ARM processor implementation

P. Bakowski

bako@ieee.org

ARM clocking scheme

ARM architecture design is based around 2-phase non-overlapping clocks.

ARM clocking scheme

ARM architecture design is based around 2-phase non-overlapping clocks.

This clocking scheme allows the use of level sensitive transparent latches.

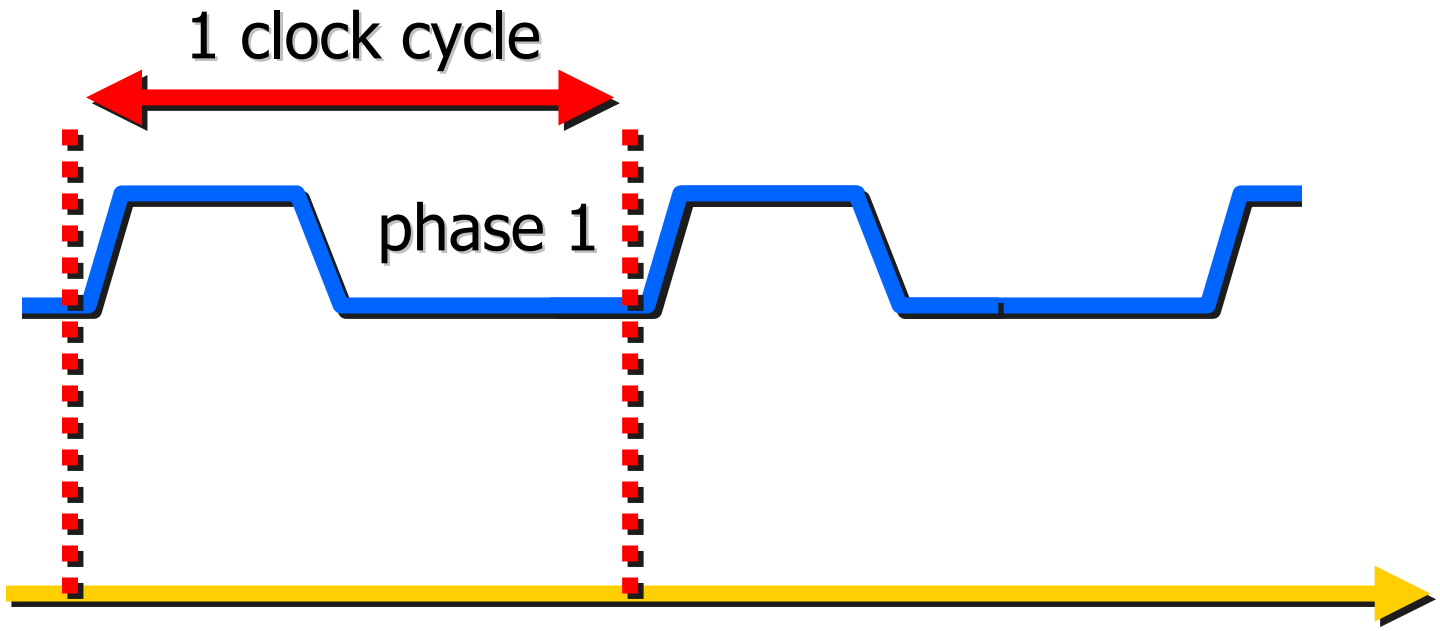
ARM clocking scheme

ARM architecture design is based around 2-phase non-overlapping clocks.
This clocking scheme allows the use of level sensitive transparent latches.
Data move alternatively through latches that are open during phase 1 and latches that are open during phase 2.
This overlapping property ensures that there are no race conditions in the circuit.

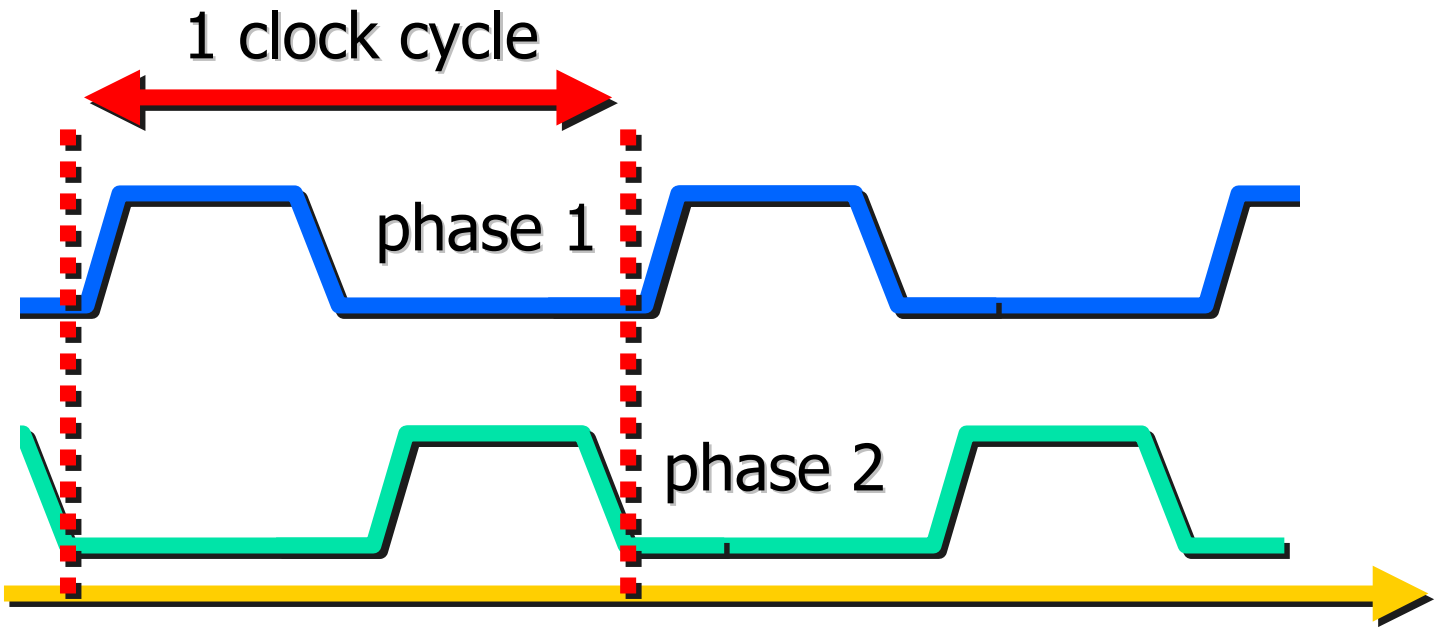
ARM clocking scheme

ARM architecture design is based around 2-phase non-overlapping clocks.
This clocking scheme allows the use of level sensitive transparent latches.
Data move alternatively through latches that are open during phase 1 and latches that are open during phase 2.
This overlapping property ensures that there are no race conditions in the circuit.

