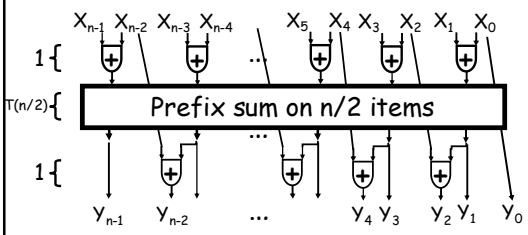
Recursive Algorithm
n items (n = power of 2)

If n = 1, $Y_0 = X_0$; else



Parallel time complexity

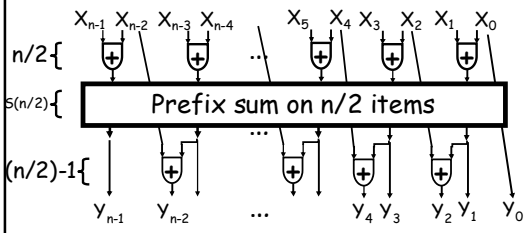
If n = 1, $Y_0 = X_0$;



T(1)=0; T(2) = 1; T(n) = T(n/2) + 2
$T(n) = 2 \log_2(n) - 1$

Size

If n = 1, $Y_0 = X_0$;



S(1)=0; S(n) = S(n/2) + n - 1
$S(n) = 2n - \log_2 n - 2$

Putting it all together: "Carry Look-Ahead Addition"

To add two n-bit numbers: a and b

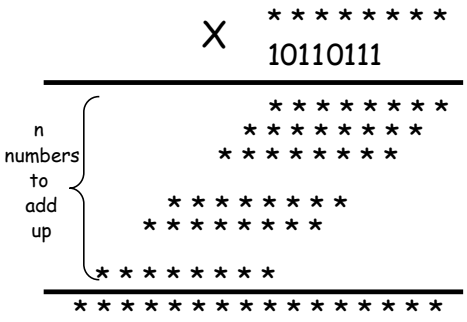
- 1 step to compute carries using ($\leftarrow 01$)
- $2\log_2 n - 1$ steps to compute binary carries c
- 1 step to compute c XOR (a XOR b)

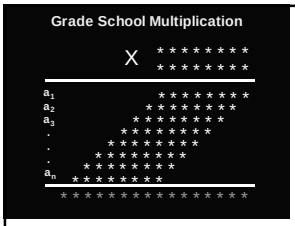
$2 \log_2 n + 1$ steps total

Addition can be done in O(log n) parallel time, with only O(n) components!

What about multiplication?

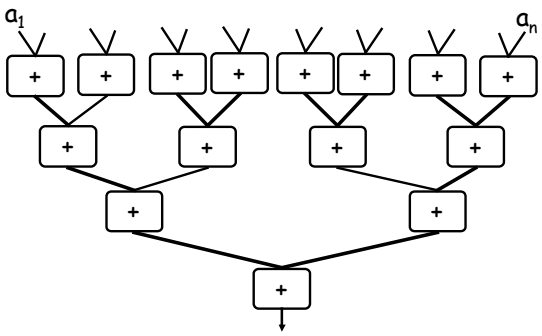
How about multiplication?





We need to add n 2n-bit numbers:
$a_1, a_2, a_3, \ldots, a_n$

Adding these numbers in parallel



What is the depth of the circuit?

Each addition takes O(log n) parallel time

Depth of tree = $\log_2 n$

Total $O(\log n)^2$ parallel time

Can we do better?

How about O(log n) parallel time?

How about multiplication?

Here's a really neat trick:

Let's think about how to add 3 numbers to make 2 numbers.

"Carry-Save Addition"

The sum of three numbers can be converted into the sum of 2 numbers in constant parallel time!

$$\begin{array}{r} 1100111011 \\ + \; 1011111101 \\ + \; 1000000110 \\ \hline \end{array}$$

"Carry-Save Addition"

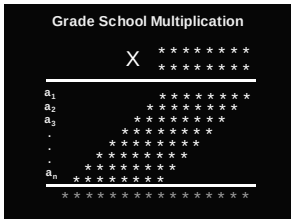
The sum of three numbers can be converted into the sum of 2 numbers in constant parallel time!

```
  1100111011
+ 1011111101
+ 1000000110
-----------
+ 1111000000   ← XOR
  1000111110   ← Carries
```

Grade School Multiplication

```
        X  ********
           10110111
-----------------------
           ********
          ********
         ********
       ********
      ********
     ********
-----------------------
 ****************
```

