Algovision
Arithmetic Algorithms: Addition

Operands

In this tour, you will see how to add two binary numbers as fast as possible.

Both the upper and the lower operands are written in the usual way:
the least significant digits in the rightmost column,
the most significant digits in the lefttmost column.

The first scene shows just two operands; you will learn how to manipulate with them:

- click any box representing an operand digit; the corresponding digit changes its value;
- or click one of the $\boxed{\text{RND}}$ buttons: the corresponding operand is set to a random number;
- or click one of the $\boxed{\text{CMPL}}$ buttons: the corresponding operand is set to the complement of the other one (which means that in each column there is one 0 and one 1).

Using checkbox $\boxed{\text{Column Nr}}$ you can get column numbers for better orientation.

You can change the number of digits of the operands by the menu $\boxed{\text{Bit Width}}$ (as usual is Computer Science, the choice is limited to certain powers of two: 4, 8, 16, 32, 64, 128).

Elementary Algorithm

Use the $\boxed{\text{Step}}$ button to add two binary numbers using the standard elementary school algorithm.

The carry is shown in an obvious way:

- a black/gray dot means no carry;

- a red/pink triangle, pointing to the higher order column shows a carry in the column

Moreover, a red triangle is used in columns, where carry is generated, no matter if another carry comes from the previous column (i.e., two 1's in the column).

Similarly, a black dot is used in columns, where carry is killed, even if another carry comes from the previous column (i.e., two 0's in the column).

Gray and pink colors are used in complementary columns (one 0, one 1) that would propagate carry from the previous column, but too week to generate carry by itself.

The algorithm is slow - the number of time steps is equal to the number of digits of the operands. Try addition of 64 or 128 digit operands to see that a faster algorithm is often needed.

Parallel Elementary

Speeding up can be obtained by parallel processing:
an adder is able to perform several operations in the same time.

Even though it seems at the first moment that addition is inherently sequential:
to perform operations in a column, one has to wait the result (carry or no carry) of the previous column this is not true.

In columns where the digits are 1-1 or 0-0, we do not need to wait:
In the former case (visualized by a red triangle), carry is always generated,
in the latter case (visualized by a black dot), carry is never generated.

Step through the computation: in the first step, carry is determined in 1-1 and 0-0 columns, and then we proceed sequentially through the 'islands' of complementary (1-0 or 0-1) columns.

Try to get random values operands again and again (using $\boxed{\text{RND}}$ buttons) and add them and you will see that the addition is finished much faster then in the previous scene, because of simultaneous performing of several operation in the same time.

Parallel Worst Case

Unfortunately, if one operand is a complement of the other one (columns 1-0 or 0-1 only), then the paralelized algorithm of the preceeding scene is as bad as the serial elementary algorithm.

Try to generate one operand in random using its $\boxed{\text{RND}}$ button, and them making the other operaand complementary by clicking its $\boxed{\text{CMPL}}$ button and observe how slow the addition is in such a case.

In the next scenes it will be shown how to perform addition in a way that is very fast even in the *worst* case.

Blocks

The main idea to speed up the addition is as follows:

Imagine we want to know whether the most significant (the leftmost) column generates carry. The elementary algorithm waits until carry/no carry propagetes eventually into the leftmost column and this takes too much time.

Let us do it in a more clever way: During the time needed to process the right half of columns, let us start to work with the left half of the columns (higher significant ones).

There are two possibilities:

(a) in the left half of the columns, there is at least one column that is either 1-1 or 0-0.
In such a case we do not need to know whether the right half of the columns sends carry to the least significant column of the left half of columns and in the time when the right half of columns is processed we can already have the left half processed as well and the information about carry in the most significant column determined;

(b) the left half of colums is composed of complementary columns only (i.e., either 1-0 or 0-1)
in such a case,

- if the most significant (the leftmost) column of the right half of the columns sends carry to the least significant (the rightmost) column of the left half of columns,
then we know that this carry will be propagated through *all* columns of the left half of columns and in *one* more step we can decide that carry is present in all columns of the left half of columns, or
- if the most significant (the leftmost) column of the right half of the columns does not send carry to the least significant (the rightmost) column of the left half of columns,
then we know that no carry will appear in *any* columns of the left half of columns and in *one* more step we can decide that no carry is present in any column of the left half of columns.