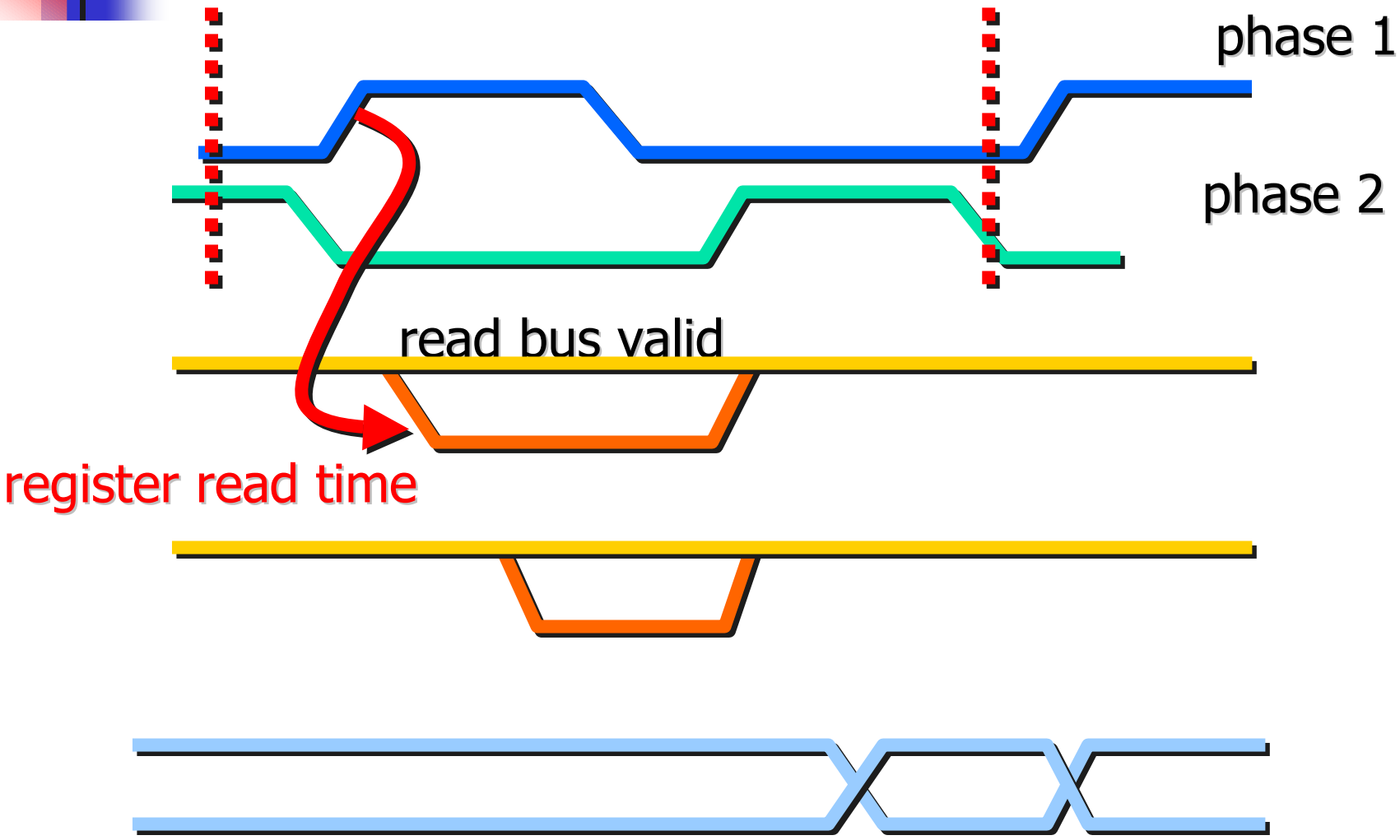
ARM clocking scheme



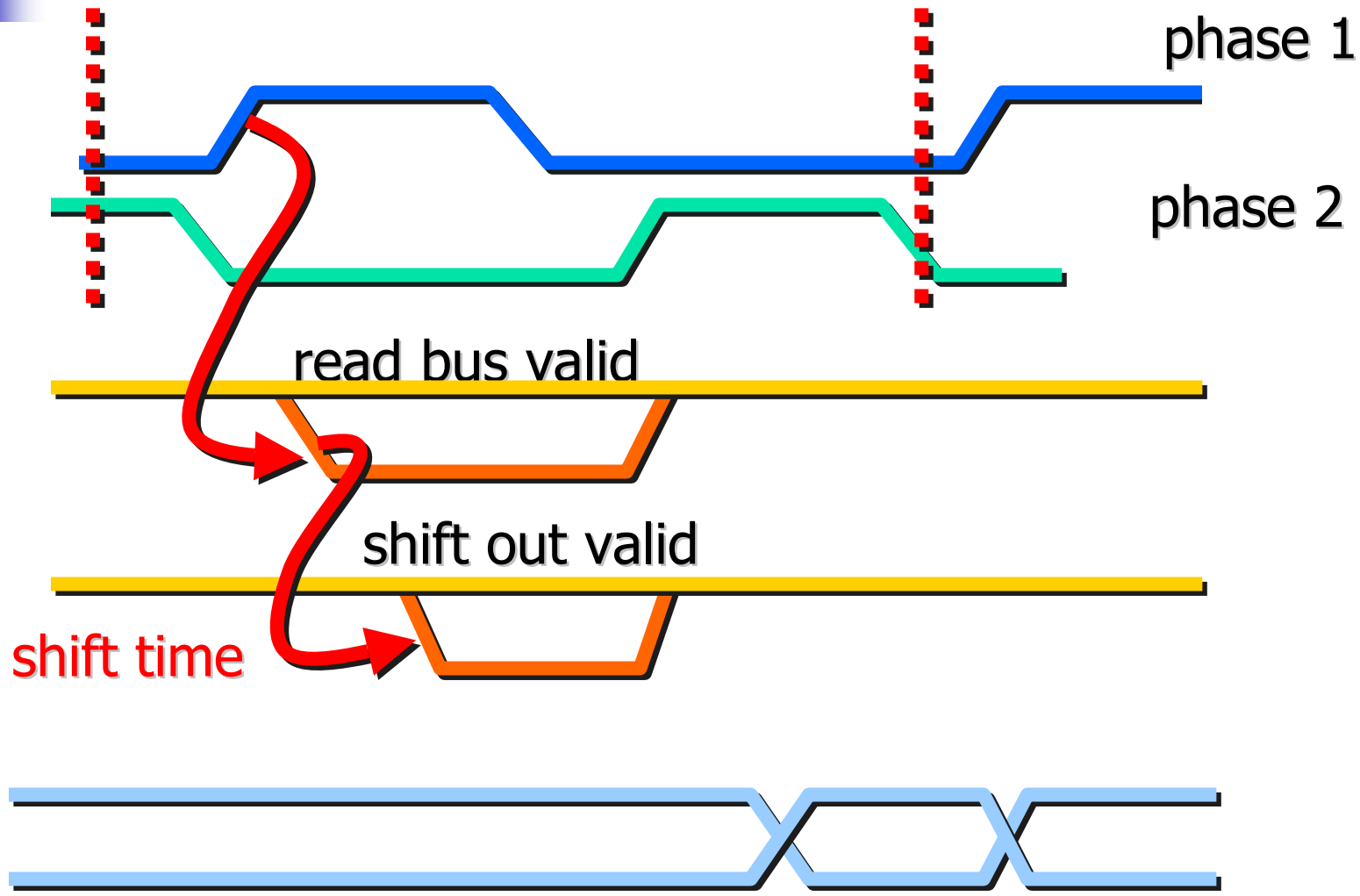
ARM clocking scheme



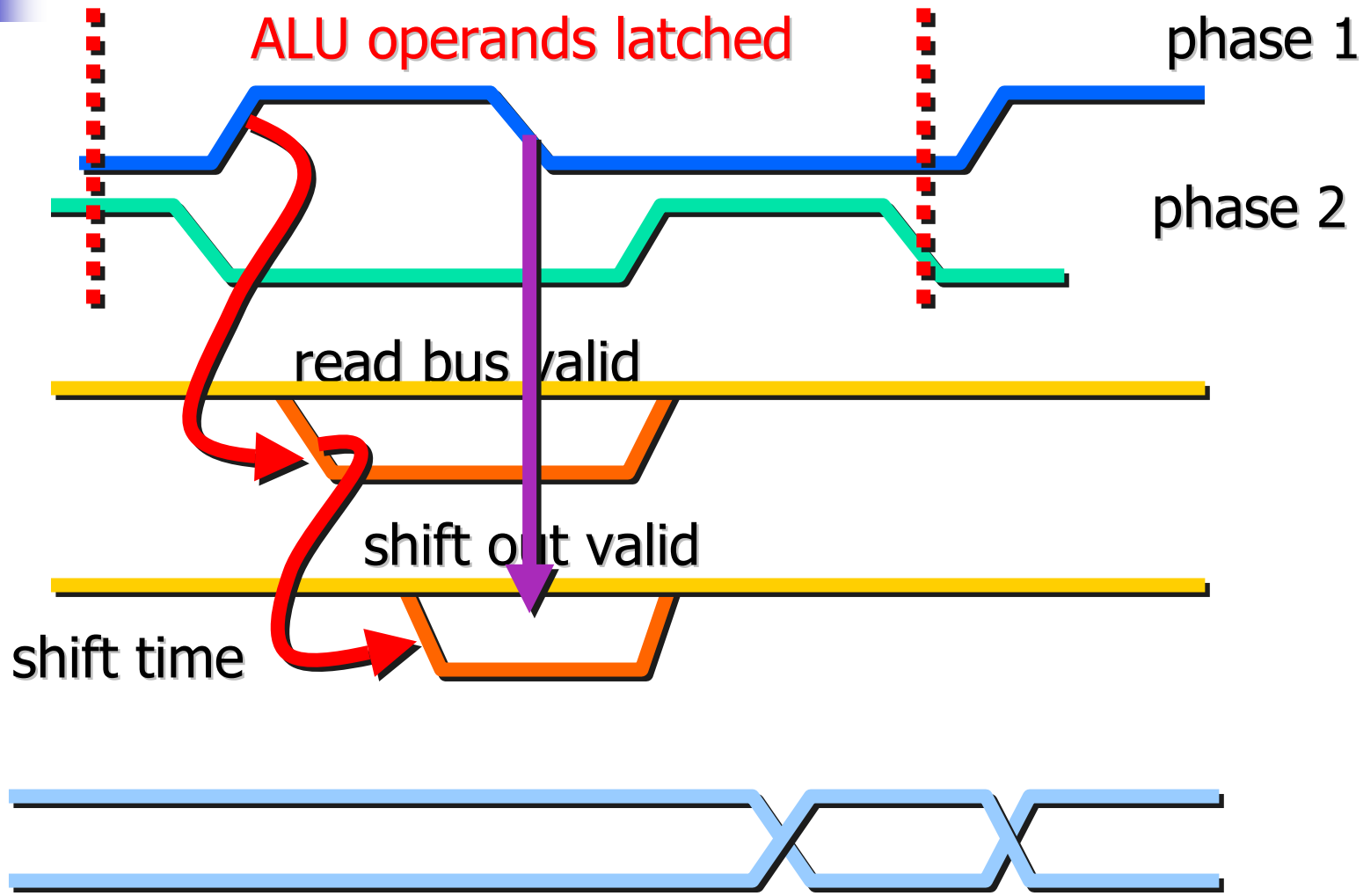
ARM data path timing



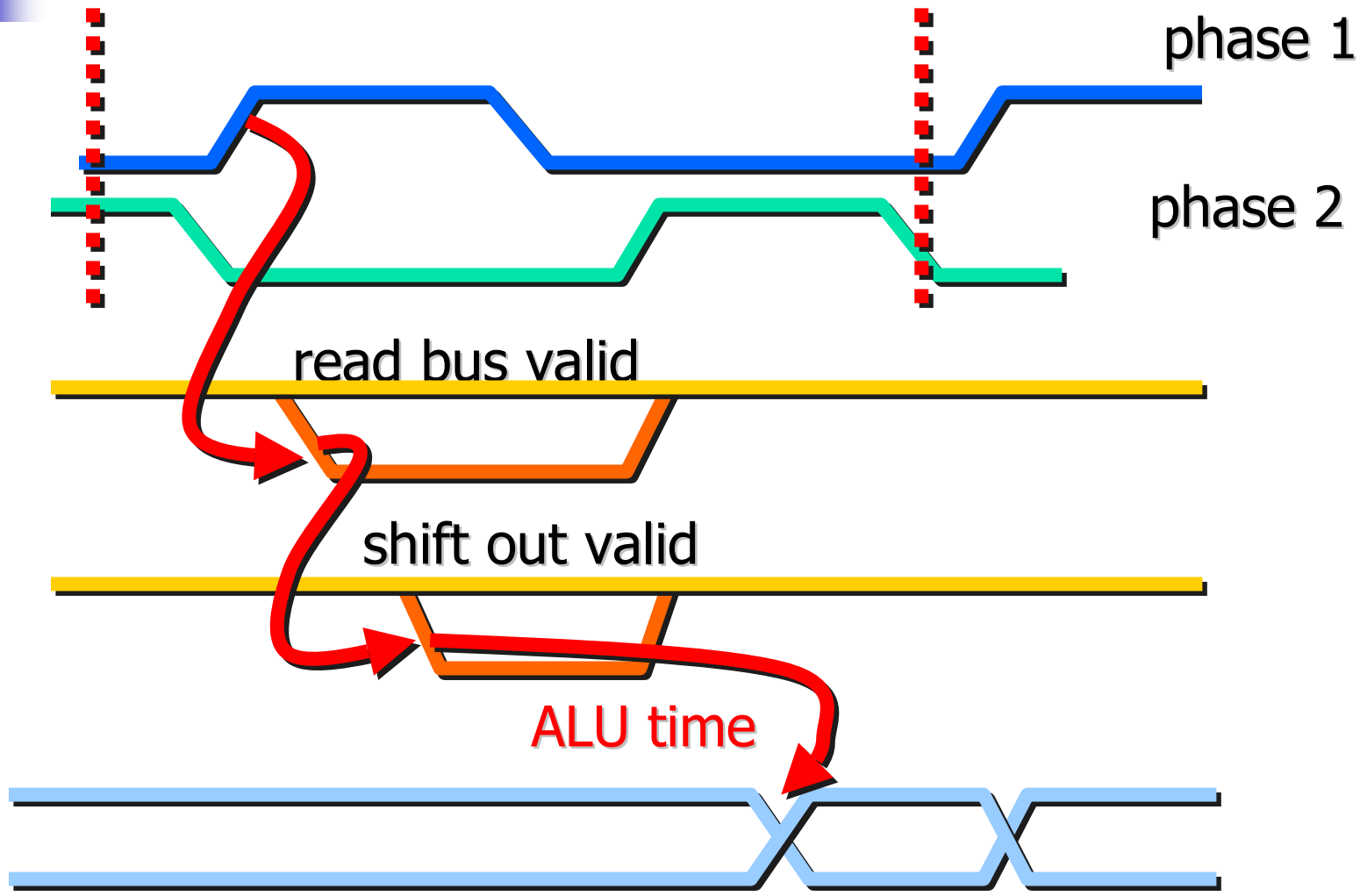
ARM data path timing



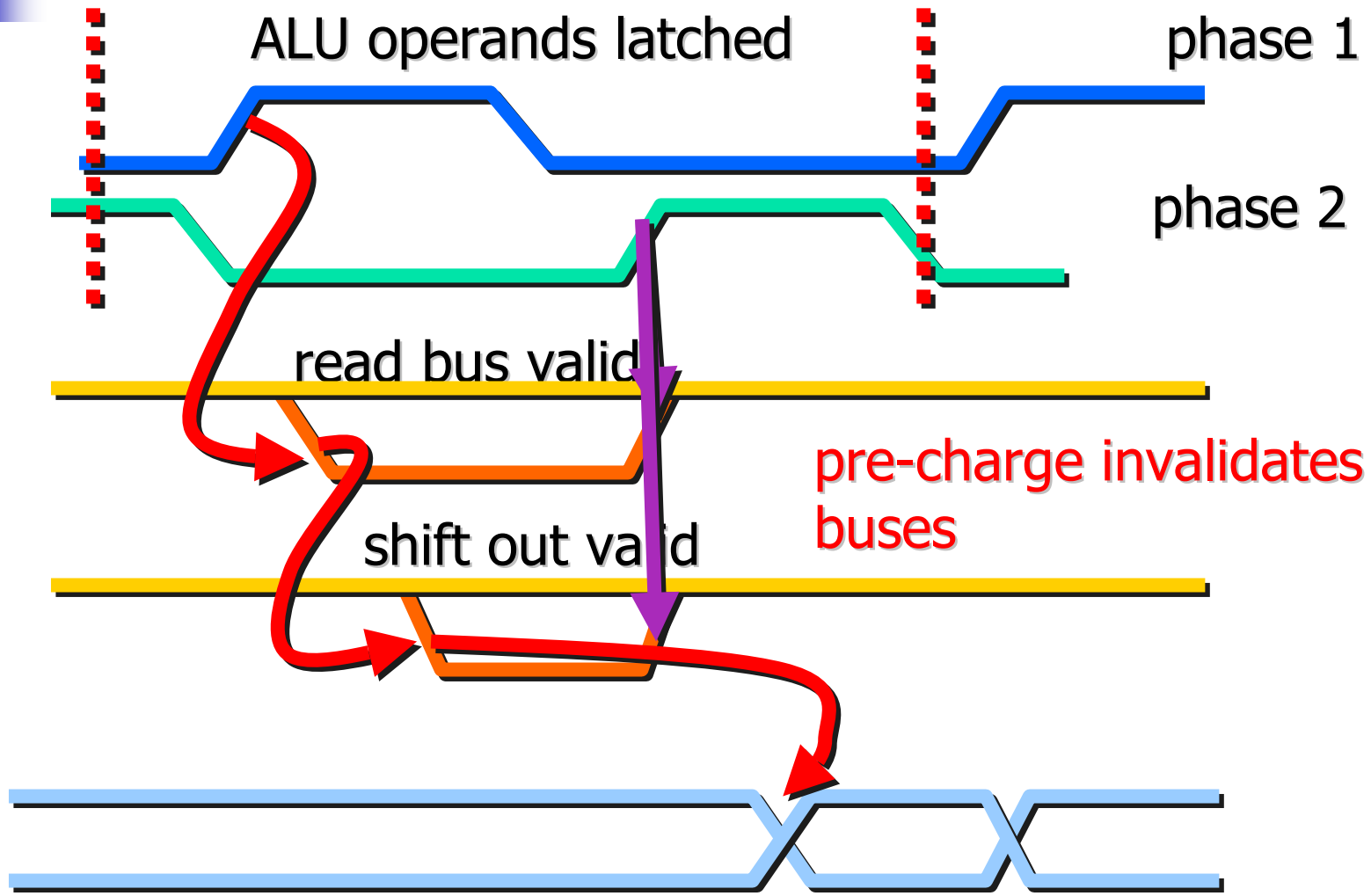
ARM data path timing



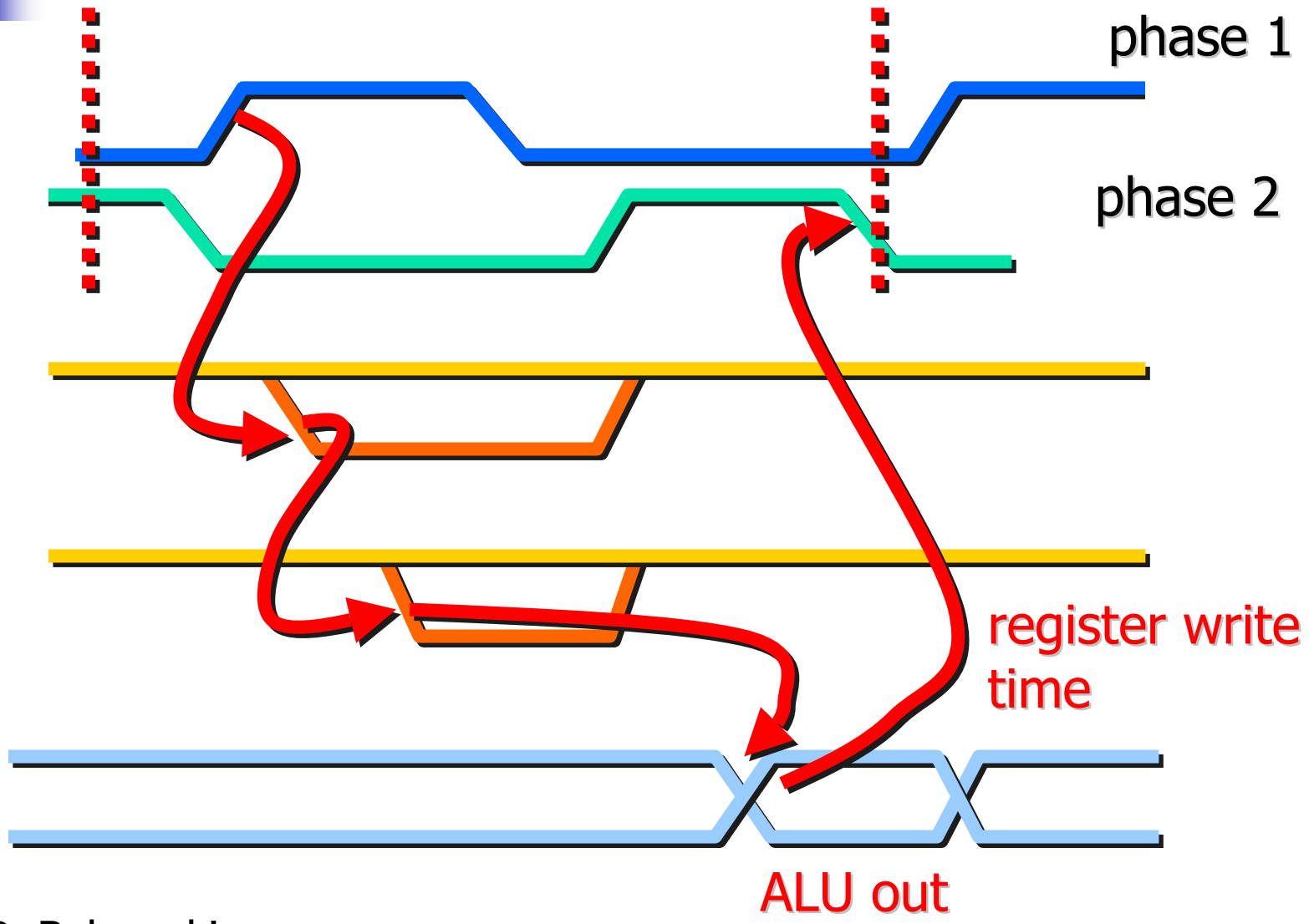
ARM data path timing



ARM data path timing



ARM data path timing



ARM data path timing

The minimum data-path cycle is the sum of:

- the register read time
- the shifter delay
- the ALU delay
- the register write set-up time
- the phase 2 to phase 1 non-overlap time

ARM data path timing

The minimum data-path cycle is the sum of:

- the register read time
- the shifter delay
- the ALU delay
- the register write set-up time
- the phase 2 to phase 1 non-overlap time

ARM data path timing

The minimum data-path cycle is the sum of:

- the register read time
- the shifter delay
- the ALU delay
- the register write set-up time
- the phase 2 to phase 1 non-overlap time

ARM data path timing

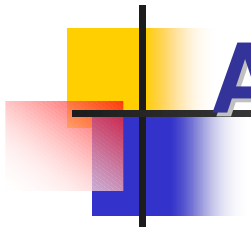
The minimum data-path cycle is the sum of:

- the register read time
- the shifter delay
- the ALU delay
- the register write set-up time
- the phase 2 to phase 1 non-overlap time

ARM data path timing

The minimum data-path cycle is the sum of:

- the register read time
- the shifter delay
- the ALU delay
- the register write set-up time
- the phase 2 to phase 1 non-overlap time



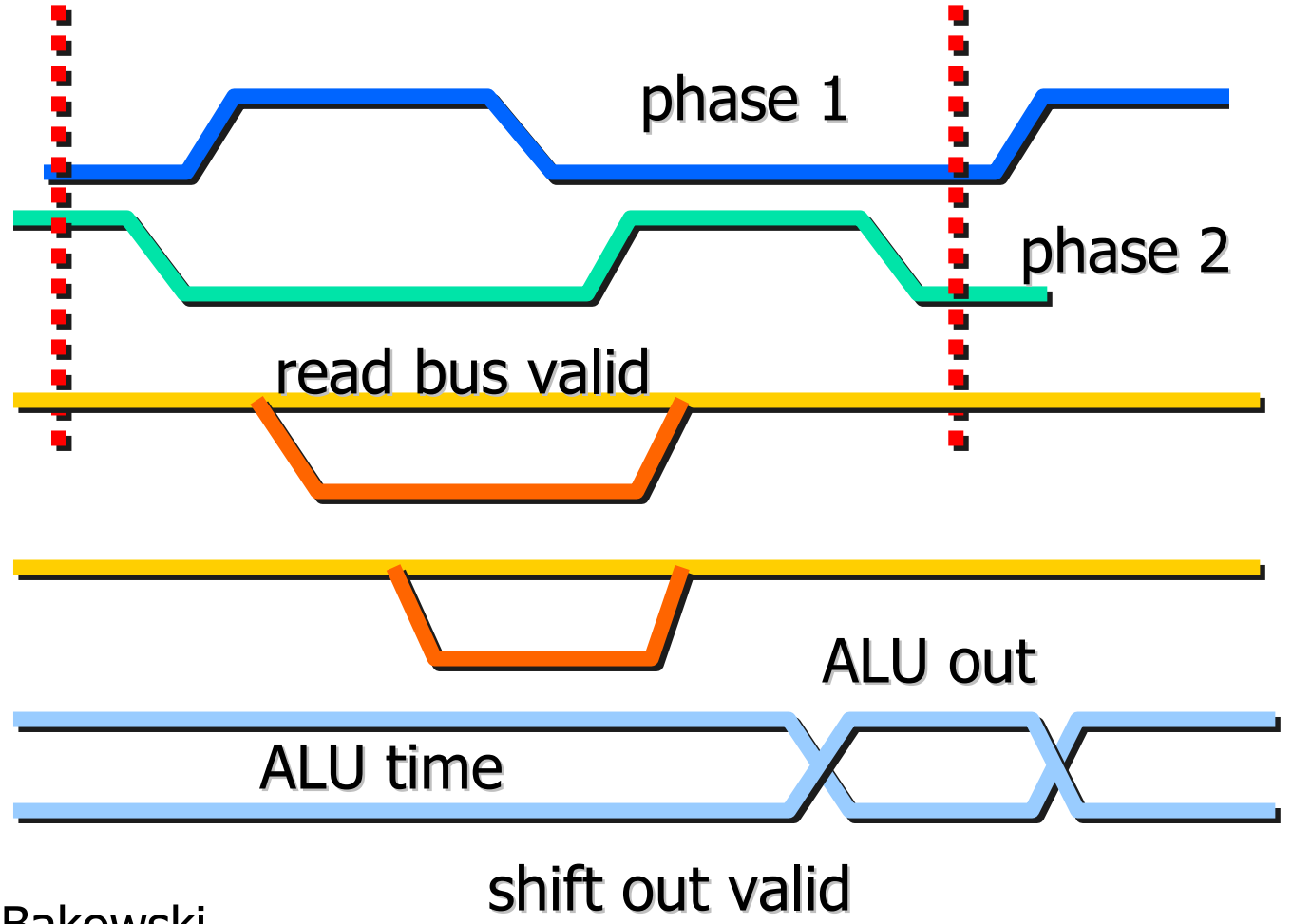
ARM data path timing

The minimum data-path cycle is the sum of:

- the register read time
- the shifter delay
- the ALU delay
- the register write set-up time
- the phase 2 to phase 1 non-overlap time

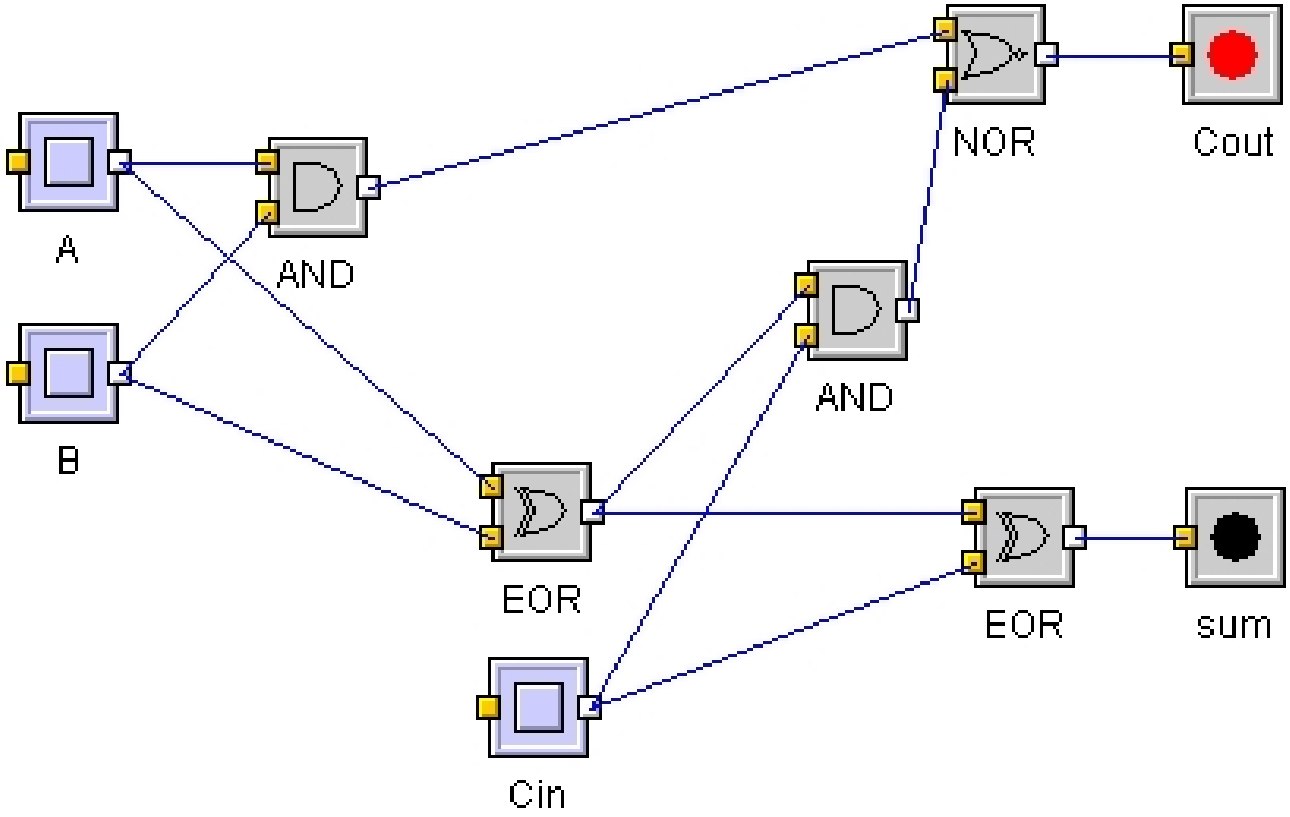
ARM data path timing

Timing in a 3-stage pipeline.



ARM 1 - adder design

ARM1 – the original ripple-carry adder:

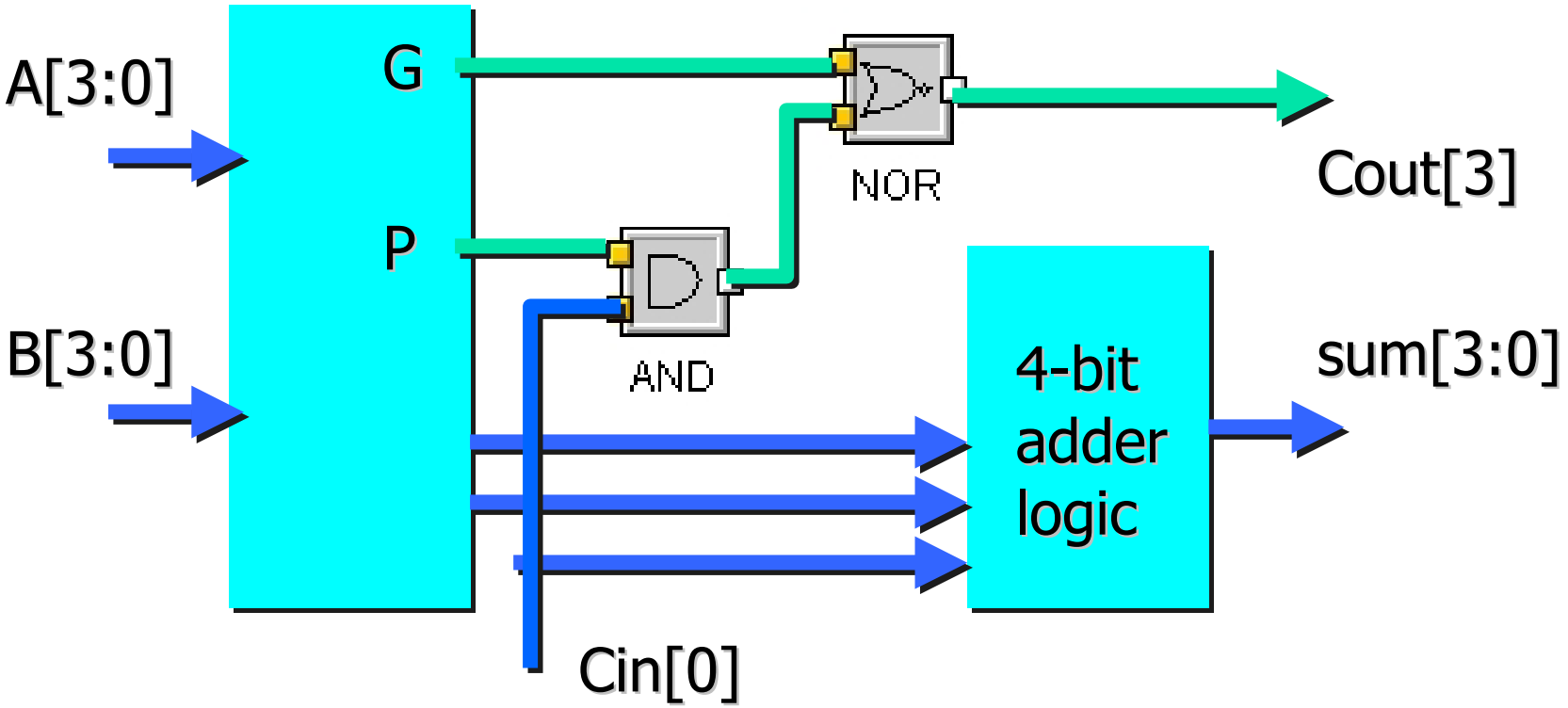


ARM 1 - adder design

Using CMOS and-or-invert gate for the carry logic and alternating and-or logic so that event bits use the circuit shown and odd bits use the dual circuit with inverted inputs and outputs and and-or gates swapped around, the worst-case carry path is 32 gates long.

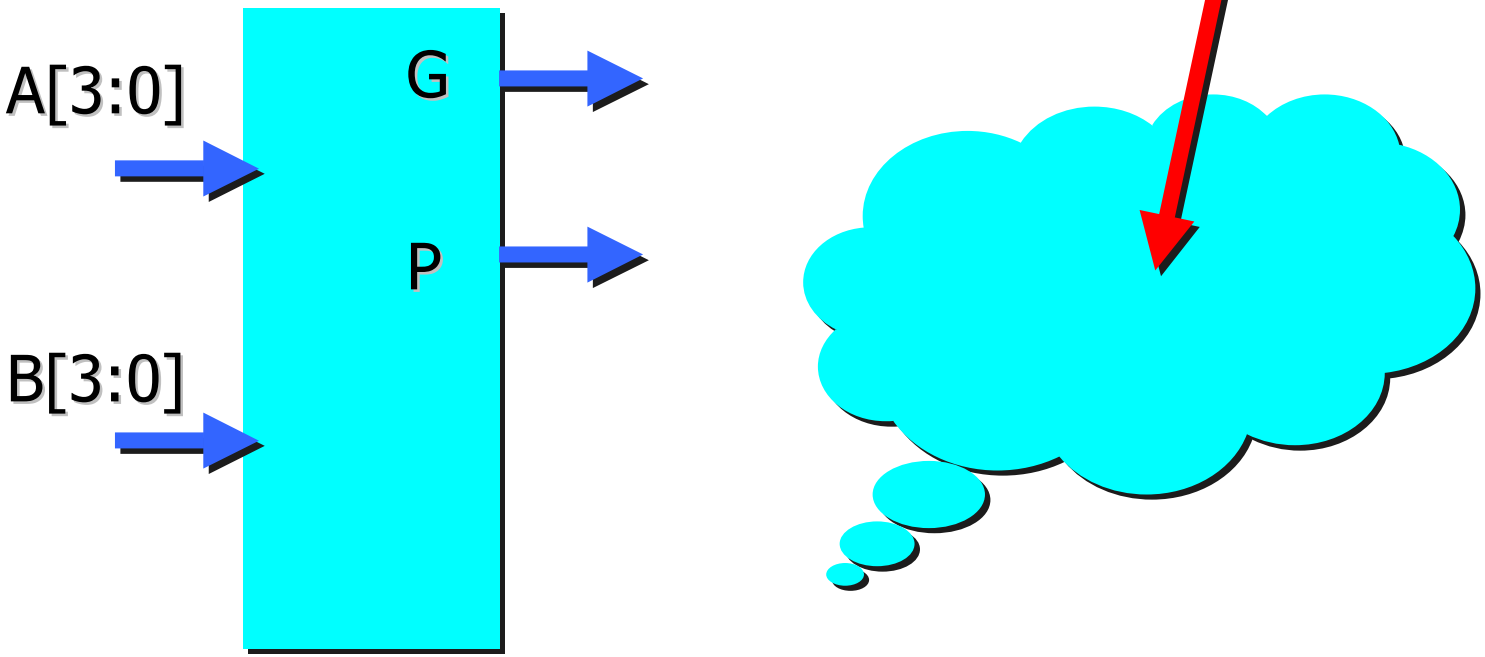
ARM 2 - adder design

To reduce the propagation path length ARM2 used a 4-bit carry look-ahead logic.