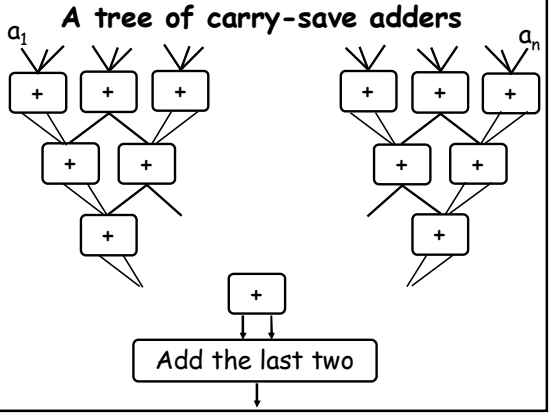
n numbers to add up



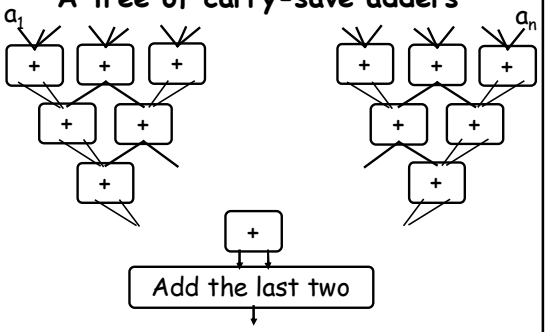
We need to add n 2n-bit numbers:

$a_1, a_2, a_3, \ldots, a_n$

A tree of carry-save adders

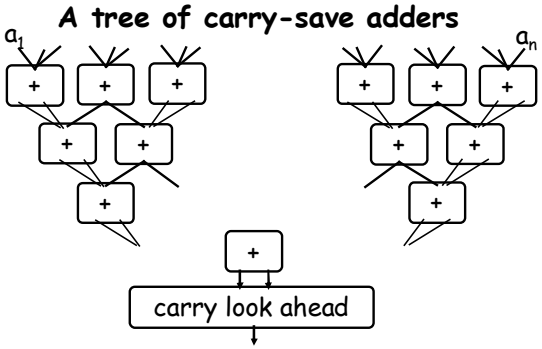


A tree of carry-save adders



$T(n) \approx \log_{3/2}(n) + [\text{last step}]$

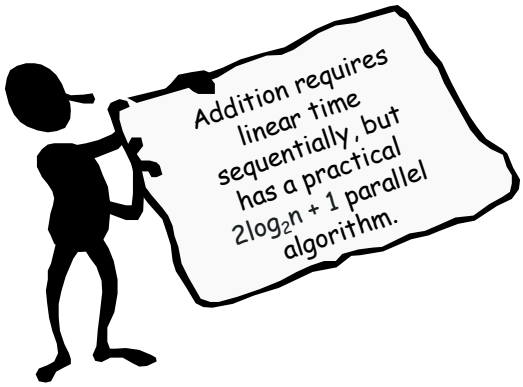
A tree of carry-save adders

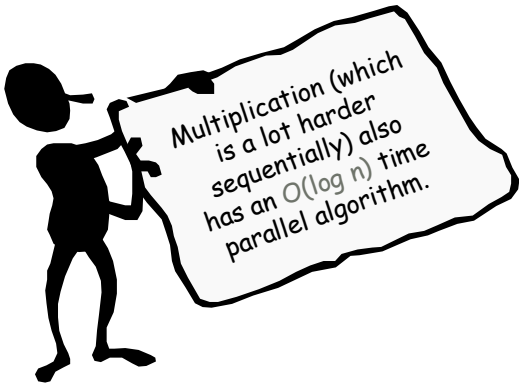


$T(n) \approx \log_{3/2}(n) + 2\log_2 2n + 1$

We can multiply in $O(\log n)$ parallel time too!

For a 64-bit word that works out to a parallel time of 22 for multiplication, and 13 for addition.