A block is an interval of successive column of operand - a block is shown in the display.

Try to get other blocks using the button $\boxed{\text{Another Block}}$. You can also drag either the left or the right side of the block by the mouse.

The value of a block is either '+' or '−' or '<' sign shown is a small rectangle below the left side of the block and has the following meaning (where a complementary column is a column with either 1-0 or 0-1 pair of digits):

- + the leftmost non-complementary column of the block is 1-1
- − the leftmost non-complementary column of the block is 0-0

- $<$ all columns of the block are complementary

Keep changing the block by the button $\boxed{\text{Another Block}}$ or dragging its sides by the mouse and/or keep changing values of operands either by buttons $\boxed{\text{RND}}$ and $\boxed{\text{CMPL}}$ of by changing a digit by the mouse and observe that the value of the block follows the above definition.

The use of the value of a block is the following:
using information about operands digits included in the block only (note that the digits outside the block are 'low-lighted') determine whether the block sends carry to the column just left to it:

- if $+$ is the value of the block, then the leftmost 1-1 column of the block generates carry that is propagated by possible complementary columns that are left to it and is eventually sent to the left neighbor column of the block;
- if $-$ is the value of the block, then the leftmost 0-0 column of the block does not generate carry and possibly absorbs carry from its right (lower significant) neighbor and possible complementary columns that are left to it are to weak to generate carry by themselves; this is why no carry is sent to the left neighbor column of the block;
- if $<$ is the value of the block, then there are just complementary columns in the block that are to weak to generate carry by themselves
but if the carry enters the block from the column that is the right neighbor of the block (if it exists), then the carry is propagated through the block and send to the left neighbor of the block
however, if no carry enters the block from the right, no carry leaves the block to the left neighbor of the block.

What was explained in the first page of the explanation of the present scene can be rephrased as follows:
in the same time, determine values of the block consisting of the left half of the columns and the block consisting of the right half of the columns

- If the value of the left block is $+$, the most significant column generates carry no matter what is the value of the right block;
- If the value of the left block is $-$, the most significant column does not generate carry no matter what is the value of the right block;
- If the value of the left block is $<$, the most significant column generates carry if and only if carry comes from the right block which happens if and only if the value of the right block is $+$ (not that the block consisting of the right half of the columns has no right neighbor column and hence if its value is $<$, there is no carry to be propagated).

Thus, if we are able to determine values of both blocks simultaneously, than at most one more step is sufficient to detemine carry of the most significant column.

We can proceed in a similar way to determine quickly carry in other columns as well.

One Column

The value of a block can be determined as defined in the previous scene. However, we will see that is faster to determine the value of a block recursively, see the following scenes,

The present scene shows the bottom of such recursion - one column block,

The value of a one column block is obvious:

- A 1-1 block has value $+$,
- a 0-0 block has value $-$, and
- a complementary block (either 1-0 or 0-1) has value $<$

Play with the block and digit values to see values of one column blocks.

Random Split

The present scene shows how to determine the value of a larger block in few parallel steps:

Split the block you see into two parts by clicking the button $\boxed{\text{Step}}$ in the group Construction.

Determine the values of both subblocks by clicking the button $\boxed{\text{Step}}$ in the group Computation.

The value of the whole block is determined as follows:

- If the value of the left block is $+$ or $-$, then the value of the whole block is the same;
- If the value of the left block is $<$, then the value of the whole block is equal to the value of the right block

The proof is simple: note that the value of the block is determined by its leftmost non-complementary column.

If leftmost non-complementary column belongs to the left subblock, the values of the left subblock and the whole block are the same.

If the left block consist of complementary columns only, then the determining non-complementary column of the block belongs to the right subblock (or does not exist) and the values of the right subblock and the whole block are the same.

Click the button $\boxed{\text{Step}}$ in the group Computation once more to see that it works in this way.

The values of the subblocks are determined recursively in the same way.

Keep clicking the button $\boxed{\text{Step}}$ of the Construction group until only one-column blocks are to be evaluated. Then, keep clicking the button $\boxed{\text{Step}}$ in the group Computation to see how values of larger blocks are computed from values of smaller blocks.