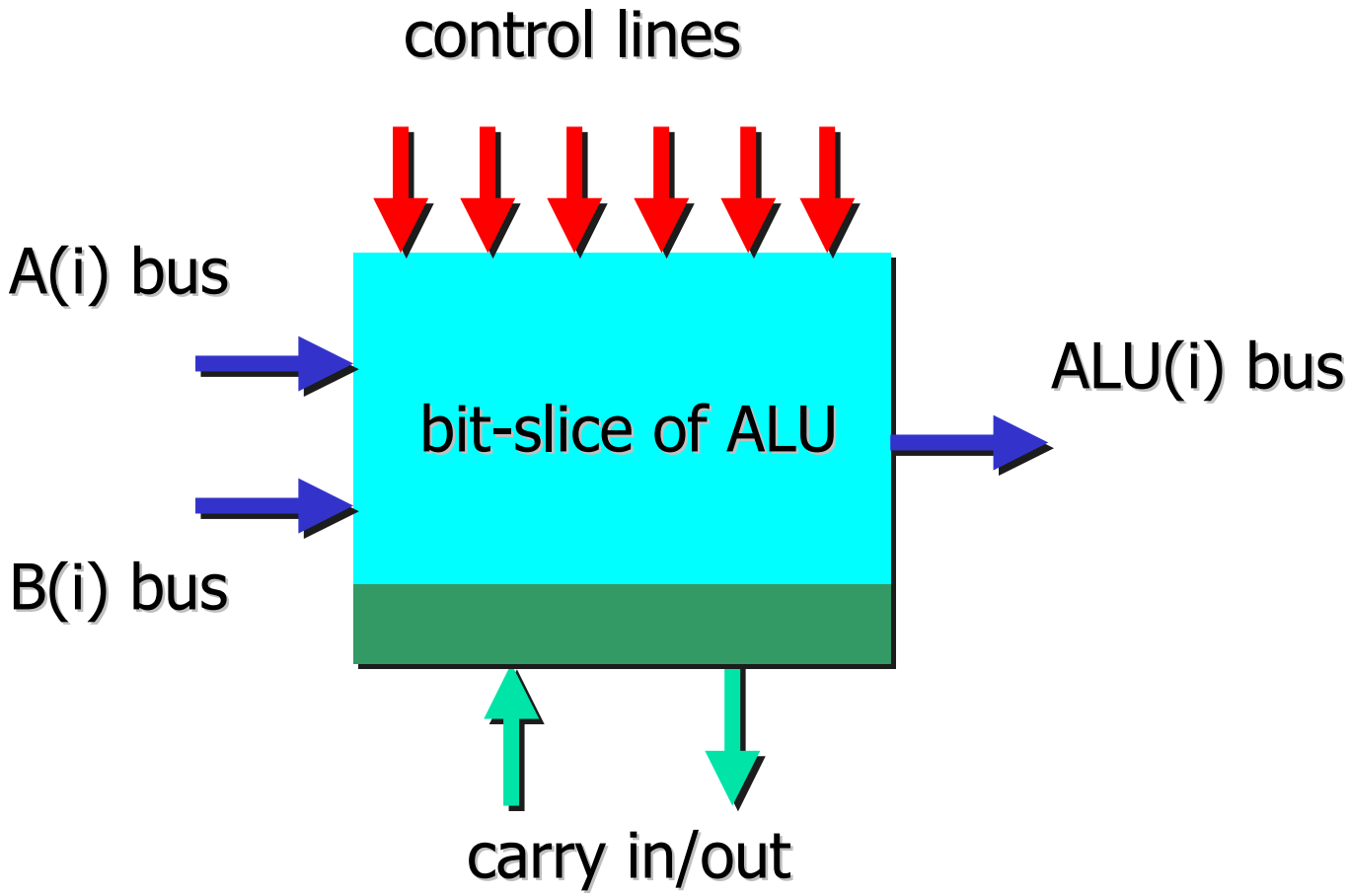


ARM 2 - adder design

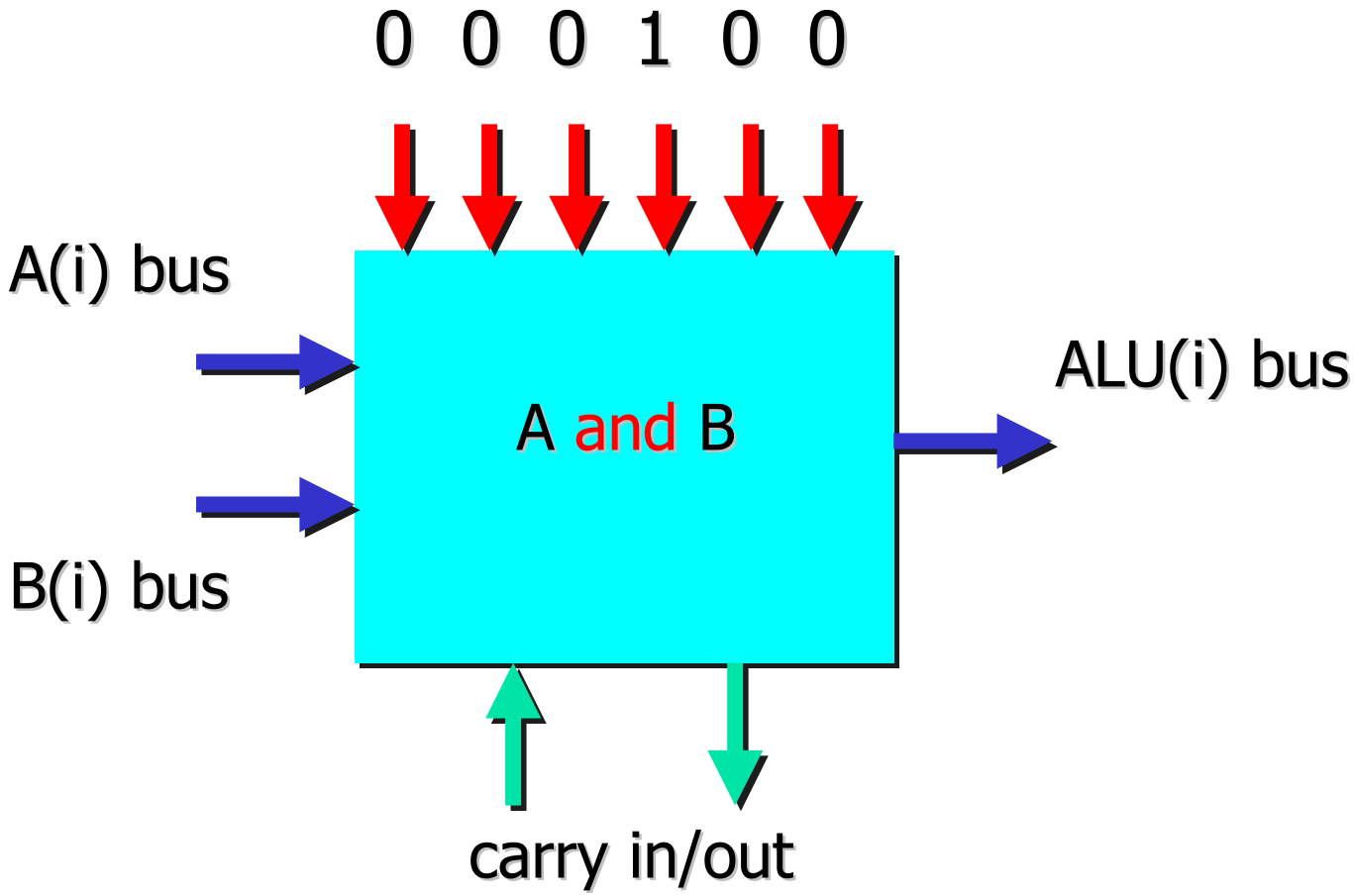
Exercise: Draw the logic scheme of the circuit for Generate and Propagate signals



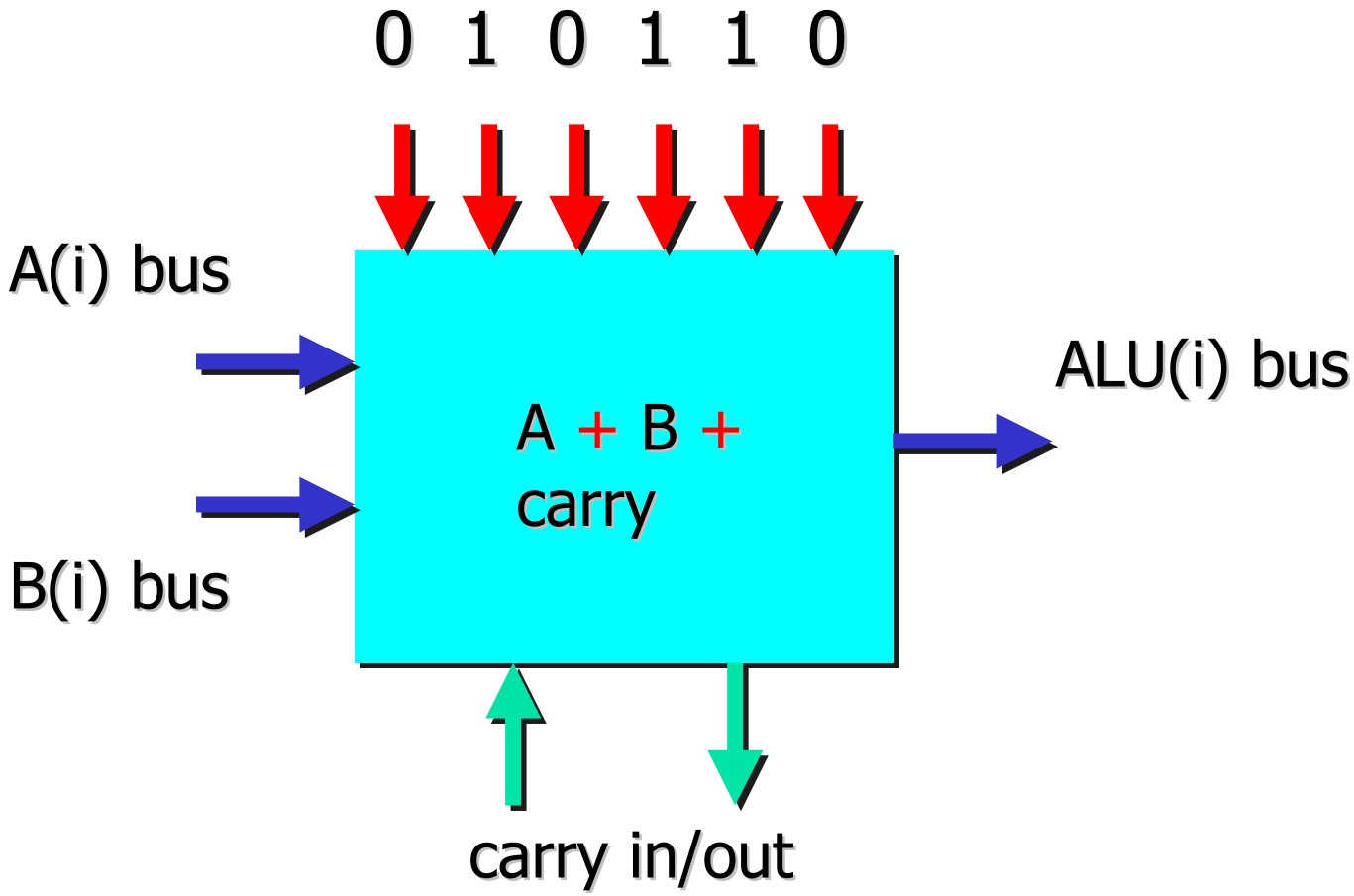
ARM 2 - ALU design



ARM 2 - ALU design



ARM 2 - ALU design



ARM 6 - adder design

The further reduction of the propagation time is provided by a carry-select adder implemented with ARM 6.

This form of adder computes the sums of various fields of the word for a carry-in both zero and one.

The final result is selected by using the correct carry-in value to control a multiplexer.

ARM 6 - adder design

The further reduction of the propagation time is provided by a carry-select adder implemented with ARM 6.

This form of adder computes the sums of various fields of the word for a carry-in both zero and one.

The final result is selected by using the correct carry-in value to control a multiplexer.

ARM 6 - adder design

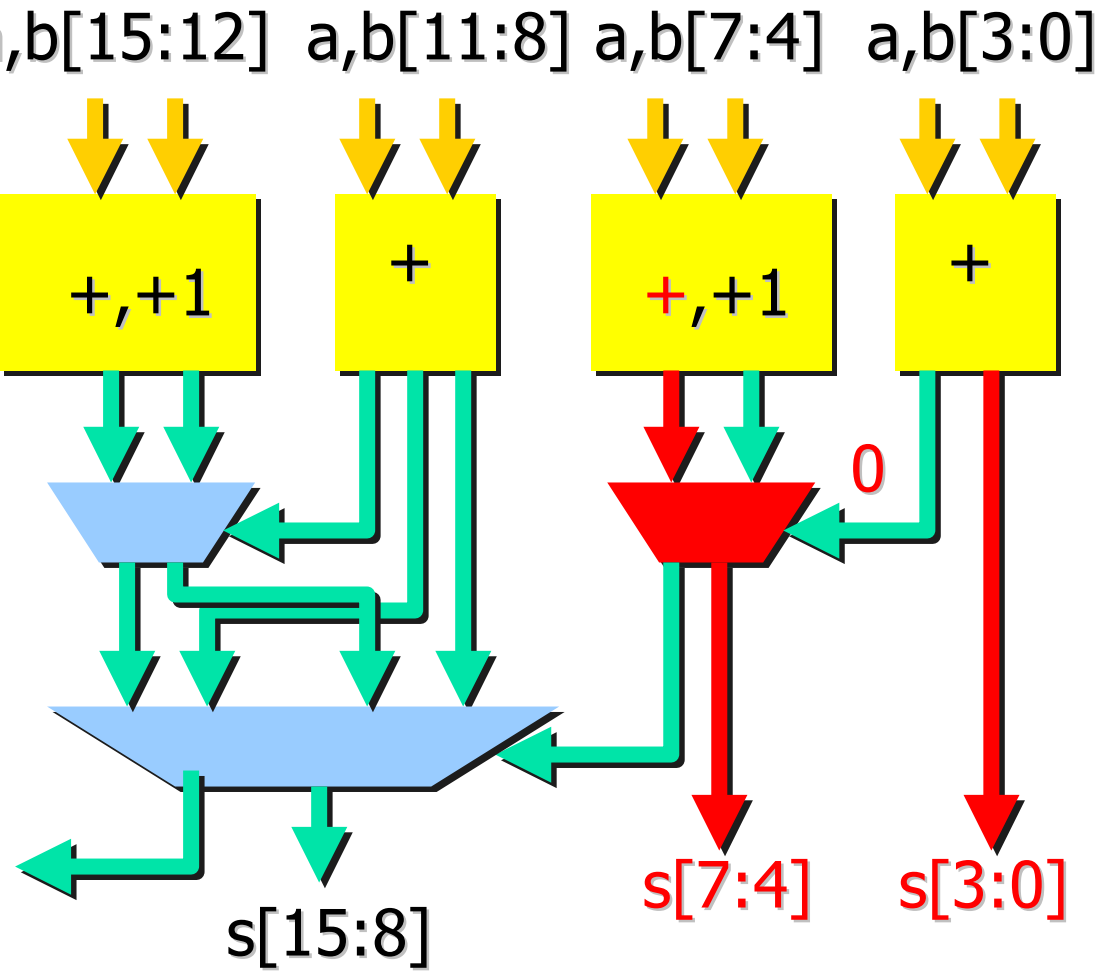
The further reduction of the propagation time is provided by a carry-select adder implemented with ARM 6.

This form of adder computes the sums of various fields of the word for a carry-in both zero and one.

The final result is selected by using the correct carry-in value to control a multiplexer.

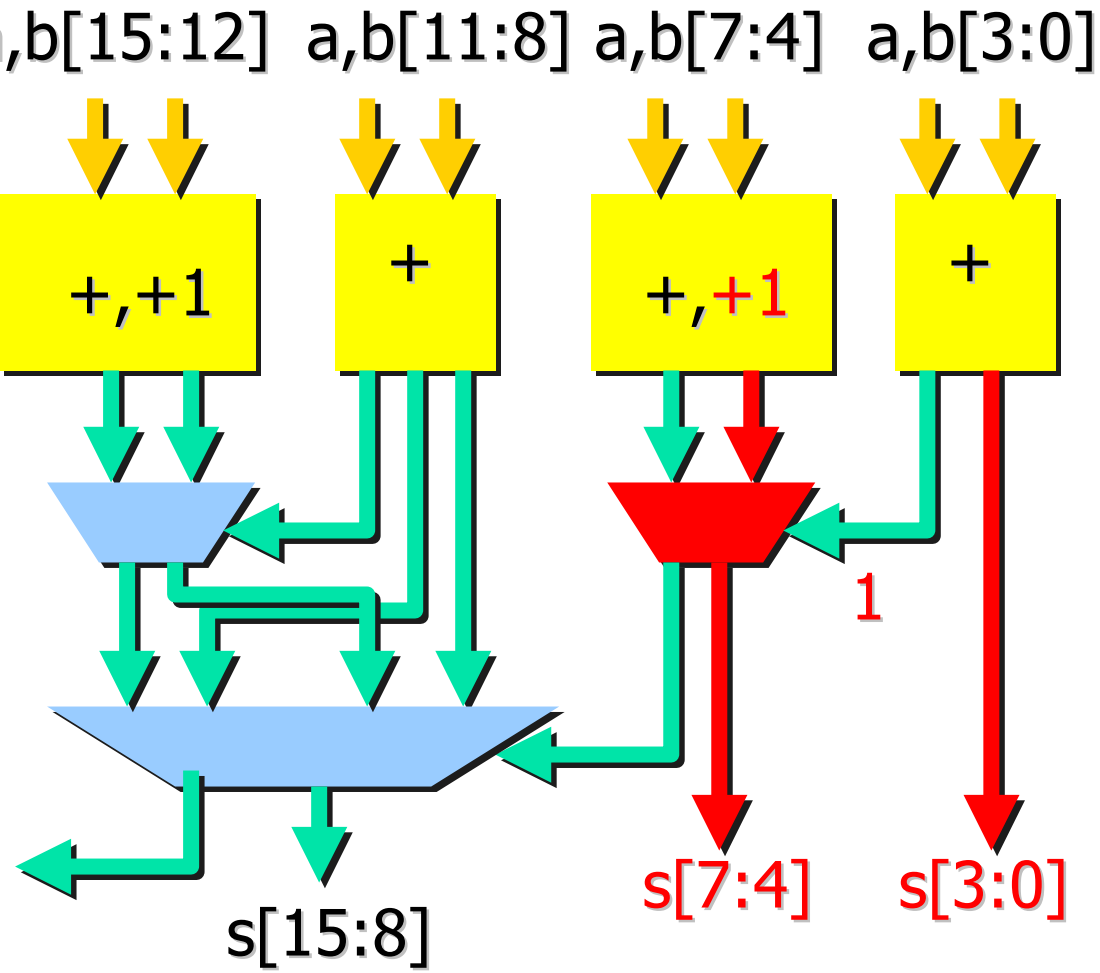
ARM 6 - adder design

the logic for s[31:16]

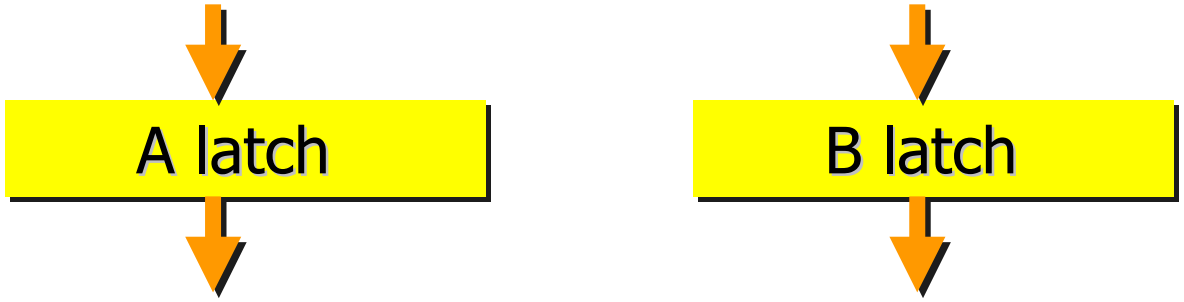


ARM 6 - adder design

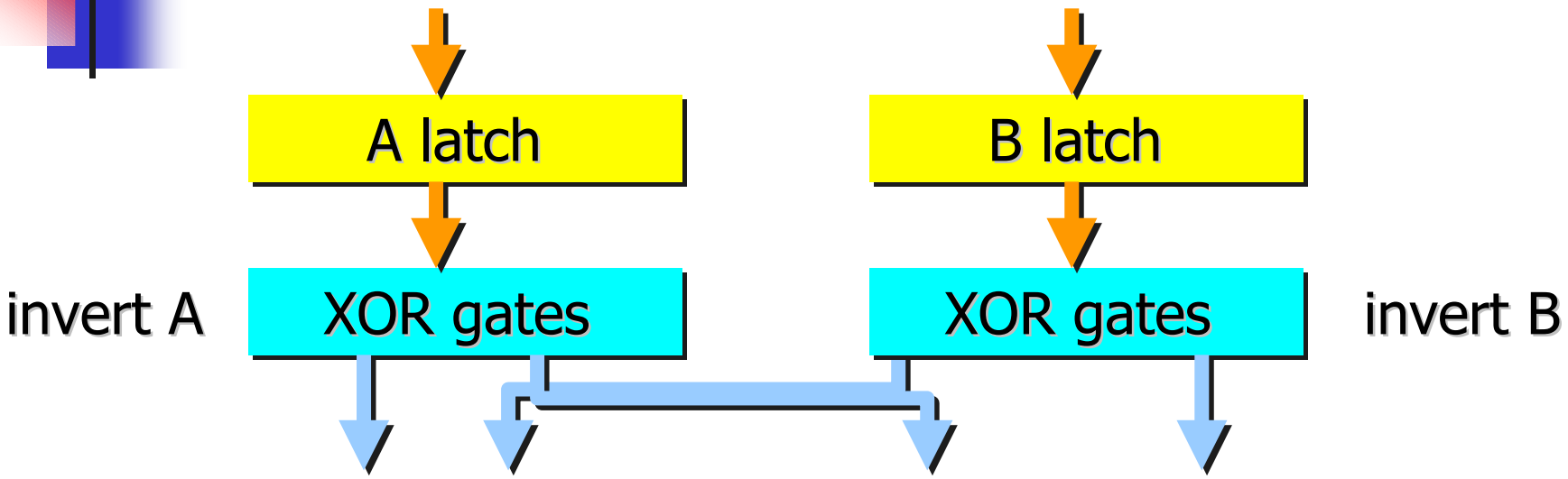
the logic for s[31:16]