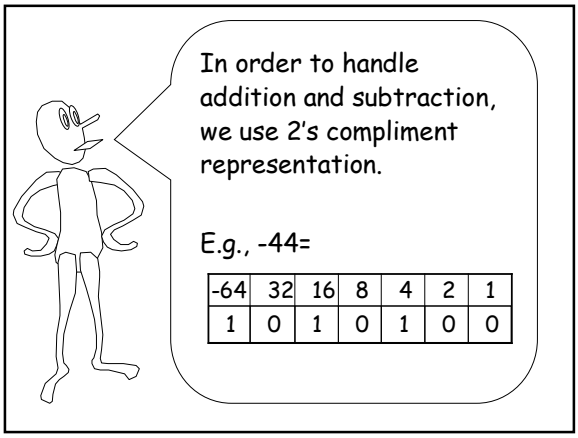


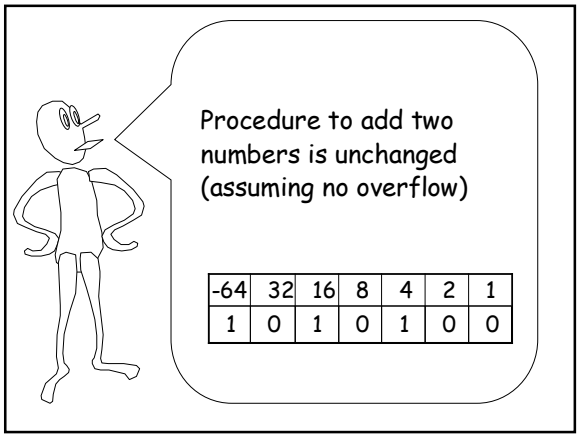
And this is how addition works on commercial chips…..

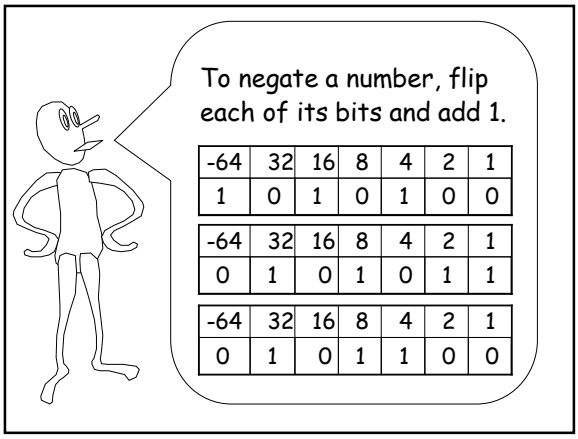
Processor	n	$2\log_2 n + 1$
80186	16	9
Pentium	32	11
Alpha	64	13

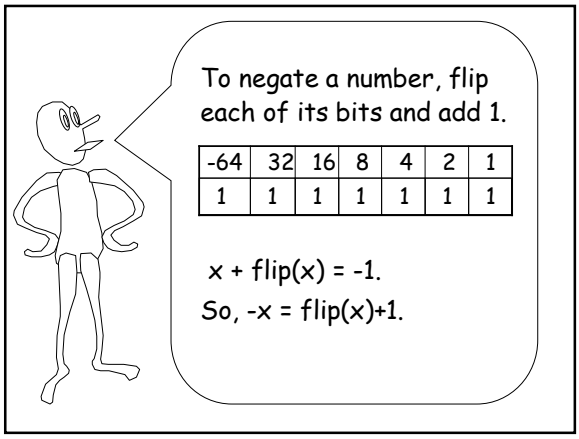
Excellent!
Parallel time for:
Addition = $O(\log n)$
Multiplication = $O(\log n)$

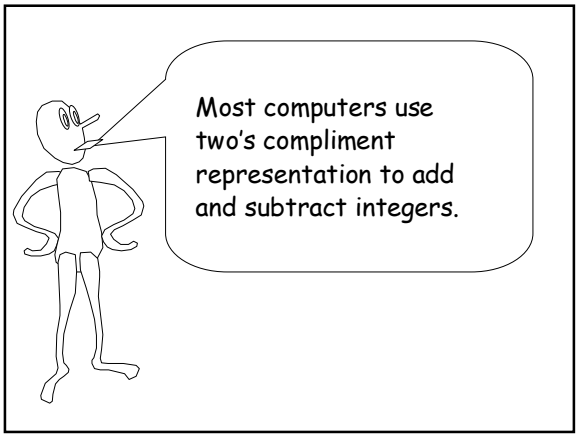
Hey, we forgot subtraction!