Check the checkbox Wide to get another graphical view of the evaluation tree.

Keep playing with the root block, operand digit values, an evaluation tree construction and block evaluation.

The way a block is split into subblocks is not important for the function of the algorithm, but it is quite important for efficiency of computation.

It is clear that subblocks can be processed sinultaneously, because processing operates on two completely disjoint and independent collections of data. Thus, the number of recursive parallel steps to evaluate a block is given by the depth of the evaluation tree.

Different advantageous (and also bad) strategies of splitting blocks are presented in scenes that follow.

Worst Split

The worst possible way of splitting (from the point of view of the height of the evaluation tree) is to separate eight the leftmost or the rightmost column. Try it!

Best Split

Of course, the best splitting strategy (from the point of view of the height of the evaluation tree) is to split the block into two parts of the same size (or almost the same size if the number of columns is odd).

The adders that will be explained later evaluate a collection of blocks of different sizes in parallel. While the largest among such blocks should be split in the most efficient way, shorter blocks need not be split optimally, provided their evaluation tree is not higher than the tree of larger blocks.

It is convenient to split short blocks in a way that is somehow compatible with larger blocks; different splitting methods used in known adders are shown in the forthcomming scenes.

Kogge-Stone Split

It is recommended to check Column Nr to get the column numbered. By $[H, L]$ we mean the block with the leftmost column numbered H and the rightmost column numbered L.

Kogge-Stone adder uses the following method of splitting of a block $[H, L]$ such that $H > L$: find the largest S such that

$$S \text{ is a power of 2,}$$

$$H - S \geq L \text{ , and}$$

and split the box to $[H, H - S + 1]$ and $[H - S, L]$

It is clear that H> $H - S \geq L$. Moreover, since $S = 1$ verifies all the conditions, S verifying the condition exists.

Ladner-Fischer Split

Ladner-Fischer adder uses the following method of splitting of a block $[H, L]$ such that $H > L$ (in the following, n is the width of the operands, i.e., the number of all columns and it is supposed that n is a power of 2):

- If $H \geq n/2 > L$, then split the block to $[H, N/2]$ and $[N/2 - 1, L]$
- else if $H \geq kn/4 > L$ for either $k = 1$ or $i = 3$
  (only one k verifies if the previous in not applicable),
  and if successfull, split the block to $[H, kn/4]$ and $[kn/4 - 1, L]$;
- else if $H \geq kn/8 > L$ for some $k = 1, 3, 5, 7$
  (only one k verifies if the previous in not applicable),
  and if successfull, split the block to $[H, kn/8]$ and $[kn/8 - 1, L]$
- else if $H \geq kn/16 > L$ for some $k = 1, 3, 5, 7, 9, 11, 13, 15$
  (only one k verifies if the previous in not applicable),
  and if successfull, split the block to $[H, kn/16]$ and $[kn/16 - 1, L]$
- etc.

In other words,

- if possible, split the block in the half of the columns;
- else try to split all columns to quarters,
  if one split line splits the block, use it;
- else split all columns to 8 equal parts,
  if one split line splits the block, use it;
- etc.

Brent-Kung Split

Brent-Kung adder uses the following method of splitting of a block $[H, L]$ such that $H > L$: find the largest S such that

$$S \text{ is a power of 2,}$$

$$H - S \geq L \text{ , and}$$

$$S \text{ divides } (H + 1);$$

and split the box to $[H, H - S + 1]$ and $[H - S, L]$

It is clear that H> $H - S \geq L$. Moreover, since $S = 1$ verifies all the conditions, S verifying the conditon exists.

In other words, Brent-Kung works as follows depending on $H + 1$

- if $H + 1$ is odd, detach the leftmost column
  (the right subblock has even number of columns);
- if $H + 1$ is even, but not divisible by 4, detach two leftmost column
  (the right subblock has the number of columns divisible by 4)
- if $H + 1$ is divisible by 4, but not divisible by 8, detach 4 leftmost column
  (the right subblock has the number of columns divisible by 8)
- if $H+1$ is divisible by 8, but not divisible by 16, detach 8 leftmost column
  (the right subblock has the number of columns divisible by 16)
- in general,
  if $H + 1$ is divisible by 2^k, but not divisible by 2^{k+1}, detach 2^k leftmost column
  (the right subblock has the number of columns divisible by $2^(k + 1)$)

Han-Carlson Split

Han-Carlson adder uses a combination of Brent-Kung and Kogge-Stone

When splitting a block $[H, L]$,

- If H is even, then split the block to $[H, H]$ (one column) and $[H - 1, L]$, and
- if H is odd, then split the block using Kogge-Stone method.