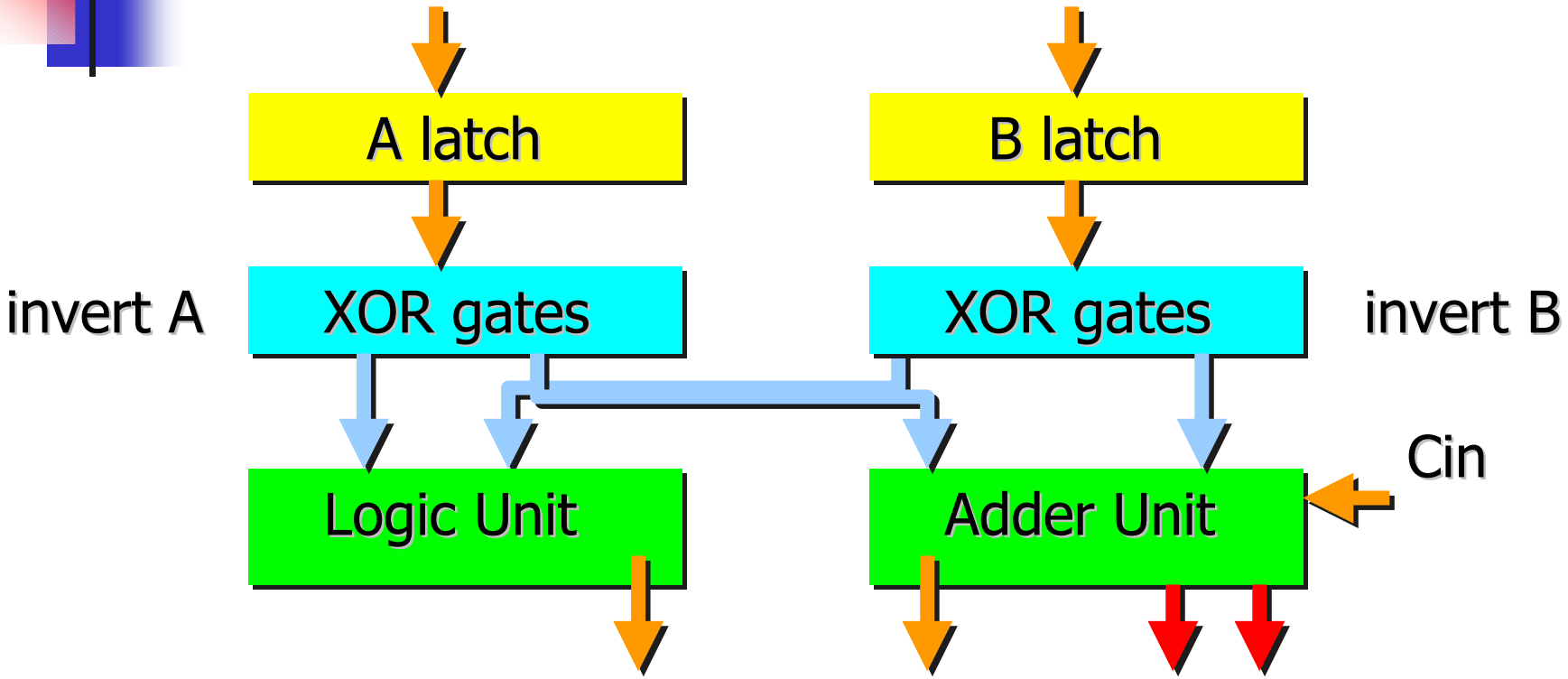
ARM 6 - ALU design



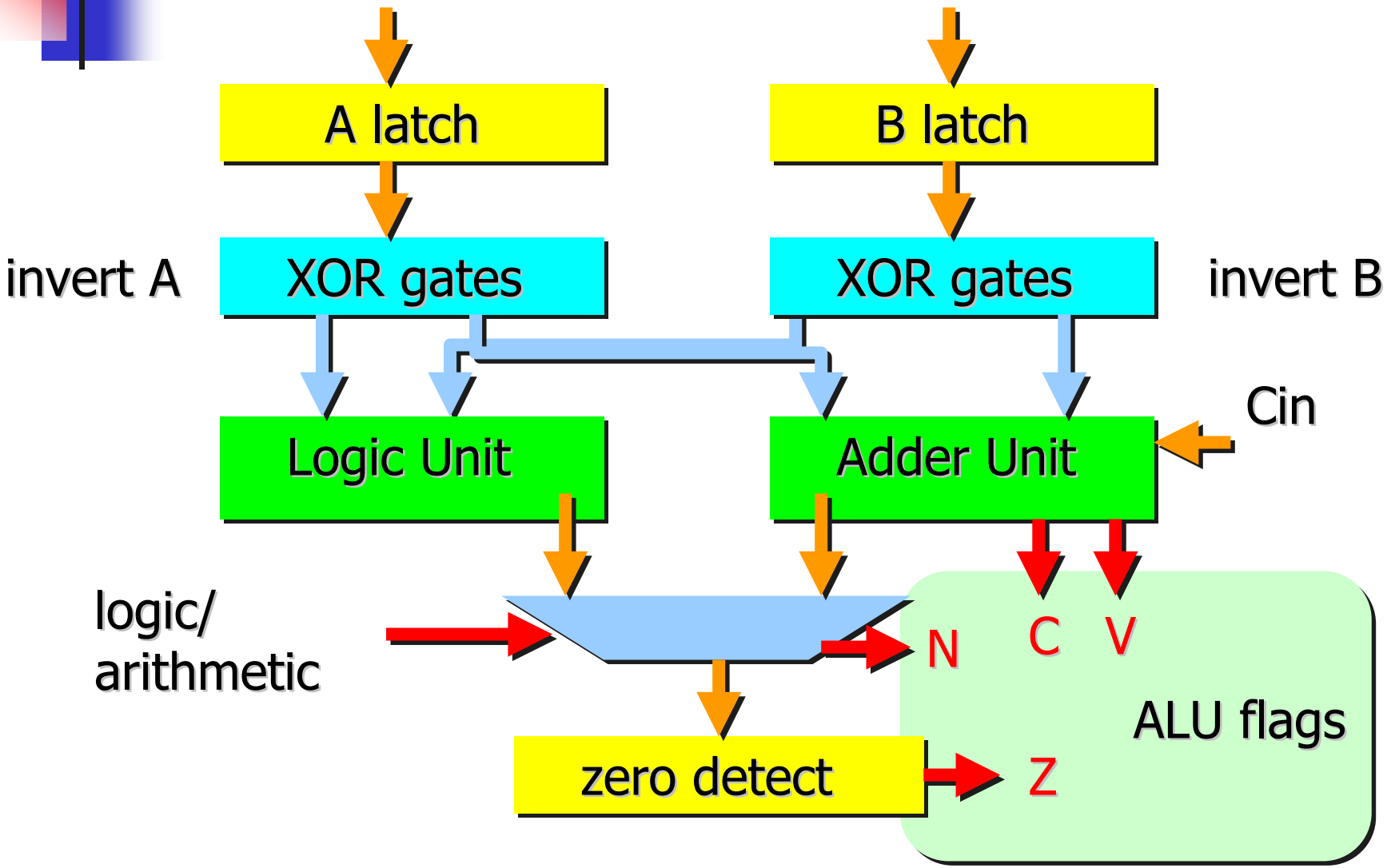
ARM 6 - ALU design



ARM 6 - ALU design



ARM 6 - ALU design



ARM barrel shifter

The shifter delay is critical because it is connected in series with ALU unit.

Therefore, it is implemented as a 32*32 cross-bar switch matrix where each input is steered directly to the appropriate output.

If a pre-charged logic is used, each switch may be implemented as one transistor.

ARM barrel shifter

The shifter delay is critical because it is connected in series with ALU unit.

Therefore, it is implemented as a 32*32 cross-bar switch matrix where each input is steered directly to the appropriate output.

If a pre-charged logic is used, each switch may be implemented as one transistor.

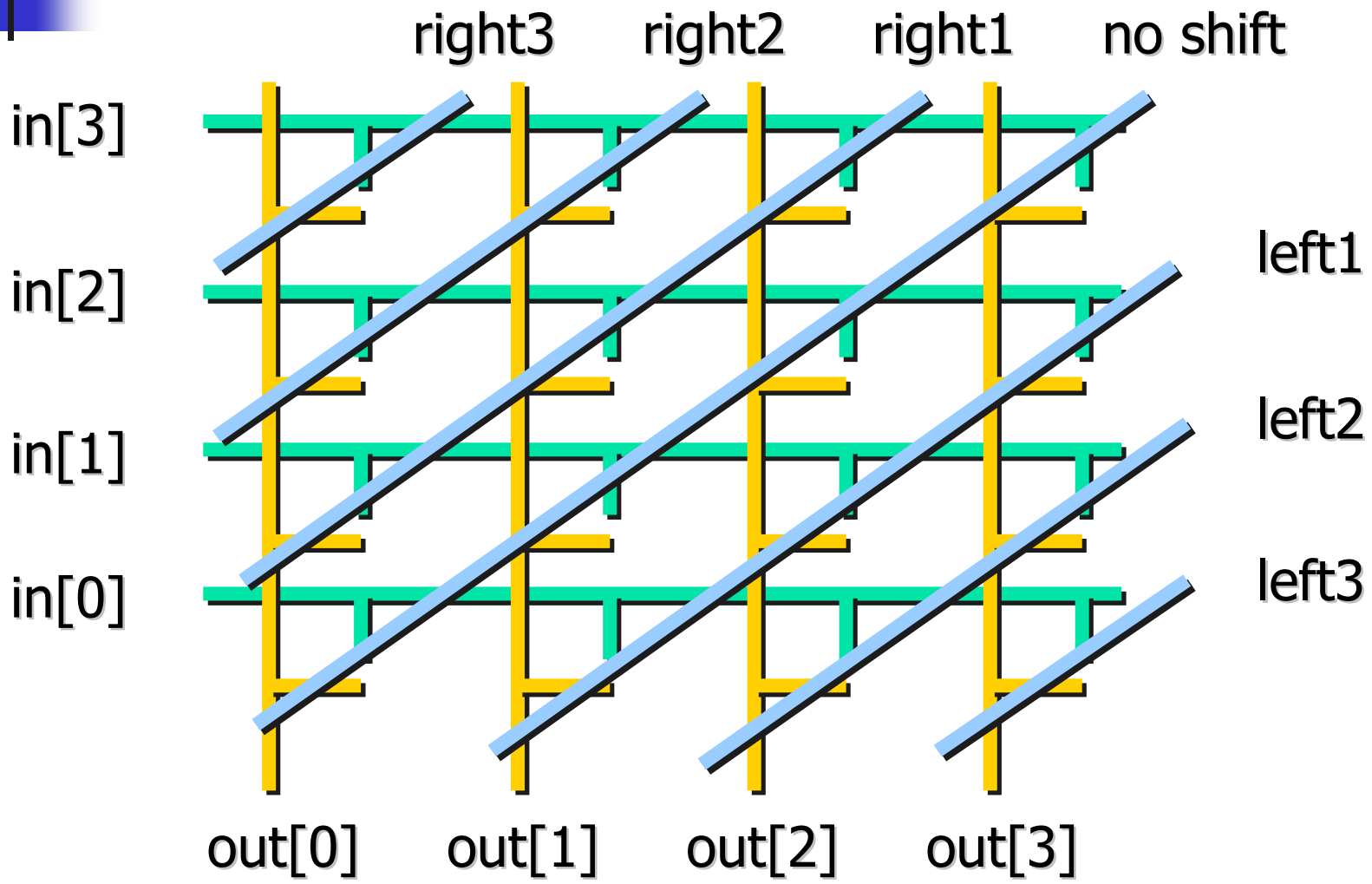
ARM barrel shifter

The shifter delay is critical because it is connected in series with ALU unit.

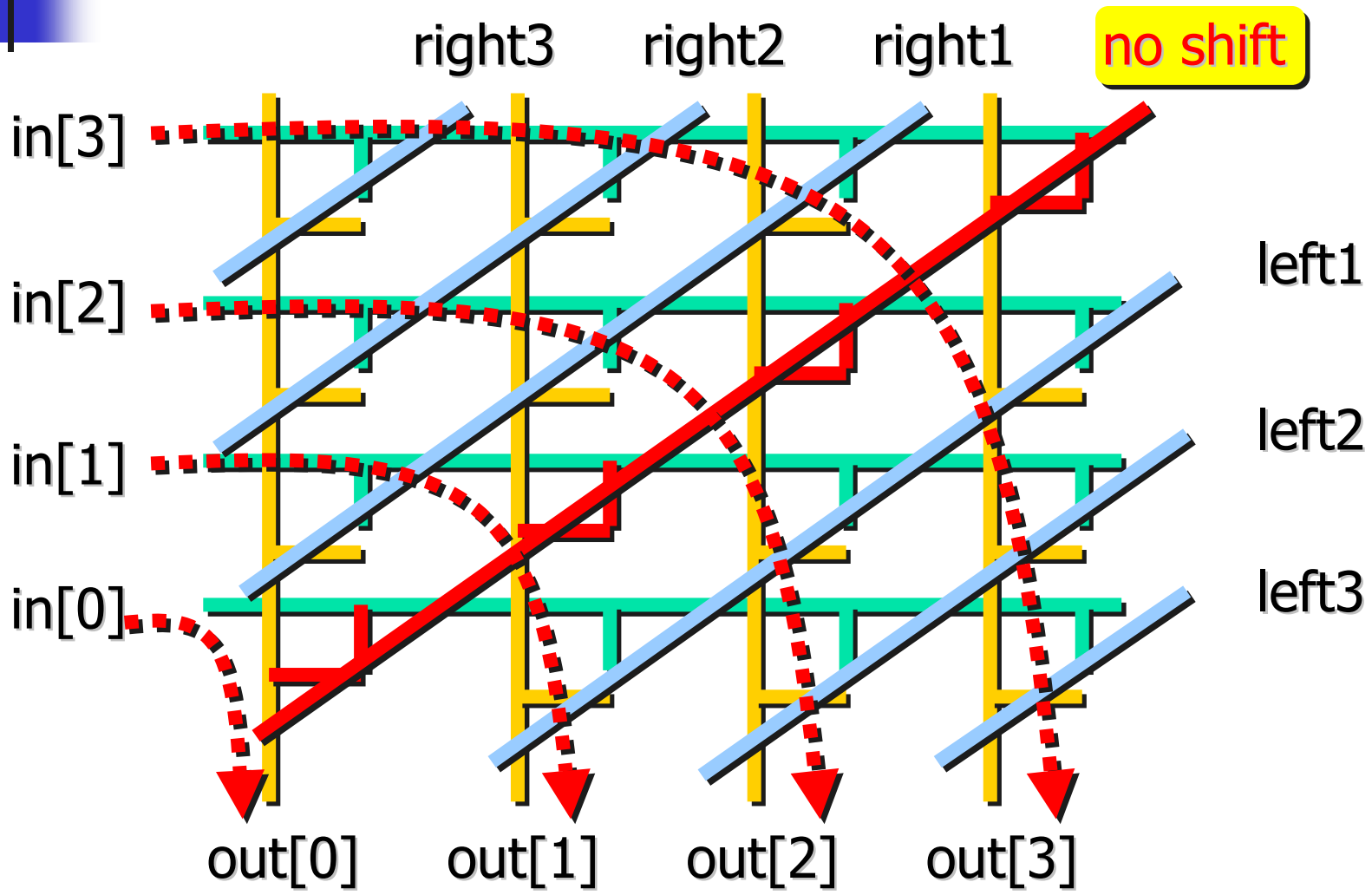
Therefore, it is implemented as a 32*32 cross-bar switch matrix where each input is steered directly to the appropriate output.

If a pre-charged logic is used, each switch may be implemented as a single transistor.

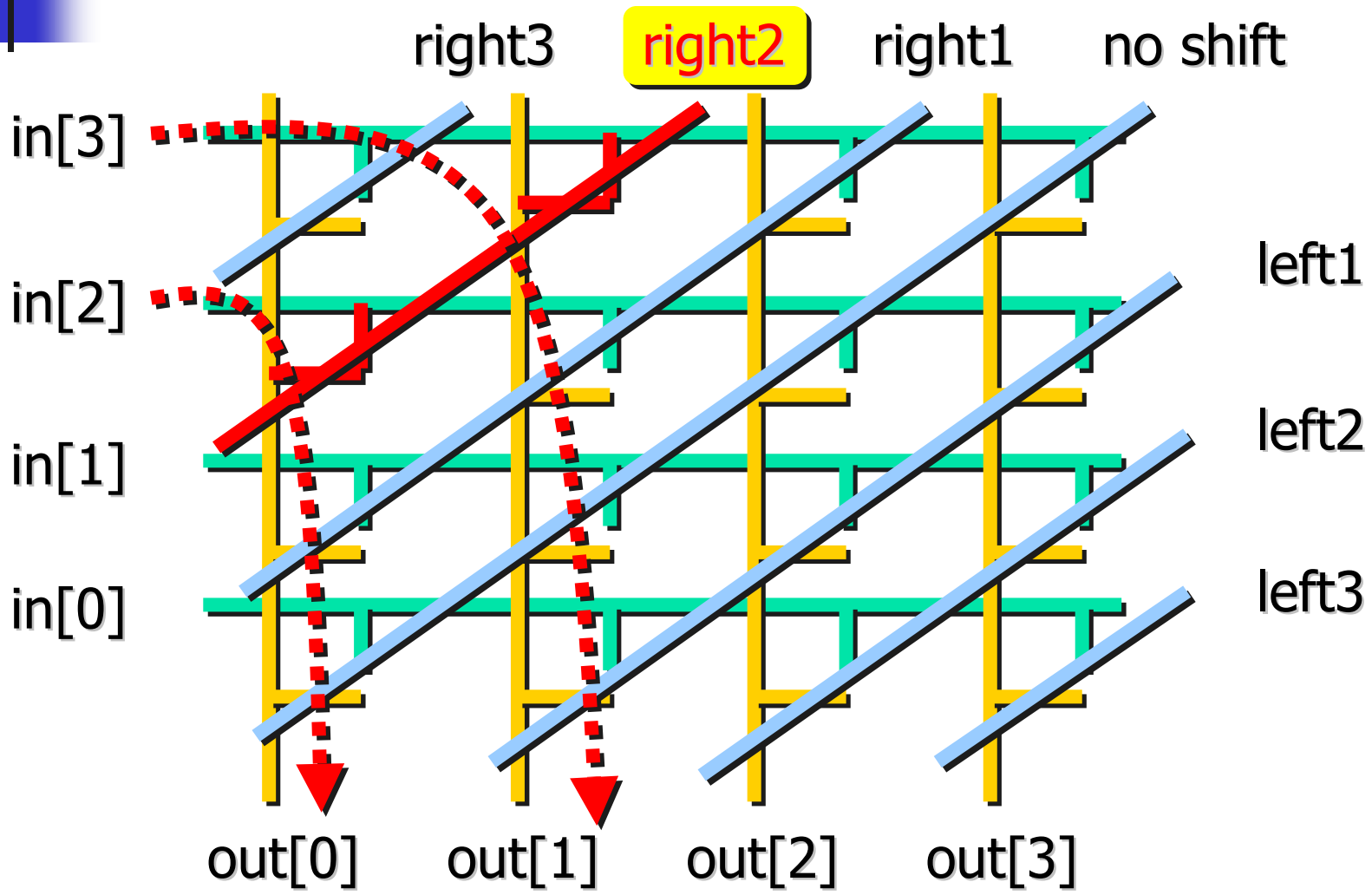
ARM barrel shifter



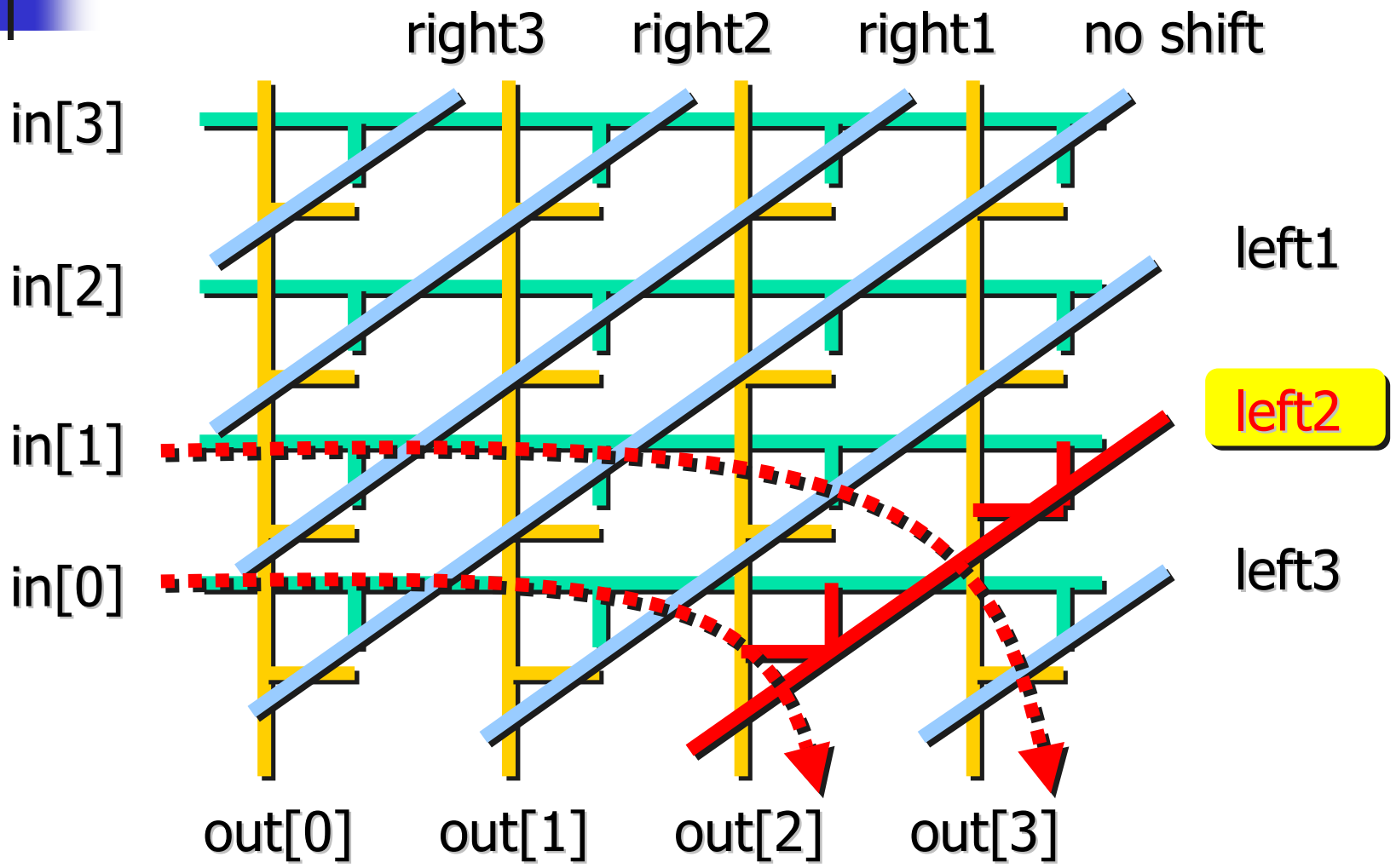
ARM barrel shifter



ARM barrel shifter



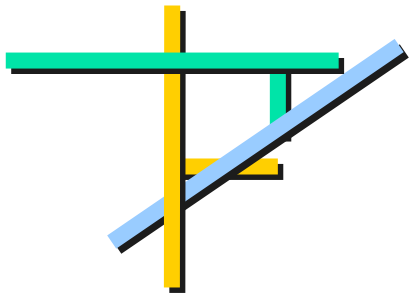
ARM barrel shifter



ARM barrel shifter

In the ARM the barrel shifter operates in negative logic where a '1' is represented as a potential near ground and a '0' by a potential near supply.

Pre-charging sets all the outputs to a logic '0' (supply potential), so those outputs that are not connected to any input during a particular switching operation remain at '0' giving the zero filling required by the shift semantics.

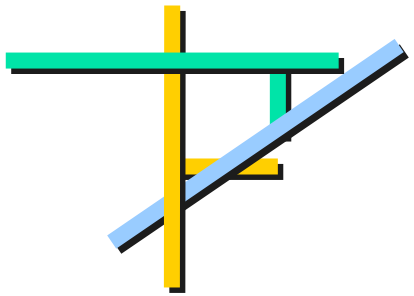


pre-charging to '0'

ARM barrel shifter

In the ARM the barrel shifter operates in negative logic where a '1' is represented as a potential near ground and a '0' by a potential near supply.

Pre-charging sets all the outputs to a logic '0' (supply potential), so those outputs that are not connected to any input during a particular switching operation remain at '0' giving the zero filling required by the shift semantics.

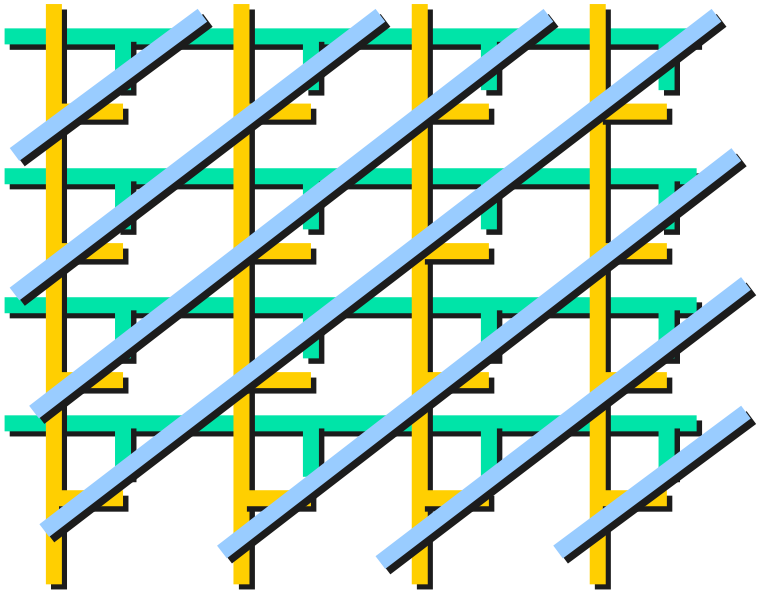


pre-charging to '0'

ARM barrel shifter - rotations

For rotate right function, the right shift diagonal is enabled together with the complementary shift diagonal.