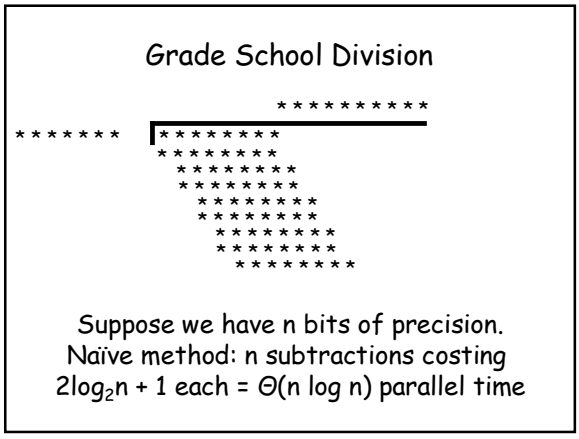

In order to handle addition and subtraction, we use 2's compliment representation.
E.g., -44=
-64 32 16 8 4 2 1
1 0 1 0 1 0 0


Procedure to add two numbers is unchanged (assuming no overflow)
-64 32 16 8 4 2 1
1 0 1 0 1 0 0


To negate a number, flip each of its bits and add 1.
-64 32 16 8 4 2 1
1 0 1 0 1 0 0
-64 32 16 8 4 2 1
0 1 0 1 0 1 1
-64 32 16 8 4 2 1
0 1 0 1 1 0 0


To negate a number, flip each of its bits and add 1.
-64 32 16 8 4 2 1
1 1 1 1 1 1 1
x + flip(x) = -1.
So, -x = flip(x)+1.


Most computers use two's compliment representation to add and subtract integers.


Grade School Division
****
******* ********
****
****
****
****
****
****
****
****
Suppose we have n bits of precision.
Naïve method: n subtractions costing
2log2n + 1 each = Θ(n log n) parallel time

Let's see if we can reduce to O(n) by being clever about it.

Idea: use extended binary all through the computation!

Then convert back at the end.

SRT division algorithm

```
                1  1 -1  1  0   r -1 -1  1
1 0 1 1 | 1 1 1 0 1 1 0 1
        -1 0 -1 -1
          1  0 -1  1
         -1  0 -1 -1
             -2  0
          = -1  0  0  1
            1  0  1  1
               1  2
          =  1  0  0  0
            -1  0 -1 -1
               0 -1 -1  1
```

Rule: Each bit of quotient is determined by comparing first bit of divisor with first bit of dividend. Easy!

Time for n bits of precision in result:

$$\approx \underbrace{3n}_{\text{1 addition per bit}} + \underbrace{2\log_2(n) + 1}_{\text{Convert to standard representation by subtracting negative bits from positive.}}$$

```
   21 r 6        22 r -5
11 | 237       11 | 237
```

Intel Pentium division error

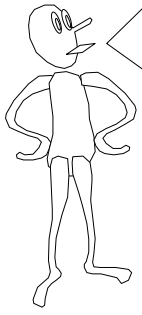
The Pentium uses essentially the same algorithm, but computes more than one bit of the result in each step.

Several leading bits of the divisor and quotient are examined at each step, and the difference is looked up in a table.

The table had several bad entries.

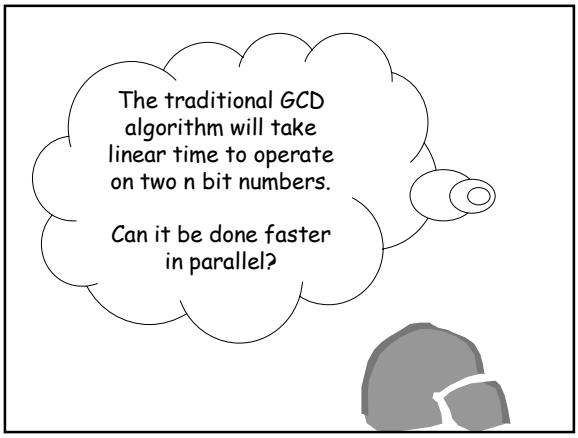
Ultimately Intel offered to replace any defective chip, estimating their loss at $475 million.

If I had millions of processors, how much of a speed-up might I get over a single processor?