Note that is H is even, then $H + 1$ is odd, and no non-trivial power of 2 divides $H + 1$. In this case (see previous scene), Brent-Kung splits to $[H, H]$ and $[H - 1, L]$

Kogge-Stone: Single Column

If we know determining values of blocks, it is easy to determine which columns send carry:

For the column that is numbered H, consider the block $[H, 0]$:

- if the value of the block $[H, 0]$ is $+$, the column H sends carry
- if the value of the block $[H, 0]$ is $-$, the column H does not send carry
- if the value of the block $[H, 0]$ is $<$, the column H would propagate carry comming to it from the right; however there is no column right to $[H, 0]$, no carry to propagate, hence H does not send carry.

In this scene, for one number H (initially the number of the leftmost column) the evaluation tree of the block $[H, 0]$, built using the Kogge-Stone splitting strategy, is shown.

Clock to some column number box (the upper row of boxes) to change H. The construction of the tree and evaluation are not shown.

Kogge-Stone: Overlapping

This scene is very similar to the previous one, but the evaluation trees of several blocks of the form $[H, 0]$ are shown.

Click to some column number box (the upper row of boxes) to select/deselect another H, or click the button $\boxed{\text{All}}$ or $\boxed{\text{Clear}}$ to select/deselect all columns.

You can see that trees for different blocks overlap and therefore if we need the evaluation trees for all blocks $[H, 0]$, we do not need to build them separately, but we could recycle parts of one tree to construct another one.

Kogge-Stone: Full Adder

This scene shows the full Kogge-Stone adder.

Look how the specific way of splitting makes the evaluation trees of different blocks $[H, 0]$ nicely overlapping.

The scene allows changing values of operands and computing carries by computing values of the blocks $[H, 0]$.

We can see that the number of parallel steps necessary to add two numbers is equal to the number of layers of the circuit, which is $1 + log_2 N$, where N is the number of the column

Kogge-Stone: Layered Adder

In the previous scene, we have seen that the path for certain signals in the circuit was shorter then the height of the cirguit. In fact, steps of the 'computation' were delayed with respect to how a real circuit would work.

In this scene, we added white “delay” boxes. They have just one input and their only function is to delay signals of shorter path to be synchronized with other signals.

Try to step the computation to see that the computation proceeds in a synchronized way by the layers of the circuit.

Kogge-Stone: Forgetting Adder

The scene is very similar to the previous one with computation layer by layer.

Note that if values in one layer are computed, the values in the previous layers are not any more needed. This is why this scene forgets the values in the previous layers.

Try the computation of block values to see that the values of a single layer are needed.

Kogge-Stone: Pipelined Adder

As it has been shown in the previous scene, addition of one pair of the operands really occupies just one layr of the circuit.

The layers that are not any more used by the present couple of operands can immediately be used to add another pair of operands. In this way, the circuit can process as many pairs of operands as is the number of its layers.

Try computation using the button $\boxed{\text{Step}}$. Values corresponding particular pairs of operands are distinguished by color.

Using this pipelined approach, the number of parallel steps necessary to add two numbers is equal to the number of layers of the circuit, but the rate in which operand pairs enter the circuit is one pair per step.

Ladner-Fischer: Single Column

If we know determining values of blocks, it is easy to determine which columns send carry:

For the column that is numbered H, consider the block $[H, 0]$:

- if the value of the block $[H, 0]$ is $+$, the column H sends carry;
- if the value of the block $[H, 0]$ is $-$, the column H does not send carry;
- if the value of the block $[H, 0]$ is $<$, the column H would propagate carry comming to it from the right;
  however there is no column right to $[H, 0]$, no carry to propagate, hence H does not send carry.