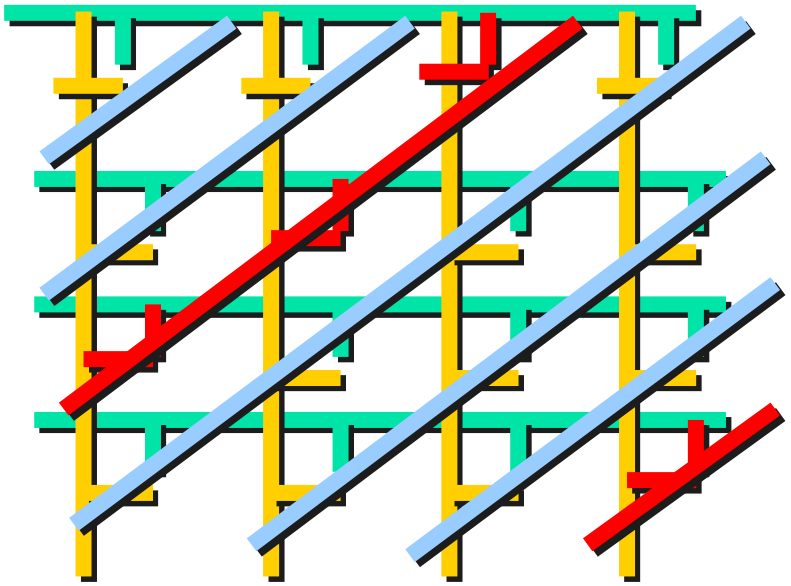
For example, on the 4-bit matrix rotate right one bit is implemented using the ‘right1’ and the ‘left3’ (3=4-1) diagonals.



ARM barrel shifter - rotations

For rotate right function, the right shift diagonal is enabled together with the complementary shift diagonal.

For example, on the 4-bit matrix rotate right one bit is implemented using the 'right1' and the 'left3' (3=4-1) diagonals.



ARM multiplier design

ARM1 no hardware for multiplication

ARM2 low-cost multiplication hardware: 32-bit result multiply and multiply-accumulate instructions

ARM6 high performance multiplication hardware: 64-bit result multiply and multiply-accumulate instructions

ARM multiplier design

ARM1 no hardware for multiplication

ARM2 low-cost multiplication hardware: 32-bit result multiply and multiply-accumulate instructions

ARM6 high performance multiplication hardware: 64-bit result multiply and multiply-accumulate instructions

ARM multiplier design

ARM1 no hardware for multiplication

ARM2 low-cost multiplication hardware: 32-bit result multiply and multiply-accumulate instructions

ARM6 high performance multiplication hardware: 64-bit result multiply and multiply-accumulate instructions

ARM low - cost multiplier

Low-cost multiplier employs the barrel shifter and ALU to generate a 2-bit product in each clock cycle.

Early-termination is used to stop the iterations when there are no more ones in the multiply register.

The multiplier logic implements a modified Booth's algorithm that allows to generate 2-bit sub-products in one step.

The additional hardware is limited to a dedicated two-bits-per-cycle shift register and a few gates for the control logic.

ARM low - cost multiplier

Low-cost multiplier employs the barrel shifter and ALU to generate a 2-bit product in each clock cycle.

Early-termination is used to stop the iterations when there are no more ones in the multiply register.

The multiplier logic implements a modified Booth's algorithm that allows to generate 2-bit sub-products in one step.

The additional hardware is limited to a dedicated two-bits-per-cycle shift register and a few gates for the control logic.

ARM low - cost multiplier

Low-cost multiplier employs the barrel shifter and ALU to generate a 2-bit product in each clock cycle.

Early-termination is used to stop the iterations when there are no more ones in the multiply register.

The multiplier logic implements a modified Booth's algorithm that allows to generate 2-bit sub-products in one step.

The additional hardware is limited to a dedicated two-bits-per-cycle shift register and a few gates for the control logic.

ARM low - cost multiplier

Low-cost multiplier employs the barrel shifter and ALU to generate a 2-bit product in each clock cycle.

Early-termination is used to stop the iterations when there are no more ones in the multiply register.

The multiplier logic implements a modified Booth's algorithm that allows to generate 2-bit sub-products in one step.

The additional hardware is limited to a dedicated two-bits-per-cycle shift register and a few gates for the control logic.

ARM low - cost multiplier

Carry-in	multiplier	logic shift	ALU	Carry-out
0	*0	2N	A+0	0
	*1	2N	A+B	0
	*2	2N+1	A-B	1
	*3	2N	A-B	1
1	*0	2N	A+B	0
	*1	2N+1	A+B	0
	*2	2N	A-B	1
	*3	2N	A+0	1

ARM low - cost multiplier

Carry-in	multiplier	logic shift	ALU	Carry-out
0	*0	2N	A+0	0
	*1	2N	A+B	0
	*2	2N+1	A-B	1
	*3	2N	A-B	1
1	*0	2N	A+B	0
	*1	2N+1	A+B	0
	*2	2N	A-B	1
	*3	2N	A+0	1

ARM low - cost multiplier

Carry-in	multiplier	logic shift	ALU	Carry-out
0	*0	2N	A+0	0
	*1	2N	A+B	0
	*2	2N+1	A-B	1
	*3	2N	A-B	1
1	*0	2N	A+B	0
	*1	2N+1	A+B	0
	*2	2N	A-B	1
	*3	2N	A+0	1

ARM low - cost multiplier

Carry-in	multiplier	logic shift	ALU	Carry-out
0	*0	2N	A+0	0
	*1	2N	A+B	0
	*2	2N+1	A-B	1
	*3	2N	A-B	1
1	*0	2N	A+B	0
	*1	2N+1	A+B	0
	*2	2N	A-B	1
	*3	2N	A+0	1

ARM high-speed multiplier

High performance multiplication employs redundant binary representation to avoid the carry-propagate delays associated with adding partial products together.

Intermediate results are held as partial sums and partial carries. These results are added in the main ALU at the end of multiplication.

During the multiplication the partial sums and carries are combined in carry save adders where carries may propagate only one bit per addition stage.

ARM high-speed multiplier

High performance multiplication employs redundant binary representation to avoid the carry-propagate delays associated with adding partial products together.

Intermediate results are held as partial sums and partial carries. These results are added in the main ALU at the end of multiplication.

During the multiplication the partial sums and carries are combined in carry save adders where carries may propagate only one bit per addition stage.

ARM high-speed multiplier

High performance multiplication employs redundant binary representation to avoid the carry-propagate delays associated with adding partial products together.

Intermediate results are held as partial sums and partial carries. These results are added in the main ALU at the end of multiplication.

During the multiplication the partial sums and carries are combined in carry save adders where carries may propagate only one bit per addition stage.

Carry - save adders

The simplest carry-save adder has 3 inputs and 2 outputs that accept a partial sum, a partial carry and a partial product, all of the same binary weight.

The output produce a new partial sum and a new partial carry where the carry has twice the weight of the sum.

The internal logic of each simple adder is identical to a conventional full adder, but the interconnection structure is different.

Carry - save adders

The simplest carry-save adder has 3 inputs and 2 outputs that accept a partial sum, a partial carry and a partial product, all of the same binary weight.

The output produce a new partial sum and a new partial carry where the carry has twice the weight of the sum.

The internal logic of each simple adder is identical to a conventional full adder, but the interconnection structure is different.

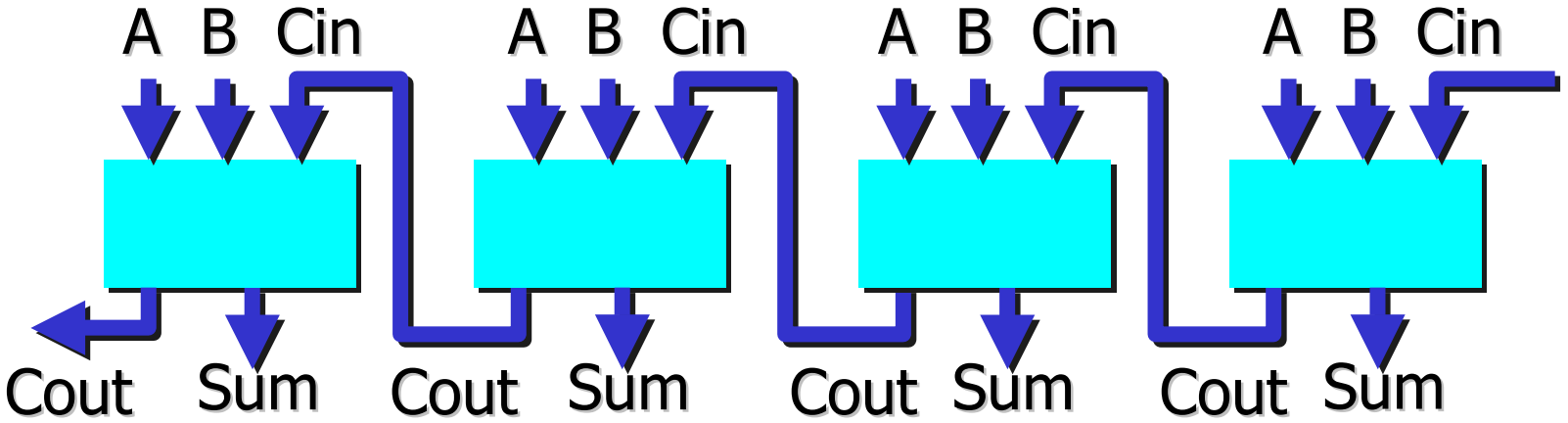
Carry - save adders

The simplest carry-save adder has 3 inputs and 2 outputs that accept a partial sum, a partial carry and a partial product, all of the same binary weight.

The output produce a new partial sum and a new partial carry where the carry has twice the weight of the sum.

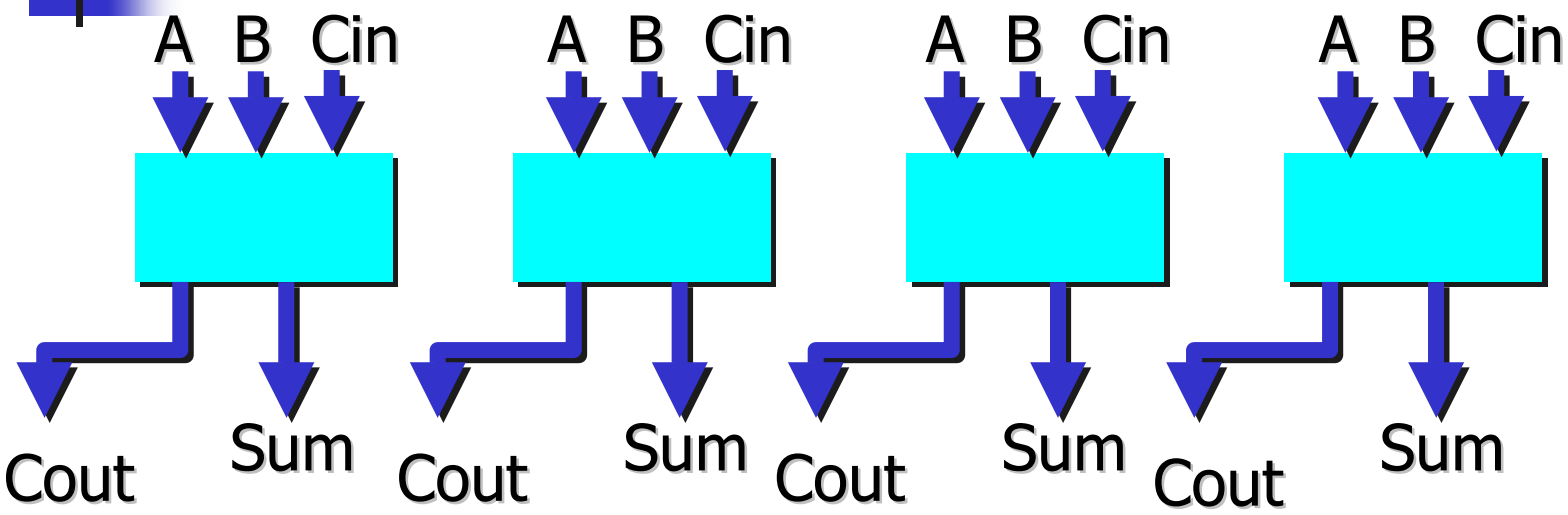
The internal logic of each simple adder is identical to a conventional full adder, but the interconnection structure is different.

Carry - propagate adder



The carry-propagate adder takes two conventional (irredundant) binary numbers as inputs and produces a binary sum.

Carry - save adder



The carry-save adder takes one binary and one redundant (partial sum and partial carry) input and produces a sum in a redundant binary representation.

Carry - save adder

During the iterative multiplication stages, the sum is fed back and combined with one partial product in each iteration.

When all partial products have been added, the redundant representation is converted into a conventional binary number by adding the partial sum and partial carry in the carry propagate adder.

Carry - save adder

During the iterative multiplication stages, the sum is fed back and combined with one partial product in each iteration.

When all partial products have been added, the redundant representation is converted into a conventional binary number by adding the partial sum and partial carry in the carry propagate adder.

High performance multiplier

High speed multipliers have several layers of carry-save adders, each handling one partial product.

In some ARM cores the carry-save array has four layers of adders, each handling two multiplier bits (see Booth algorithm), so the array can multiply eight bits per clock cycle.

The array is cycled up to four times, using early termination, to complete multiplication in less than four cycles when the multiplier has sufficient zeros in the top bits.

High performance multiplier

High speed multipliers have several layers of carry-save adders, each handling one partial product.

In some ARM cores the carry-save array has four layers of adders, each handling two multiplier bits (see Booth algorithm), so the array can multiply eight bits per clock cycle.

The array is cycled up to four times, using early termination, to complete multiplication in less than four cycles when the multiplier has sufficient zeros in the top bits.

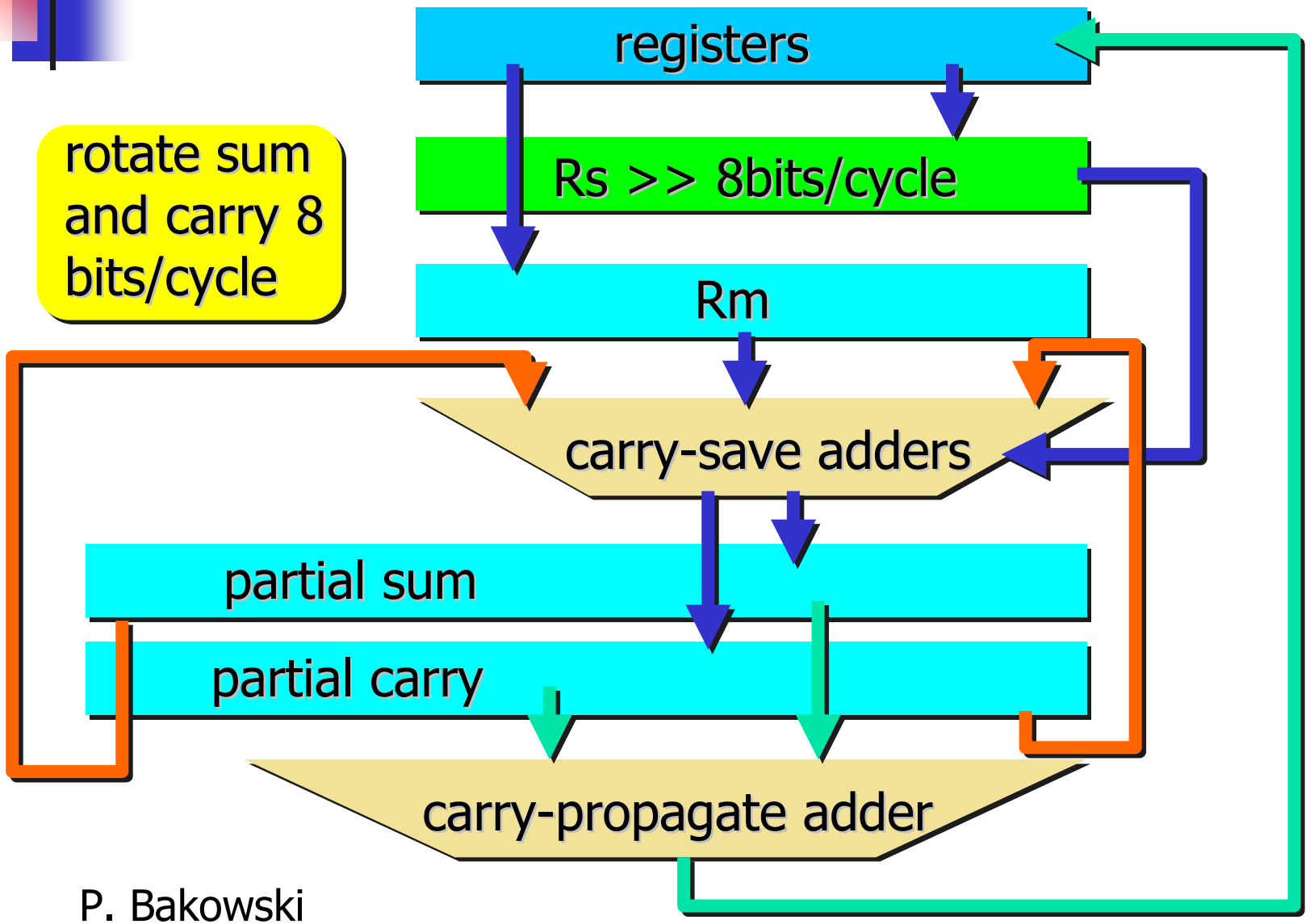
High performance multiplier

High speed multipliers have several layers of carry-save adders, each handling one partial product.

In some ARM cores the carry-save array has four layers of adders, each handling two multiplier bits (see Booth algorithm), so the array can multiply eight bits per clock cycle.

The array is cycled up to four times, using early termination, to complete multiplication in less than four cycles when the multiplier has sufficient number of zeros in the top bits.

High speed multiplier organization

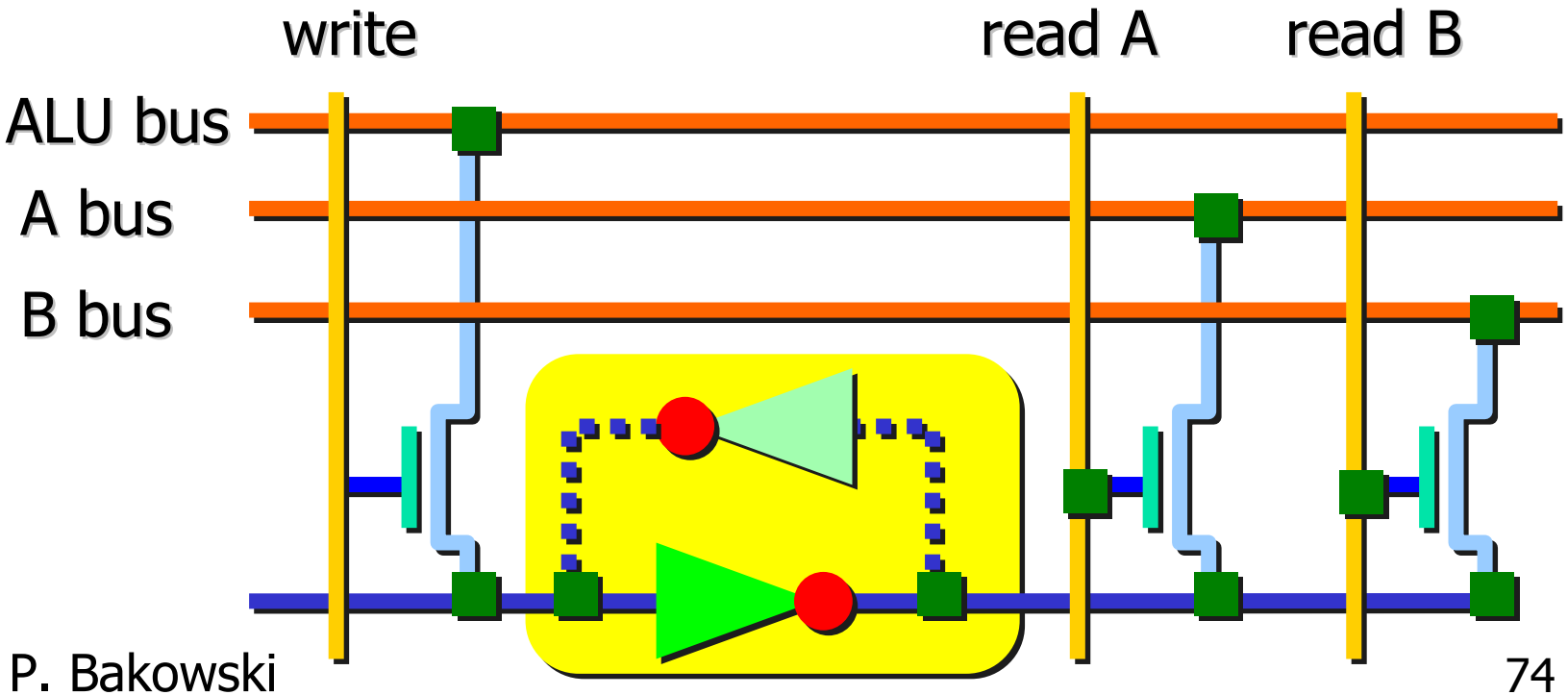


ARM register bank

ARM has 31 general-purpose 32-bit registers containing almost 1 Kbytes of data.

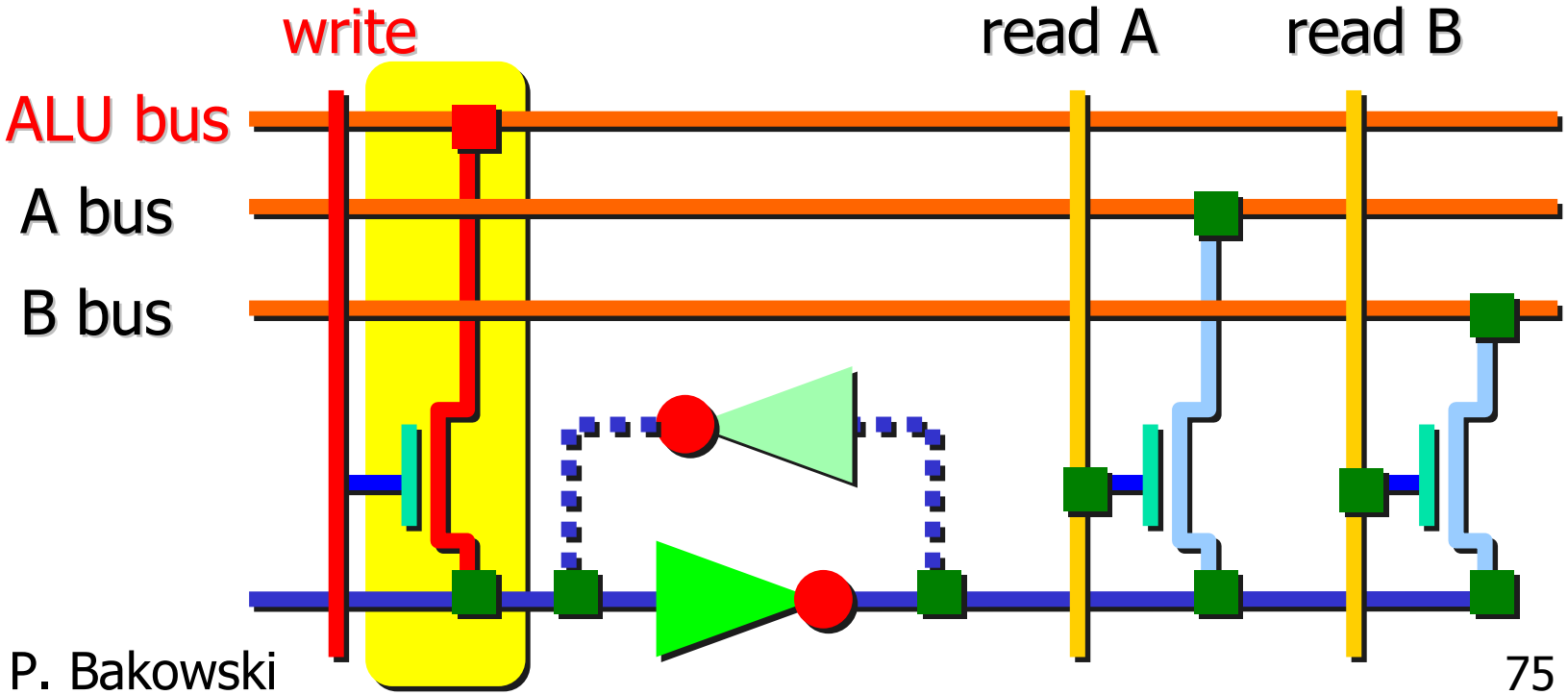
ARM register cell

The transistor circuit of the register cell used in ARM cores up to the ARM6 is based on an asymmetric cross-coupled pair of CMOS inverters which is overdriven by a strong signal from the ALU bus when the register contents are changed.