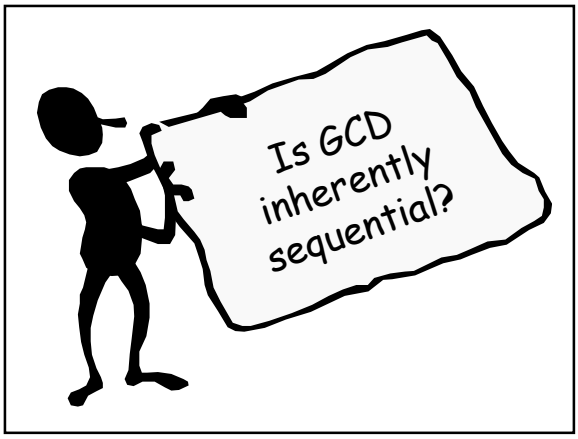


Brent's Law

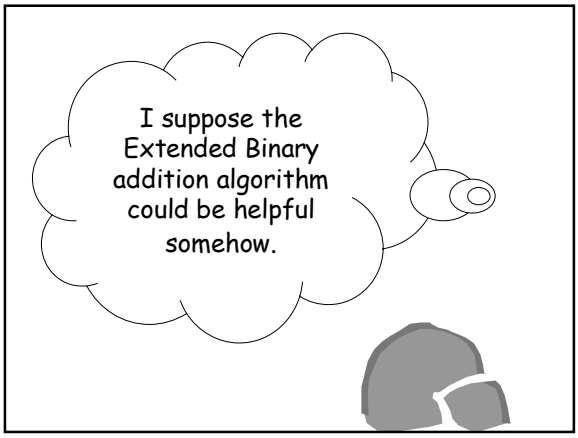
At best, p processors will give you a factor of p speedup over the time it takes on a single processor.

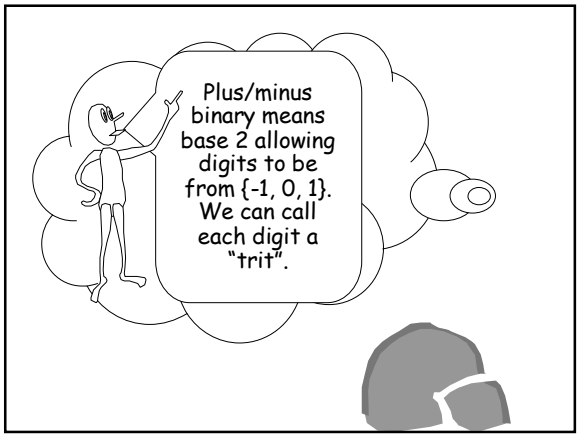


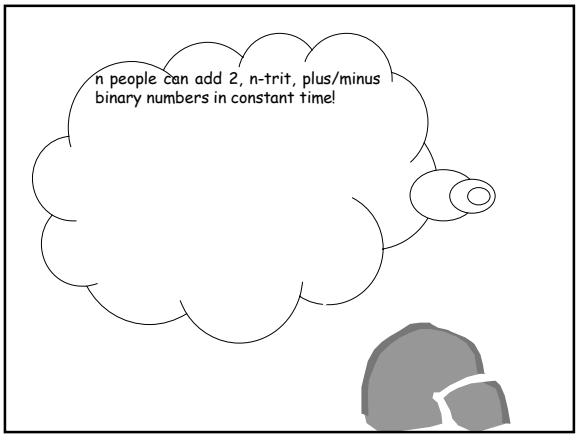
If n^2 people agree to help you compute the GCD of two n bit numbers, it is not obvious that they can finish faster than if you had done it yourself.

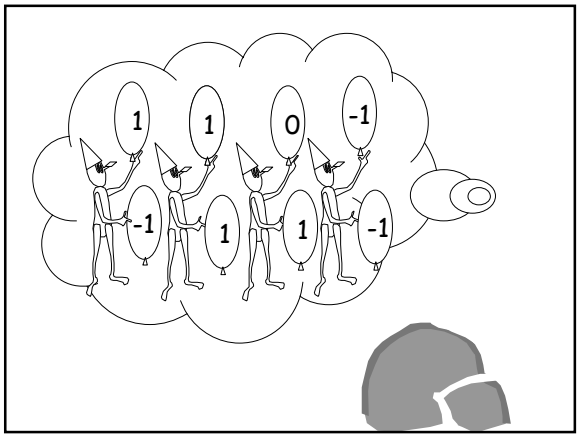


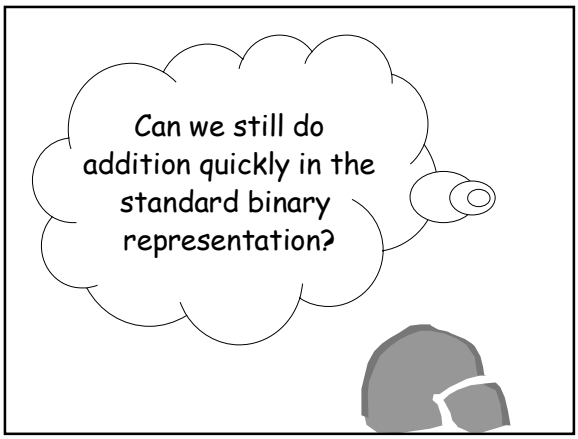
No one knows.






n people can add 2, n-trit, plus/minus binary numbers in constant time!


1
1
0
-1
-1
1
1
-1


Can we still do addition quickly in the standard binary representation?