In this scene, for one number H (initially the number of the leftmost column) the evaluation tree of the block $[H, 0]$, built using the Ladner-Fischer splitting strategy, is shown.

Clock to some column number box (the upper row of boxes) to change H. The construction of the tree and evaluation are not shown.

Ladner-Fischer: Overlapping

This scene is very similar to the previous one, but the evaluation trees of several blocks of the form $[H, 0]$ are shown.

Click to some column number box (the upper row of boxes) to select/deselect another H, or click the button $\boxed{\text{All}}$ or $\boxed{\text{Clear}}$ to select/deselect all columns.

You can see that trees for different blocks overlap and therefore if we need the evaluation trees for all blocks $[H, 0]$, we do not need to build them separately, but we could recycle parts of one tree to construct another one.

Ladner-Fischer: Full Adder

This scene shows the full Ladner-Fischer adder.

Look how the specific way of splitting makes the evaluation trees of different blocks $[H, 0]$ nicely overlapping.

The scene allows changing values of operands and computing carries by computing values of the blocks $[H, 0]$.

We can see that the number of parallel steps necessary to add two numbers is equal to the number of layers of the circuit, which is $1 + log_2 N$, where N is the number of the column.

Ladner-Fischer: Layered Adder

In the previous scene, we have seen that the path for certain signals in the circuit was shorter then the height of the cirguit. In fact, steps of the 'computation' were delayed with respect to how a real circuit would work.

In this scene, we added white 'delay' boxes. They have just one input and their only function is to delay signals of shorter path to be synchronized with other signals.

Try to step the computation to see that the computation proceeds in a synchronized way by the layers of the circuit.

Ladner-Fischer: Forgetting Adder

The scene is very similar to the previous one with computation layer by layer.

Note that if values in one layer are computed, the values in the previous layers are not any more needed. This is why this scene forgets the values in the previous layers.

Try the computation of block values to see that the values of a single layer are needed.

Ladner-Fischer: Pipelined Adder

As it has been shown in the previous scene, addition of one pair of the operands really occupies just one layr of the circuit.

The layers that are not any more used by the present couple of operands can immediately be used to add another pair of operands. In this way, the circuit can process as many pairs of operands as is the number of its layers.

Try computation using the button $\boxed{\text{Step}}$. Values corresponding particular pairs of operands are distinguished by color.

Using this pipelined approach, the number of parallel steps necessary to add two numbers is equal to the number of layers of the circuit, but the rate in which operand pairs enter the circuit is one pair per step.

Brent-Kung: Single Column

If we know determining values of blocks, it is easy to determine which columns send carry:

For the column that is numbered H, consider the block $[H, 0]$:

- if the value of the block $[H, 0]$ is $+$, the column H sends carry;
- if the value of the block $[H, 0]$ is $-$, the column H does not send carry;
- if the value of the block $[H, 0]$ is $<$, the column H would propagate carry comming to it from the right;
  however there is no column right to $[H, 0]$, no carry to propagate, hence H does not send carry.

In this scene, for one number H (initially the number of the leftmost column) the evaluation tree of the block $[H, 0]$, built using the Brent-Kung splitting strategy, is shown.

Clock to some column number box (the upper row of boxes) to change H. The construction of the tree and evaluation are not shown.

Brent-Kung: Overlapping

This scene is very similar to the previous one, but the evaluation trees of several blocks of the form $[H, 0]$ are shown.

Click to some column number box (the upper row of boxes) to select/deselect another H, or click the button $\boxed{\text{All}}$ or $\boxed{\text{Clear}}$ to select/deselect all columns.

You can see that trees for different blocks overlap and therefore if we need the evaluation trees for all blocks $[H, 0]$, we do not need to build them separately, but we could recycle parts of one tree to construct another one.

Brent-Kung: Full Adder

This scene shows the full Brent-Kung adder.

Look how the specific way of splitting makes the evaluation trees of different blocks $[H, 0]$ nicely overlapping.

The scene allows changing values of operands and computing carries by computing values of the blocks $[H, 0]$.

We can see that the number of parallel steps necessary to add two numbers is equal to the number of layers of the circuit, which is $2 * log_2 N - 1$, where N is the number of the column.