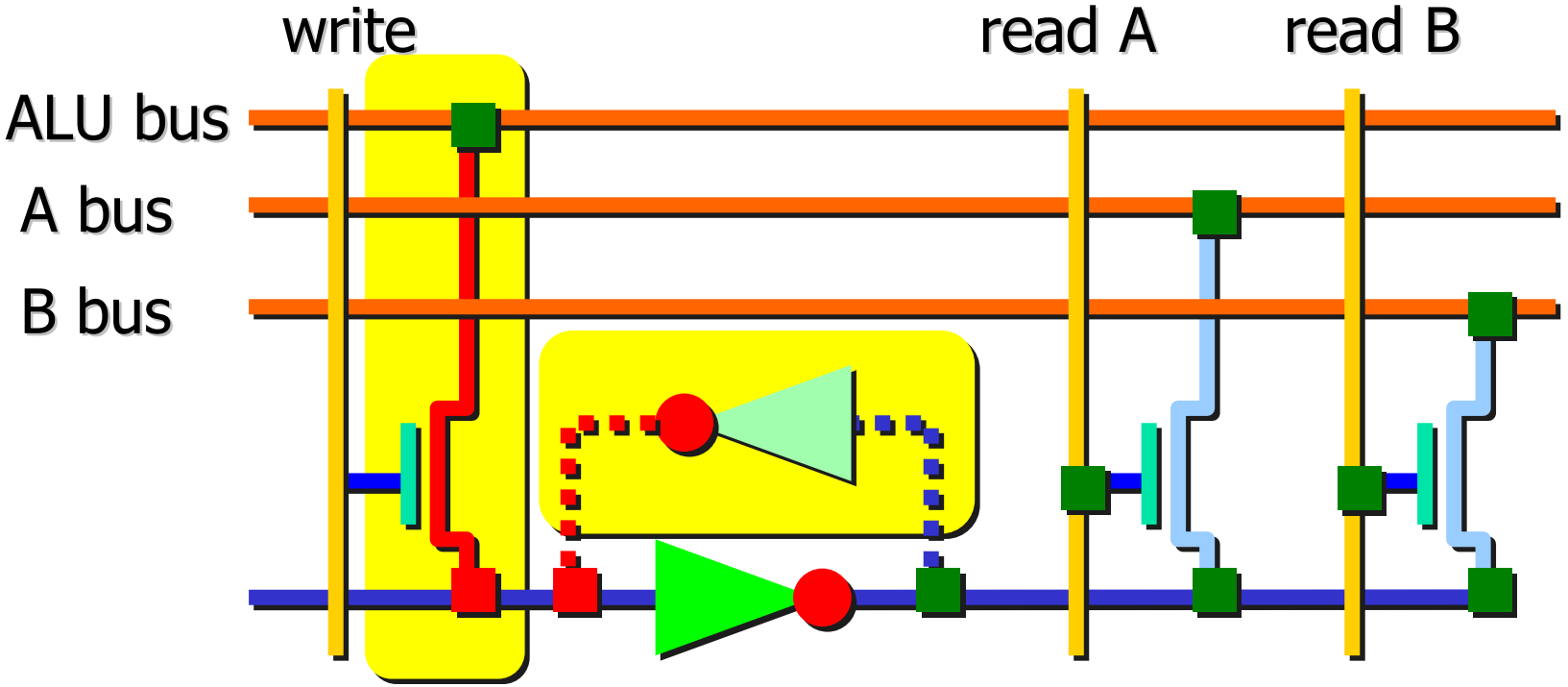
ARM register cell

The transistor circuit of the register cell used in ARM cores up to the ARM6 is based on an asymmetric cross-coupled pair of CMOS inverters which is overdriven by a strong signal from the ALU bus when the register contents are changed.



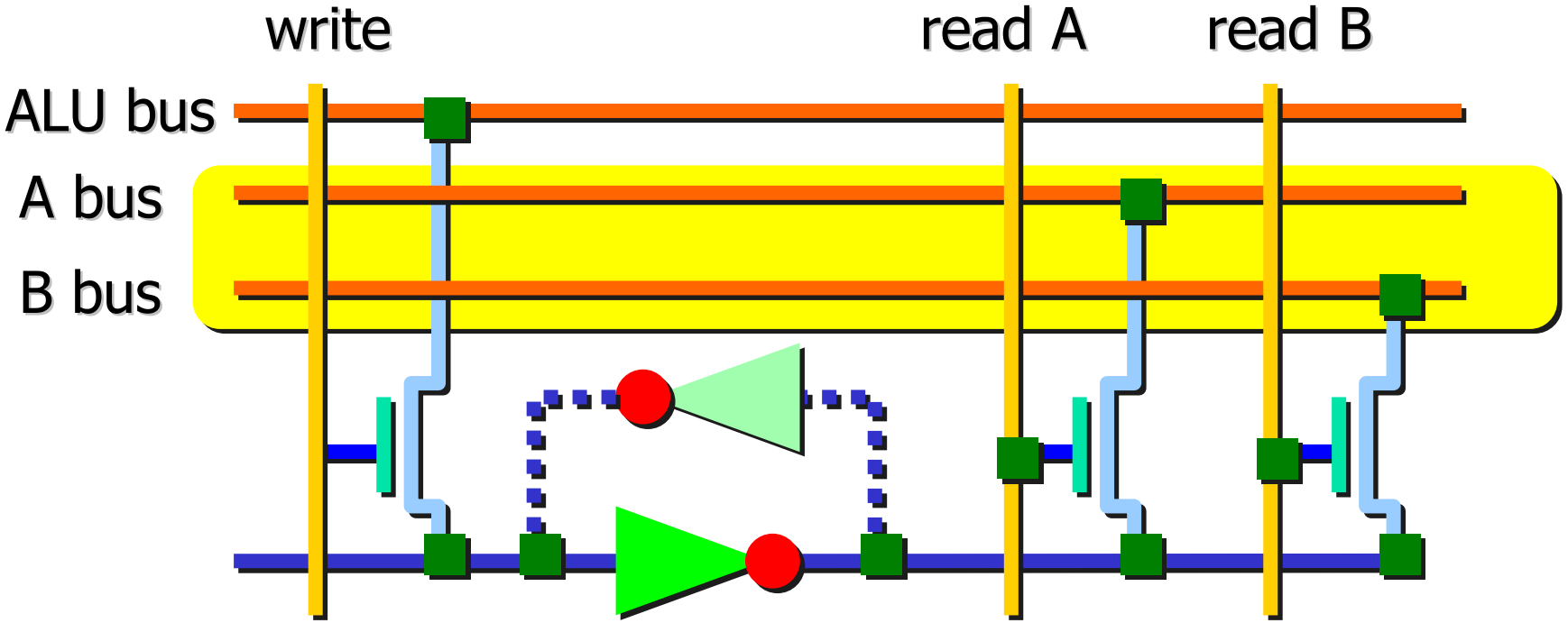
ARM register cell

The feedback inverter is made weak in order to minimize the cell's resistance to the new value.



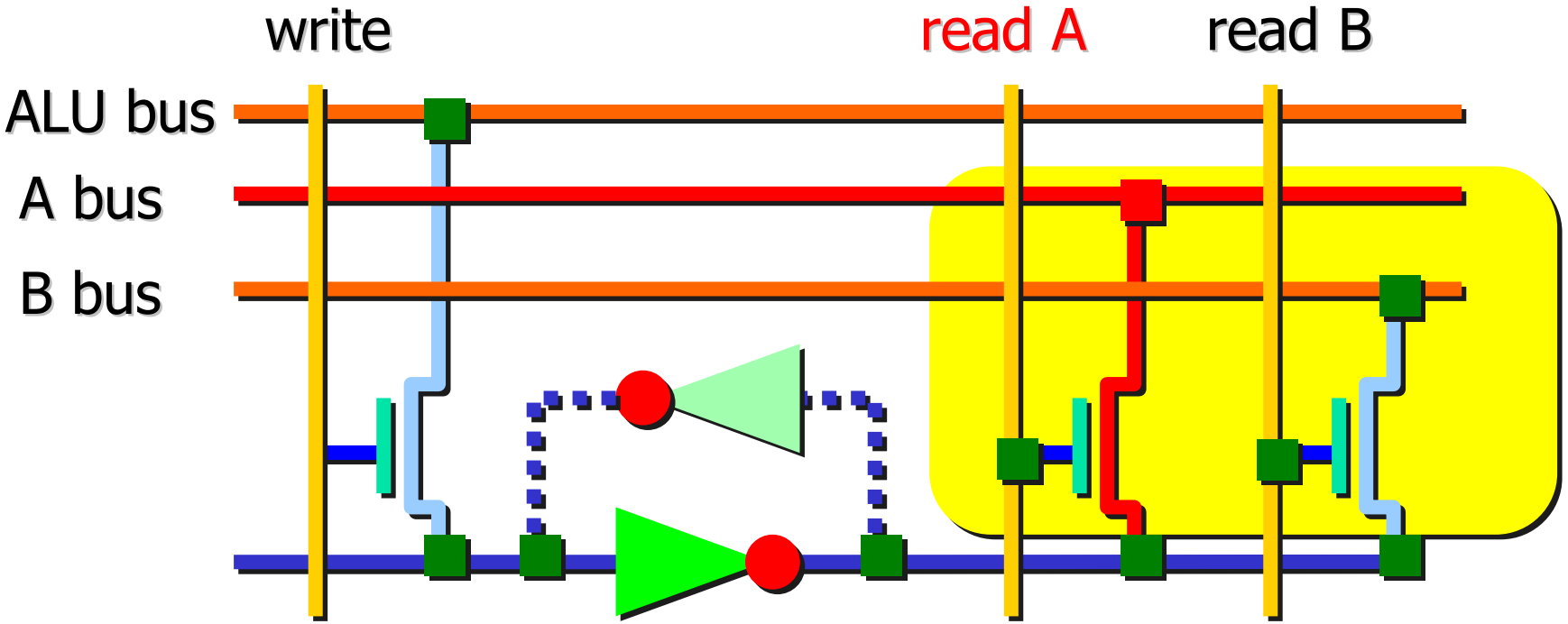
ARM register cell

The A and B buses are pre-charged to Vdd during phase 2 of the clock cycle, so the register cell needs only discharge the read buses, which it does through n-type pass transistors when the read lines are enabled.



ARM register cell

The A and B buses are pre-charged to Vdd during phase 2 of the clock cycle, so the register cell needs only discharge the read buses, which it does through n-type pass transistors when the read lines are enabled.

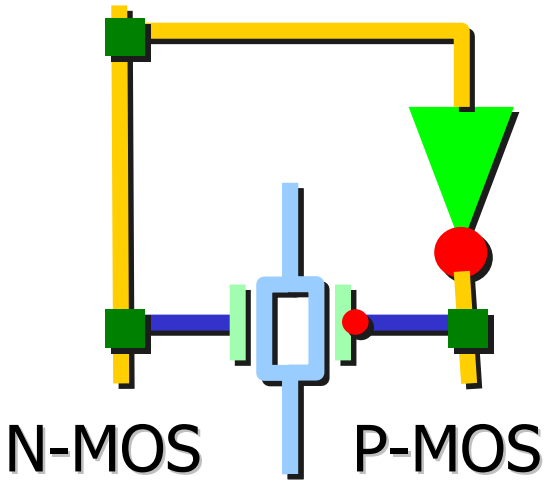


ARM register cell

This register cell works well with a 5 V supply.

In new designs ARM uses lower supply voltages where writing a '1' through the n-type pass transistor would be impossible.

Instead it uses a full CMOS transmission gate requiring complementary write enable control lines.

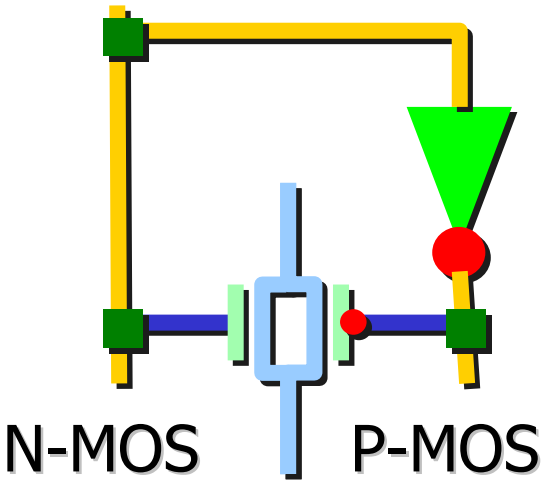


ARM register cell

This register cell works well with a 5 V supply.

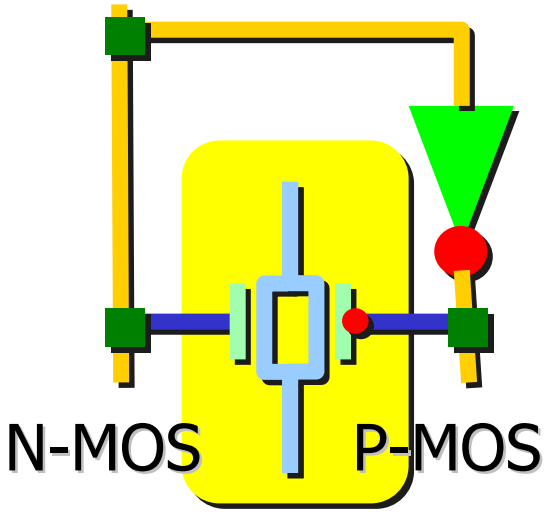
In new designs ARM uses lower supply voltages where writing a '1' through the n-type pass transistor would be impossible.

Instead it uses a full CMOS transmission gate requiring complementary write enable control lines.



ARM register cell

This register cell works well with a 5 V supply.
In new designs ARM uses lower supply voltages where writing a '1' through the n-type pass transistor would be impossible.
Instead it uses a full CMOS transmission gate requiring complementary write enable control lines.



ARM register bank

The register cells are arranged in columns to form a 32-bit register, and the columns are packed together to form the complete register bank.

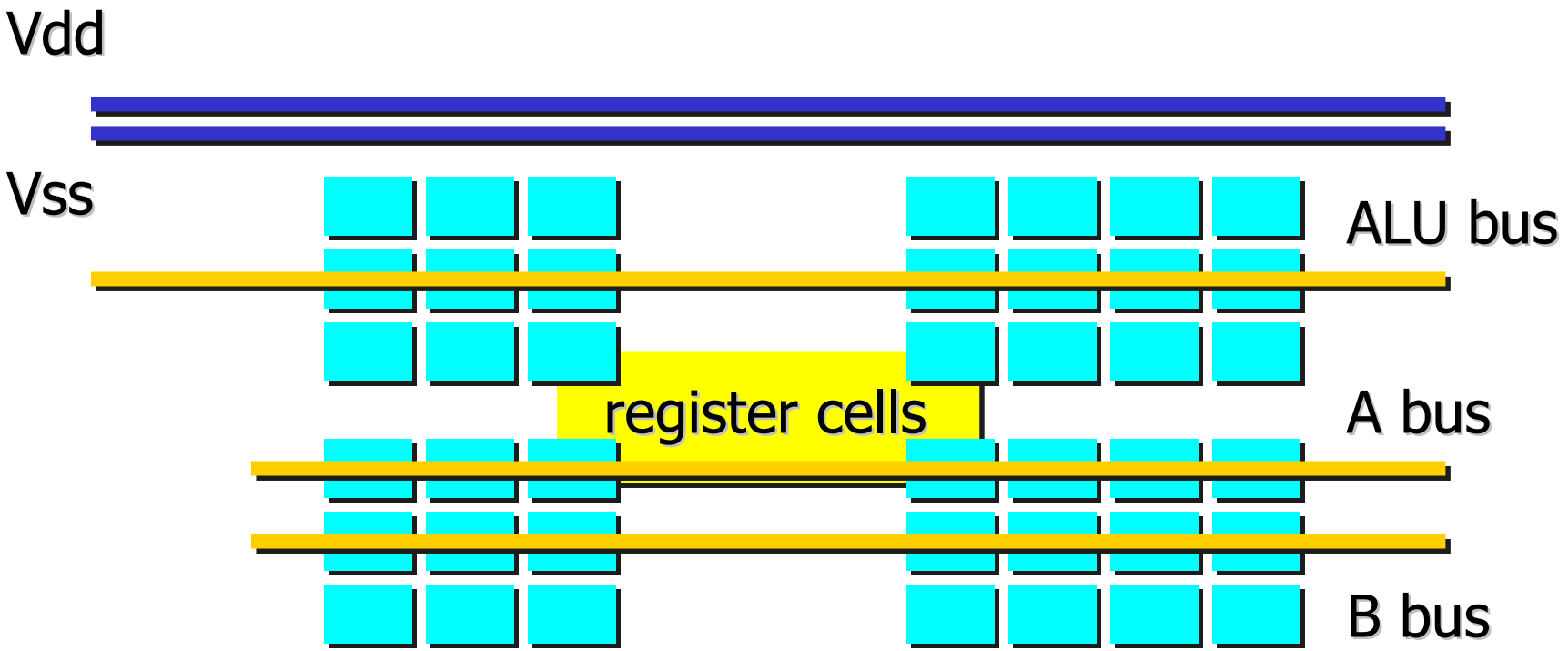
The decoders for the read and write enable lines are packed above the columns; the enable lines run vertically and the data buses horizontally across the array of register cells.

ARM register bank

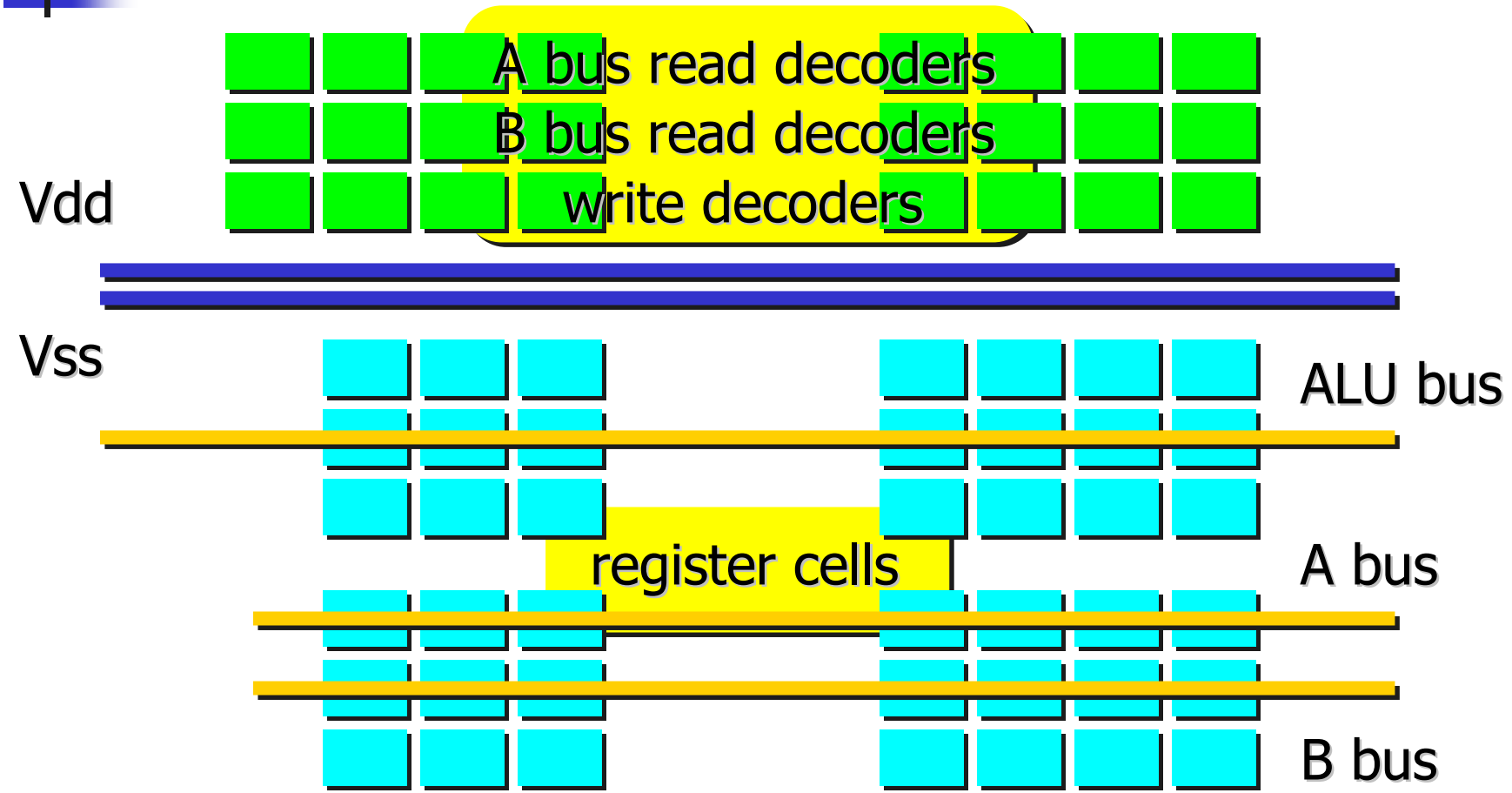
The register cells are arranged in columns to form a 32-bit register, and the columns are packed together to form the complete register bank.

The decoders for the read and write enable lines are packed above the columns; the enable lines run vertically and the data buses horizontally across the array of register cells.