Brent-Kung: Layered Adder

In the previous scene, we have seen that the path for certain signals in the circuit was shorter then the height of the cirguit. In fact, steps of the 'computation' were delayed with respect to how a real circuit would work

In this scene, we added white 'delay' boxes. They have just one input and their only function is to delay signals of shorter path to be synchronized with other signals.

Try to step the computation to see that the computation proceeds in a synchronized way by the layers of the circuit.

Brent-Kung: Forgetting Adder

The scene is very similar to the previous one with computation layer by layer.

Note that if values in one layer are computed, the values in the previous layers are not any more needed. This is why this scene forgets the values in the previous layers.

Try the computation of block values to see that the values of a single layer are needed.

Brent-Kung: Pipelined Adder

As it has been shown in the previous scene, addition of one pair of the operands really occupies just one layr of the circuit.

The layers that are not any more used by the present couple of operands can immediately be used to add another pair of operands. In this way, the circuit can process as many pairs of operands as is the number of its layers.

Try computation using the button $\boxed{\text{Step}}$. Values corresponding particular pairs of operands are distinguished by color.

Using this pipelined approach, the number of parallel steps necessary to add two numbers is equal to the number of layers of the circuit, but the rate in which operand pairs enter the circuit is one pair per step.

Han-Carlson: Single Column

If we know determining values of blocks, it is easy to determine which columns send carry:

For the column that is numbered H, consider the block $[H, 0]$:

- if the value of the block $[H, 0]$ is $+$, the column H sends carry;
- if the value of the block $[H, 0]$ is $-$, the column H does not send carry;
- if the value of the block $[H, 0]$ is $<$, the column H would propagate carry comming to it from the right;
  however there is no column right to $[H, 0]$, no carry to propagate, hence H does not send carry.

In this scene, for one number H (initially the number of the leftmost column) the evaluation tree of the block $[H, 0]$, built using the Han-Carlson splitting strategy, is shown.

Clock to some column number box (the upper row of boxes) to change H. The construction of the tree and evaluation are not shown.

Han-Carlson: Overlapping

This scene is very similar to the previous one, but the evaluation trees of several blocks of the form $[H, 0]$ are shown.

Click to some column number box (the upper row of boxes) to select/deselect another H, or click the button $\boxed{\text{All}}$ or $\boxed{\text{Clear}}$ to select/deselect all columns.

You can see that trees for different blocks overlap and therefore if we need the evaluation trees for all blocks $[H, 0]$, we do not need to build them separately, but we could recycle parts of one tree to construct another one.

Han-Carlson: Full Adder

This scene shows the full Han-Carlson adder.

Look how the specific way of splitting makes the evaluation trees of different blocks $[H, 0]$ nicely overlapping.

The scene allows changing values of operands and computing carries by computing values of the blocks $[H, 0]$

We can see that the number of parallel steps necessary to add two numbers is equal to the number of layers of the circuit, which is $2 + log_2 N$, where N is the number of the column.

Han-Carlson: Layered Adder

In the previous scene, we have seen that the path for certain signals in the circuit was shorter then the height of the cirguit. In fact, steps of the 'computation' were delayed with respect to how a real circuit would work.

In this scene, we added white 'delay' boxes. They have just one input and their only function is to delay signals of shorter path to be synchronized with other signals.

Try to step the computation to see that the computation proceeds in a synchronized way by the layers of the circuit.

Han-Carlson: Forgetting Adder

The scene is very similar to the previous one with computation layer by layer.

Note that if values in one layer are computed, the values in the previous layers are not any more needed. This is why this scene forgets the values in the previous layers.

Try the computation of block values to see that the values of a single layer are needed.

Han-Carlson: Pipelined Adder

As it has been shown in the previous scene, addition of one pair of the operands really occupies just one layr of the circuit.

The layers that are not any more used by the present couple of operands can immediately be used to add another pair of operands. In this way, the circuit can process as many pairs of operands as is the number of its layers.

Try computation using the button $\boxed{\text{Step}}$. Values corresponding particular pairs of operands are distinguished by color.

Using this pipelined approach, the number of parallel steps necessary to add two numbers is equal to the number of layers of the circuit, but the rate in which operand pairs enter the circuit is one pair per step.