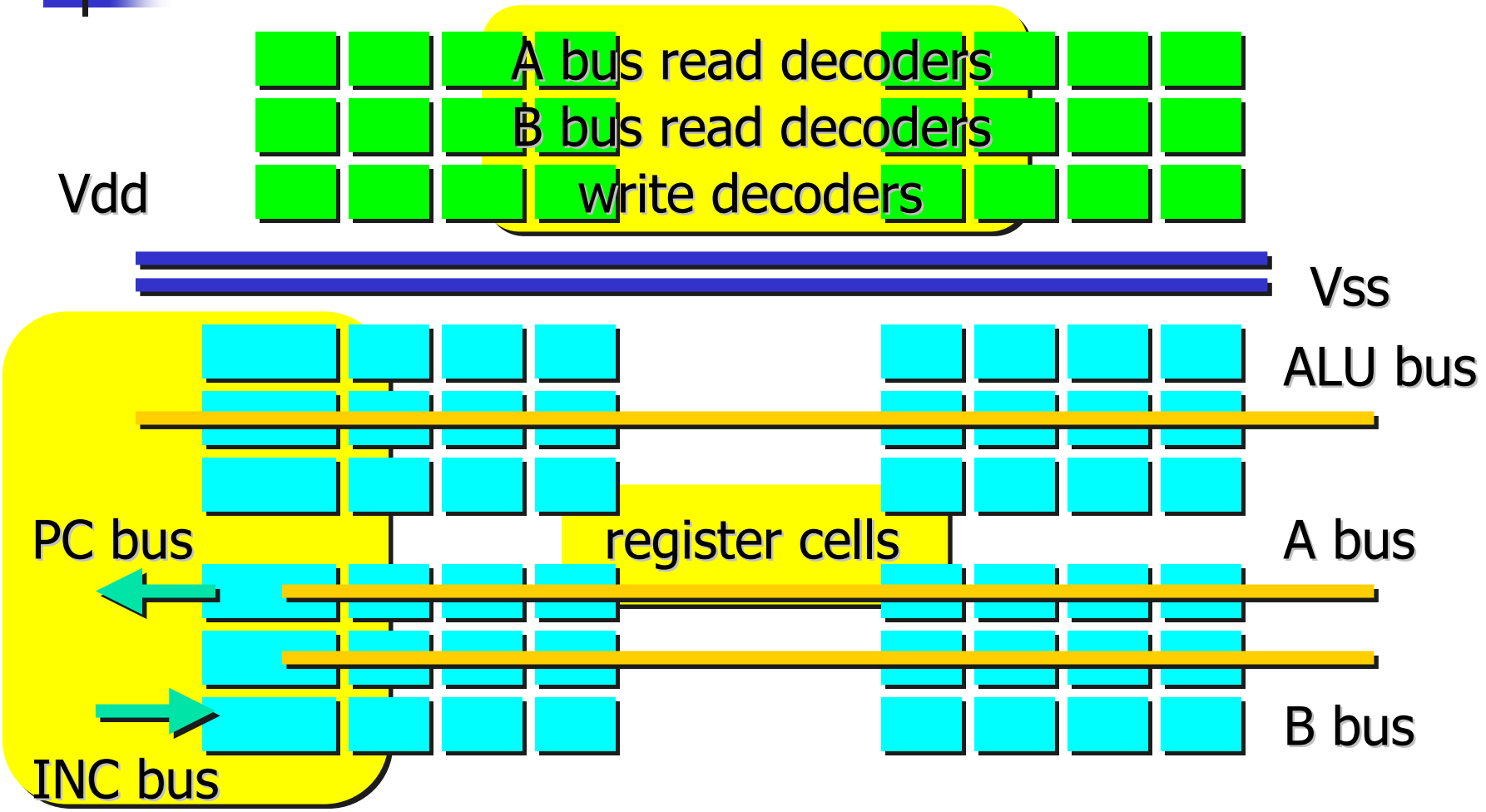
ARM register bank



ARM register bank

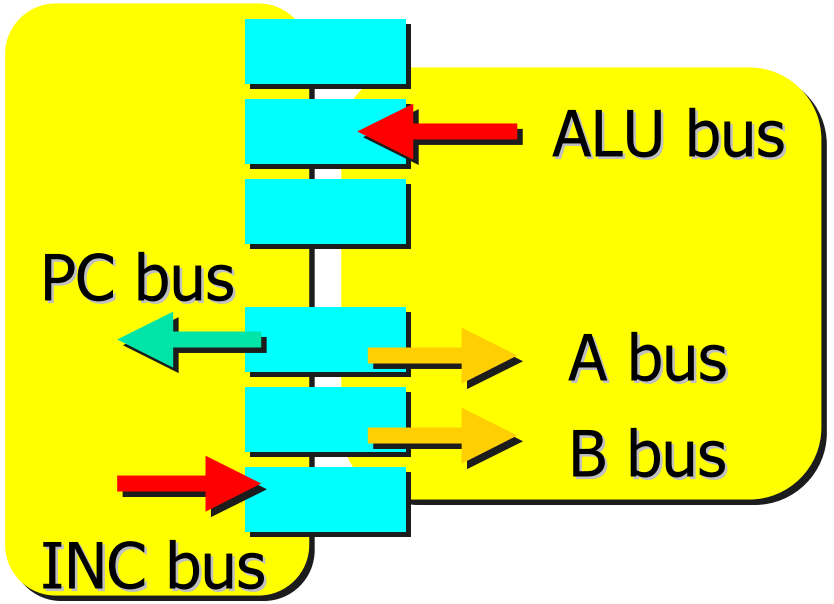


ARM register bank



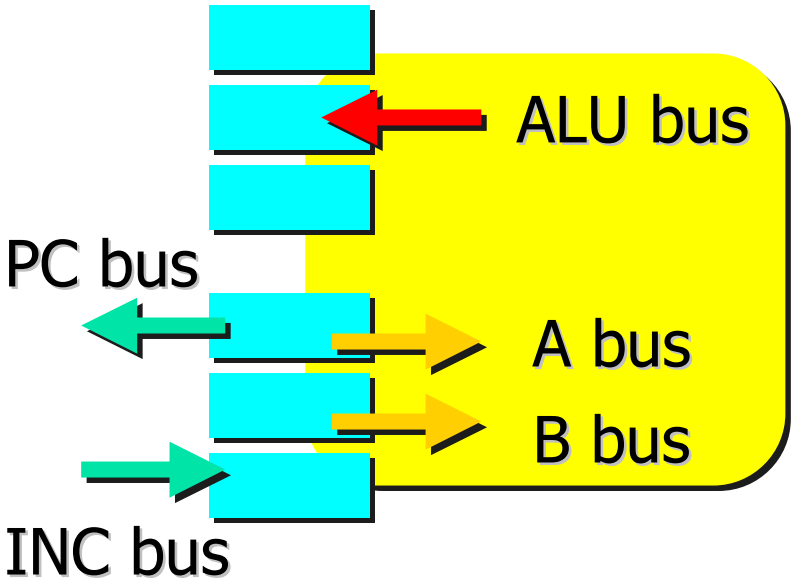
ARM register bank

The ARM program counter register is physically part of the register bank, but it has two write and three read ports whereas the other registers have one write and two read ports.



ARM register bank

The ARM program counter register is physically part of the register bank, but it has two write and three read ports whereas the other registers have one write and two read ports.



ARM data path layout

The ARM datapath is laid out to a constant pitch per bit.

Each function is laid out to this pitch.

The buses pass over the functional blocks.

The order of the blocks is chosen to minimize the number of additional buses passing over more complex functions.

ARM data path layout

The ARM datapath is laid out to a constant pitch per bit.

Each function is laid out to this pitch.

The buses pass over the functional blocks.

The order of the blocks is chosen to minimize the number of additional buses passing over more complex functions.

ARM data path layout

The ARM datapath is laid out to a constant pitch per bit.

Each function is laid out to this pitch.

The buses pass over the functional blocks.

The order of the blocks is chosen to minimize the number of additional buses passing over more complex functions.

ARM data path layout

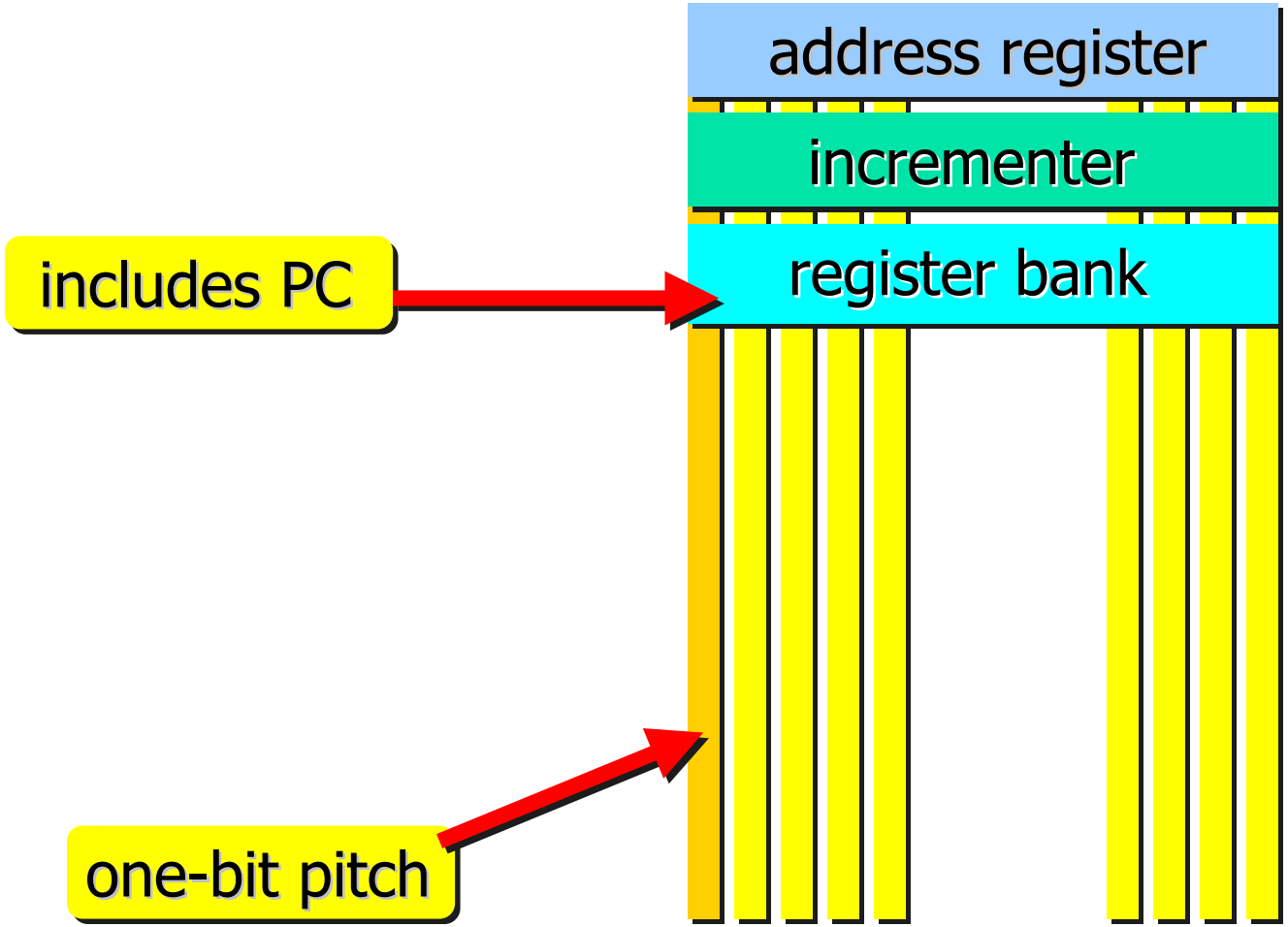
The ARM datapath is laid out to a constant pitch per bit.

Each function is laid out to this pitch.

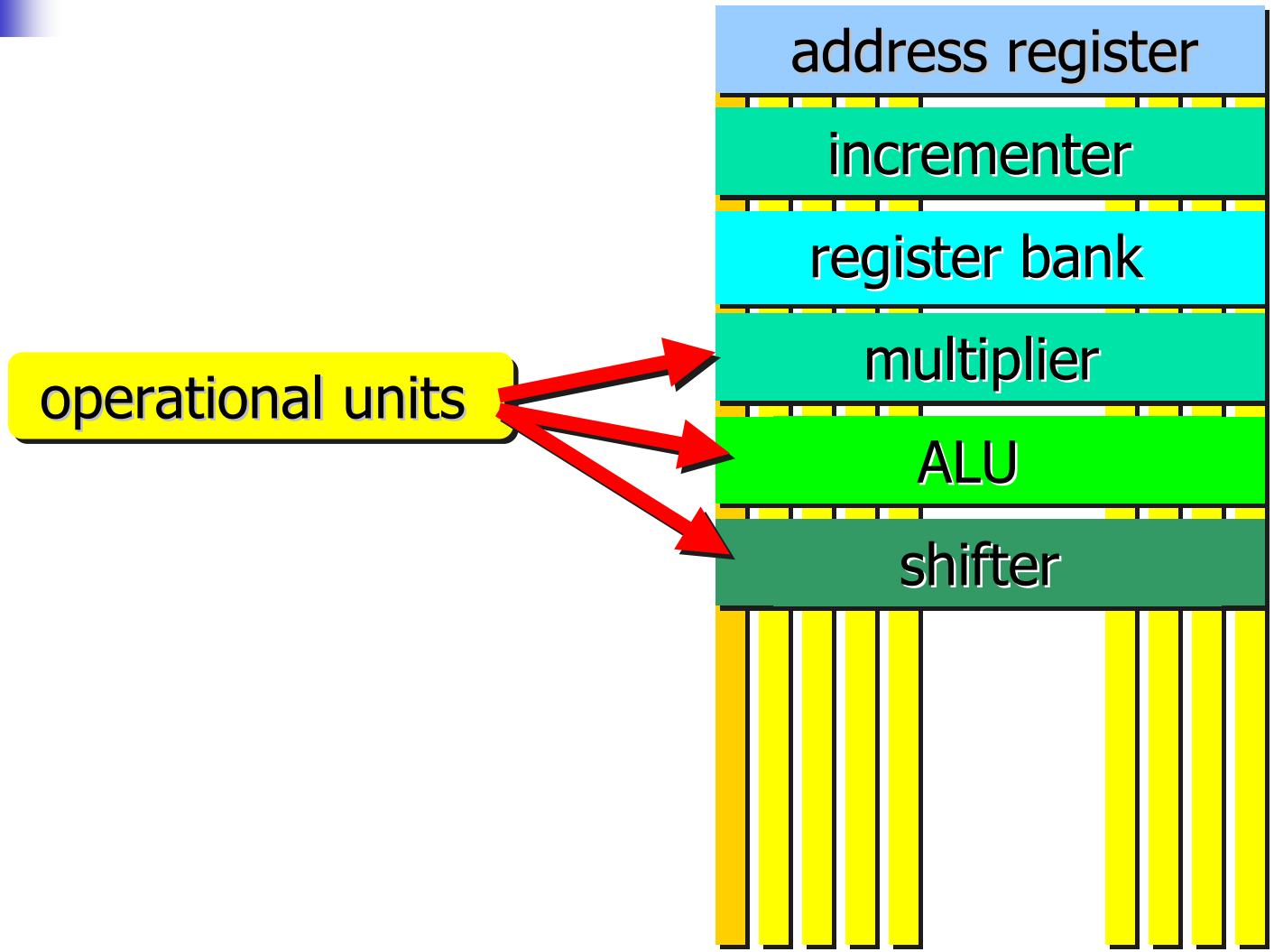
The buses pass over the functional blocks.

The order of the blocks is chosen to minimize the number of additional buses passing over more complex functions.

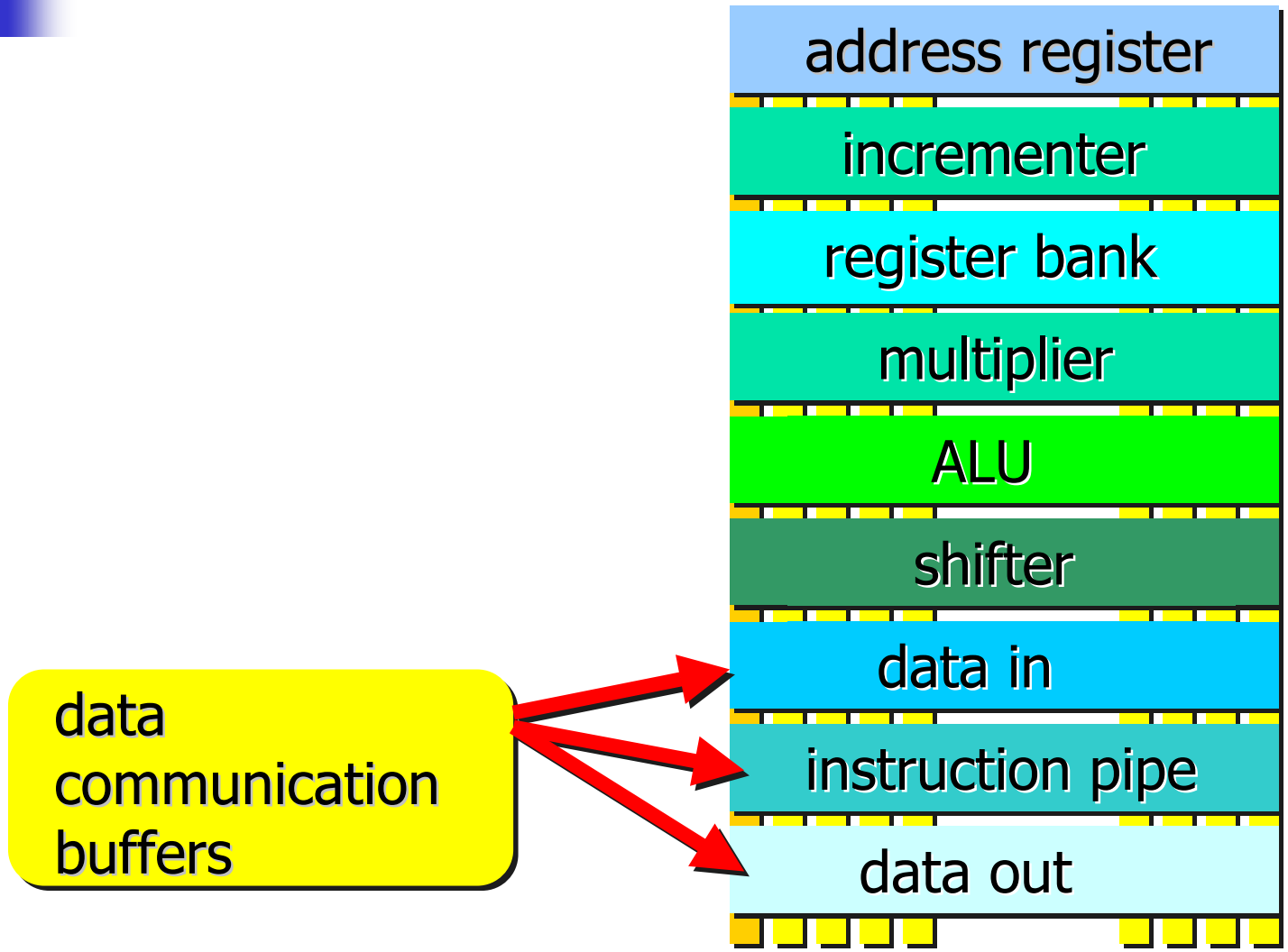
ARM data path layout & buses



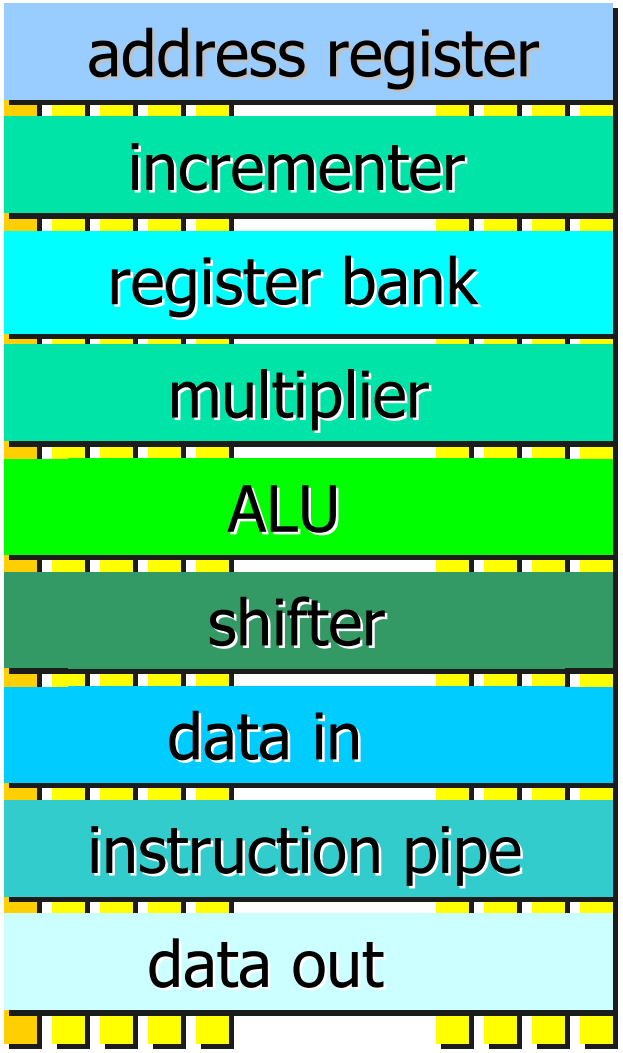
ARM data path layout & buses



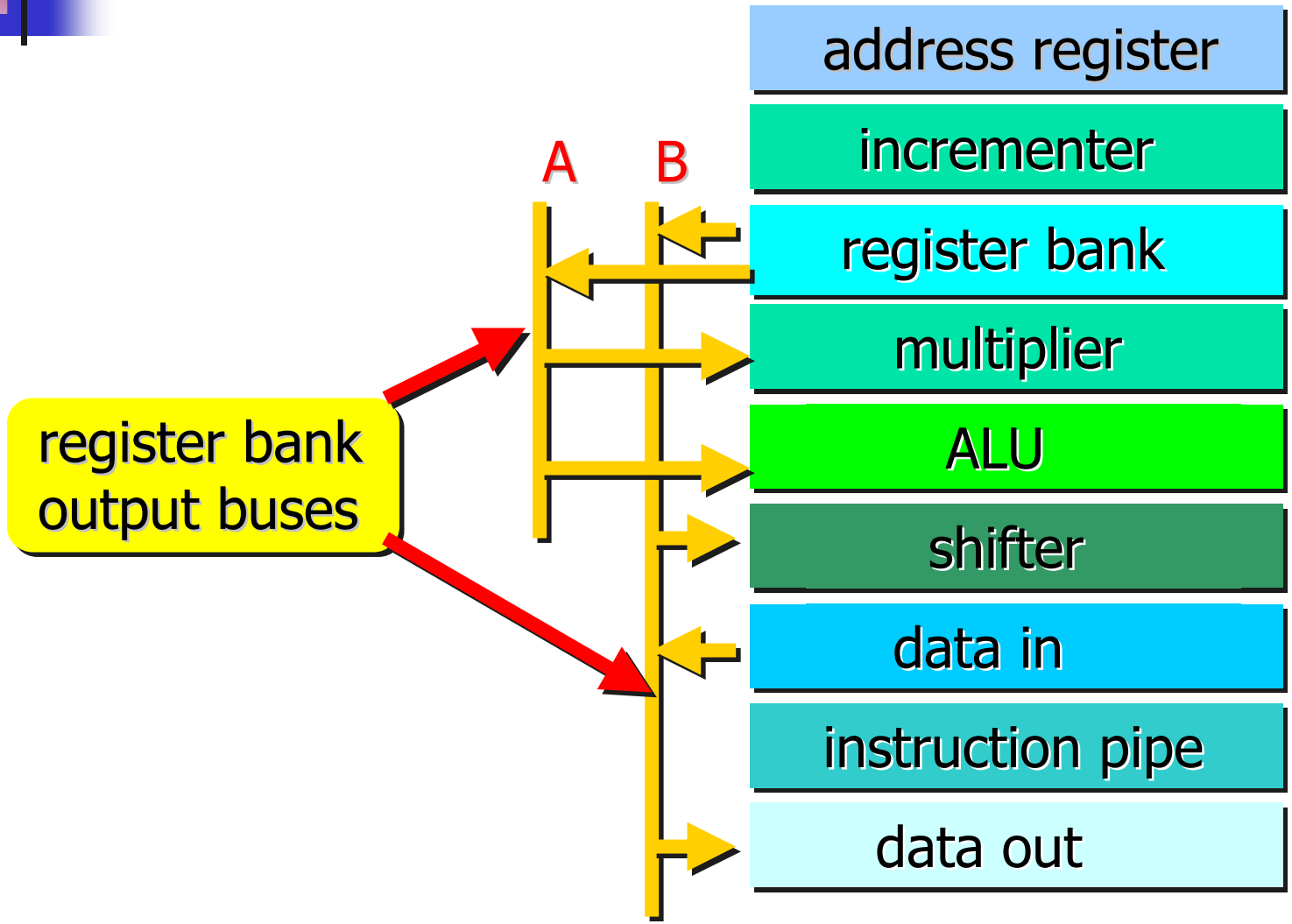
ARM data path layout & buses



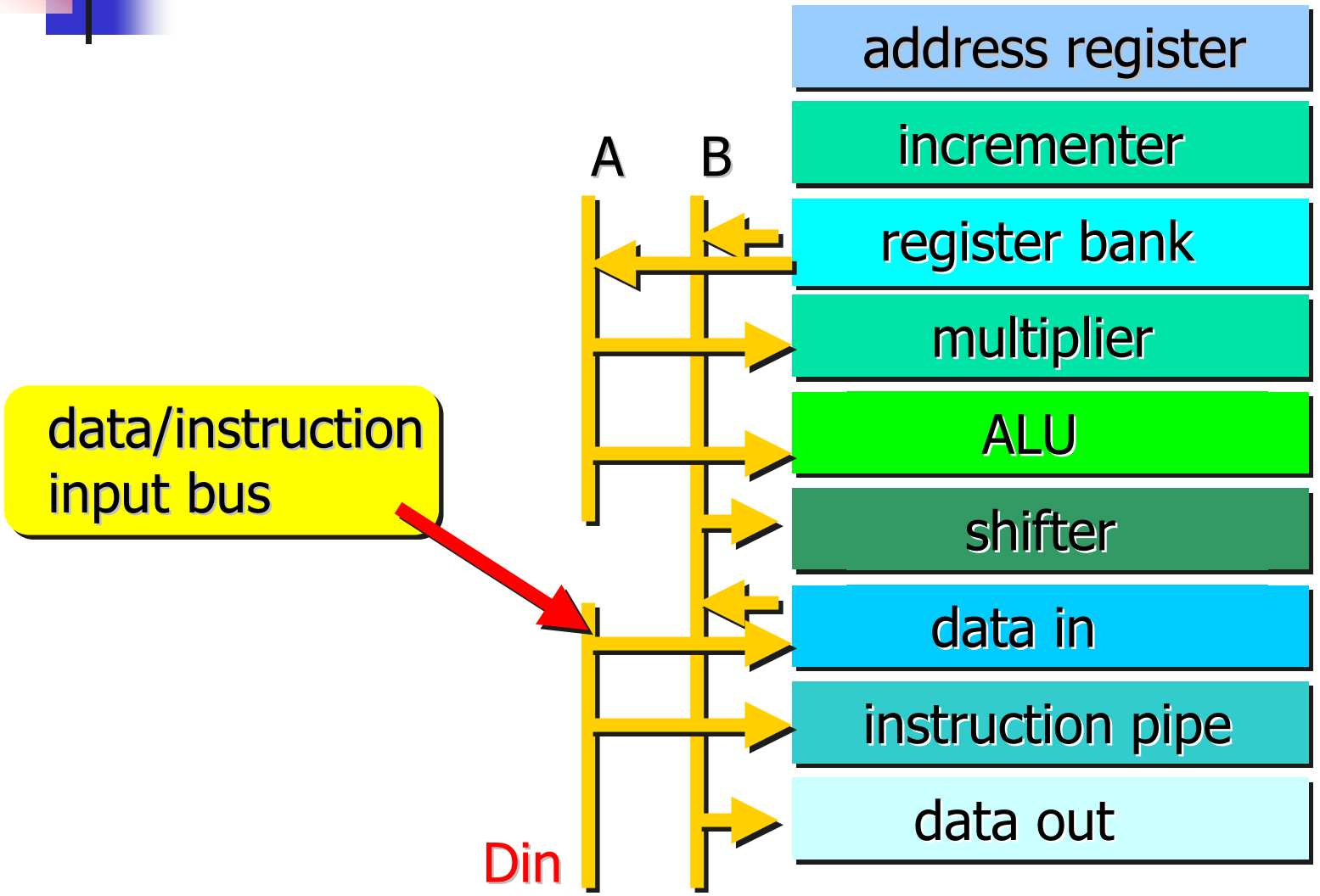
ARM data path layout & buses



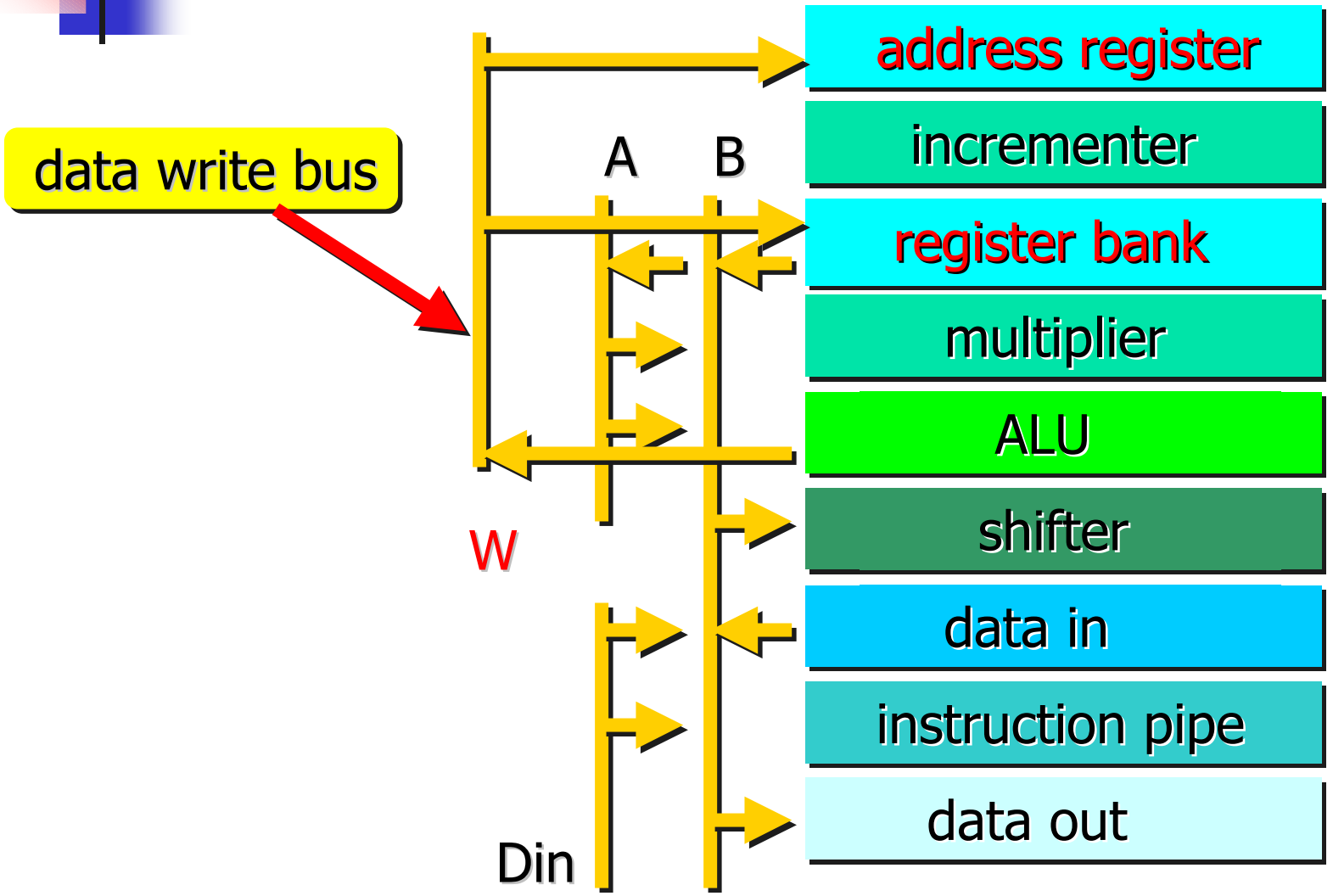
ARM data path layout & buses



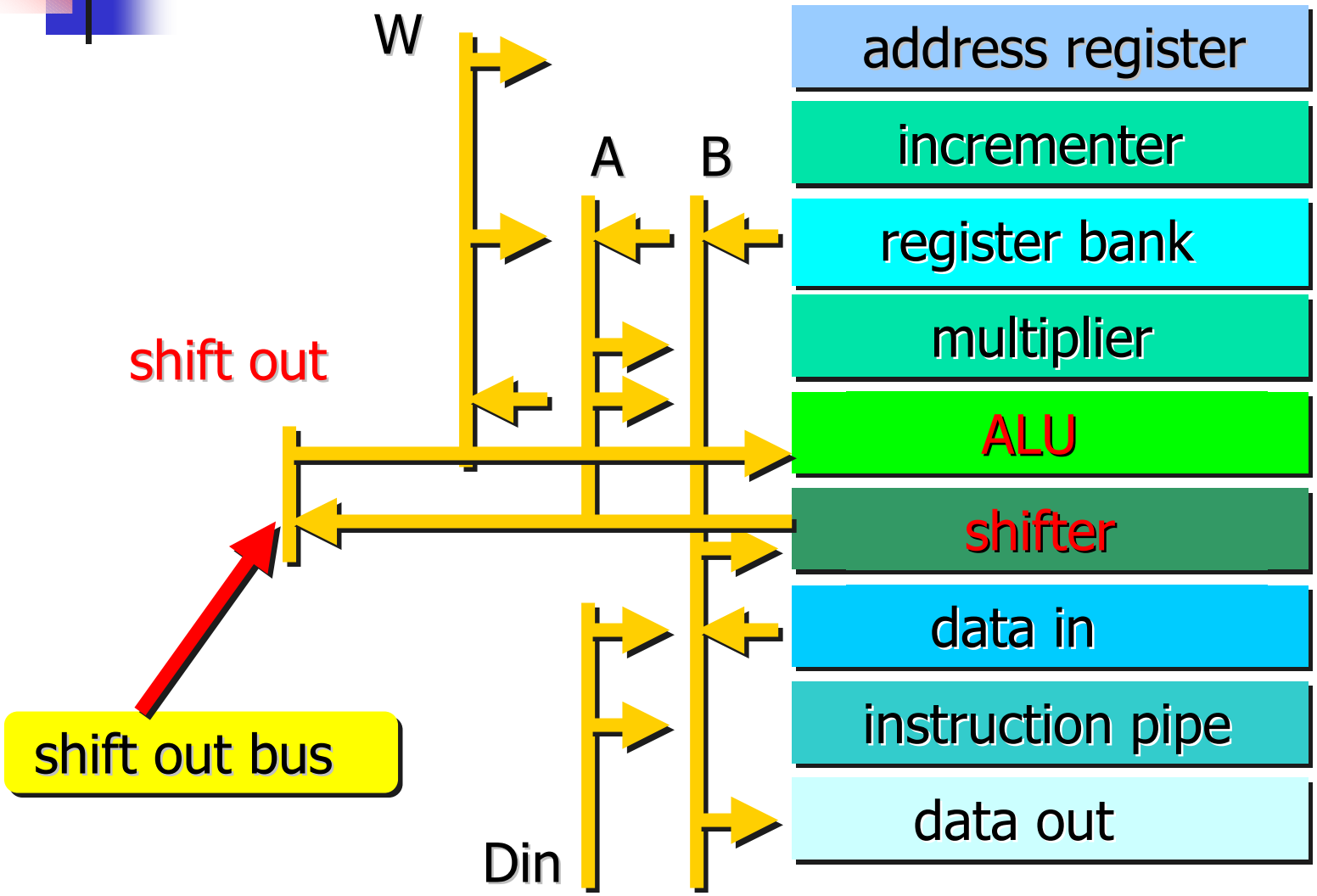
ARM data path layout & buses



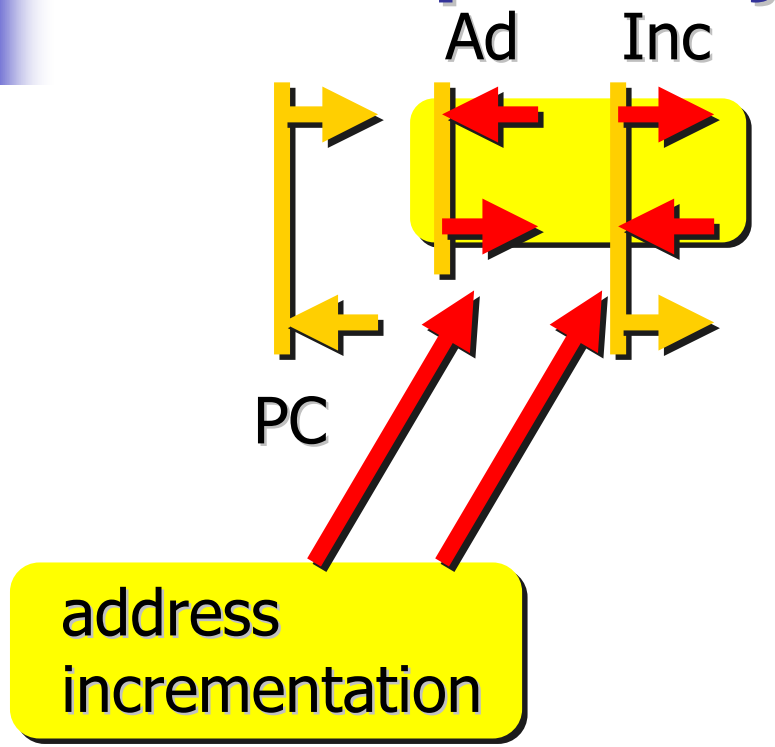
ARM data path layout & buses



ARM data path layout & buses



ARM data path layout & buses



address register

incrementer

register bank

multiplier

ALU

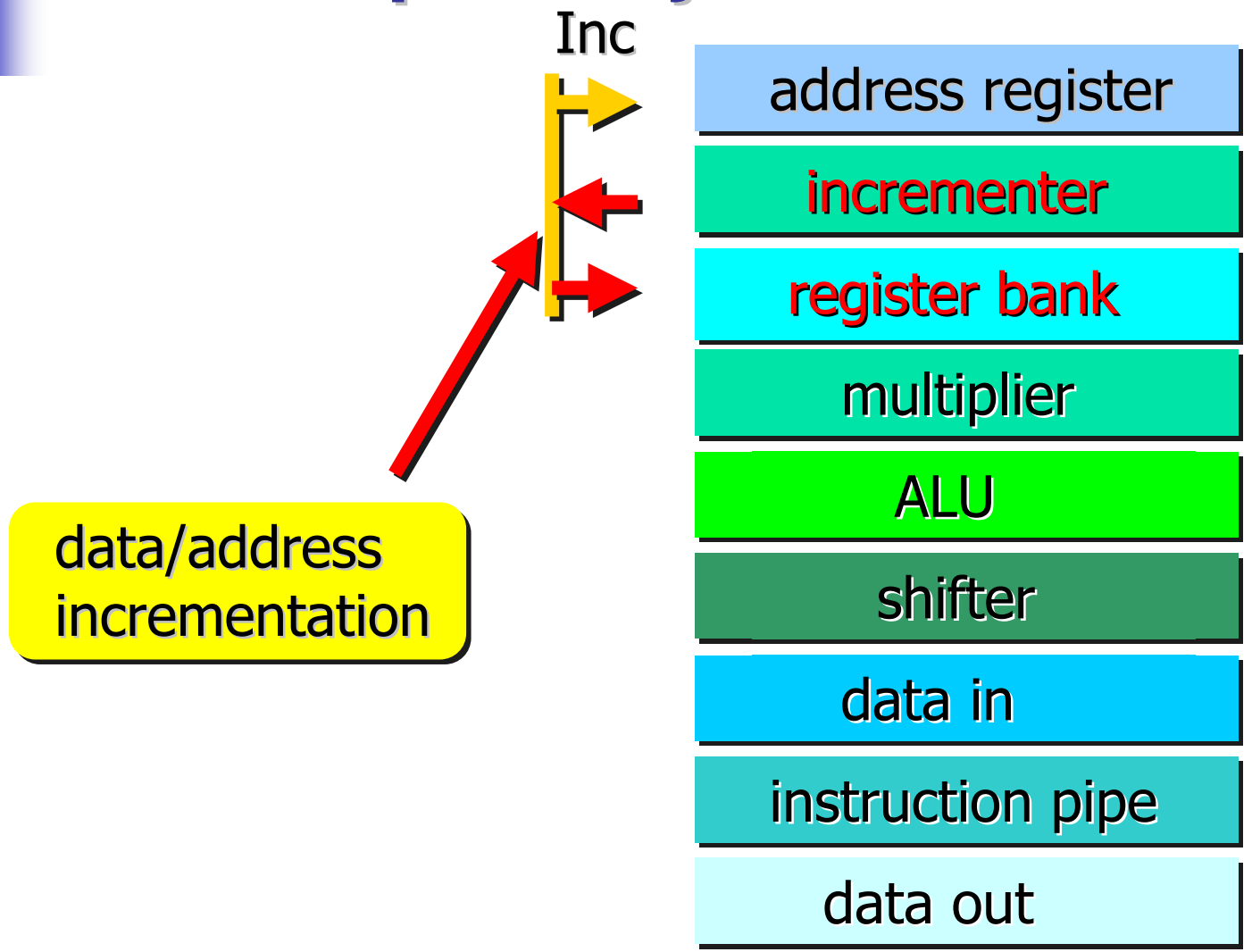
shifter

data in

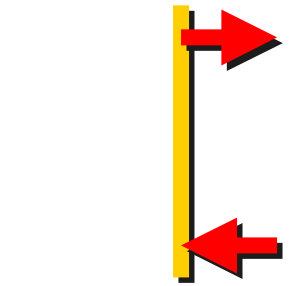
instruction pipe

data out

ARM data path layout & buses



ARM data path layout & buses



address register

incrementer

register bank

multiplier

ALU

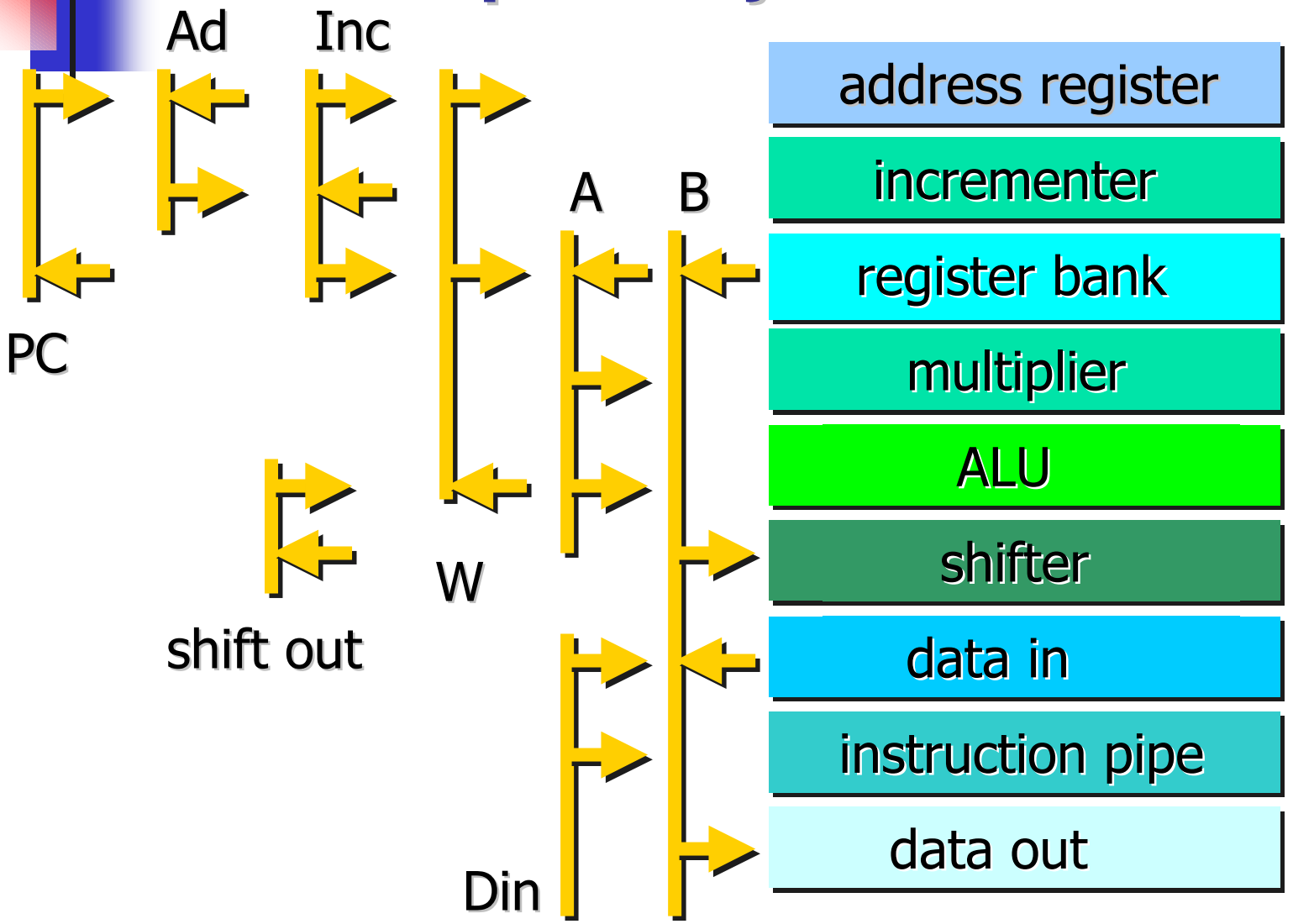
shifter

data in

instruction pipe

data out

ARM data path layout & buses



ARM control path

The control path in simpler ARM cores has three structural components:

- an instruction decoder PLA
- distributed secondary control associated with main functional units
- decentralized control units for specific instructions

ARM control path

The control path in simpler ARM cores has three structural components:

- an instruction decoder PLA
- distributed secondary control associated with main functional units
- decentralized control units for specific instructions

ARM control path

The control path in simpler ARM cores has three structural components:

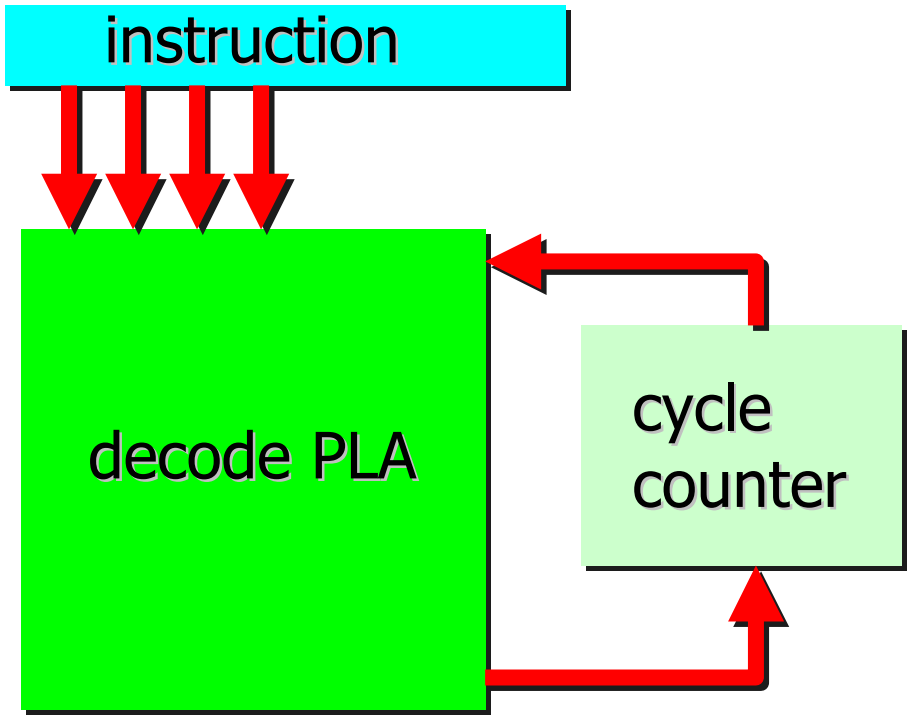
- an instruction decoder PLA
- distributed secondary control associated with main functional units
- decentralized control units for specific instructions

ARM control path

The control path in simpler ARM cores has three structural components:

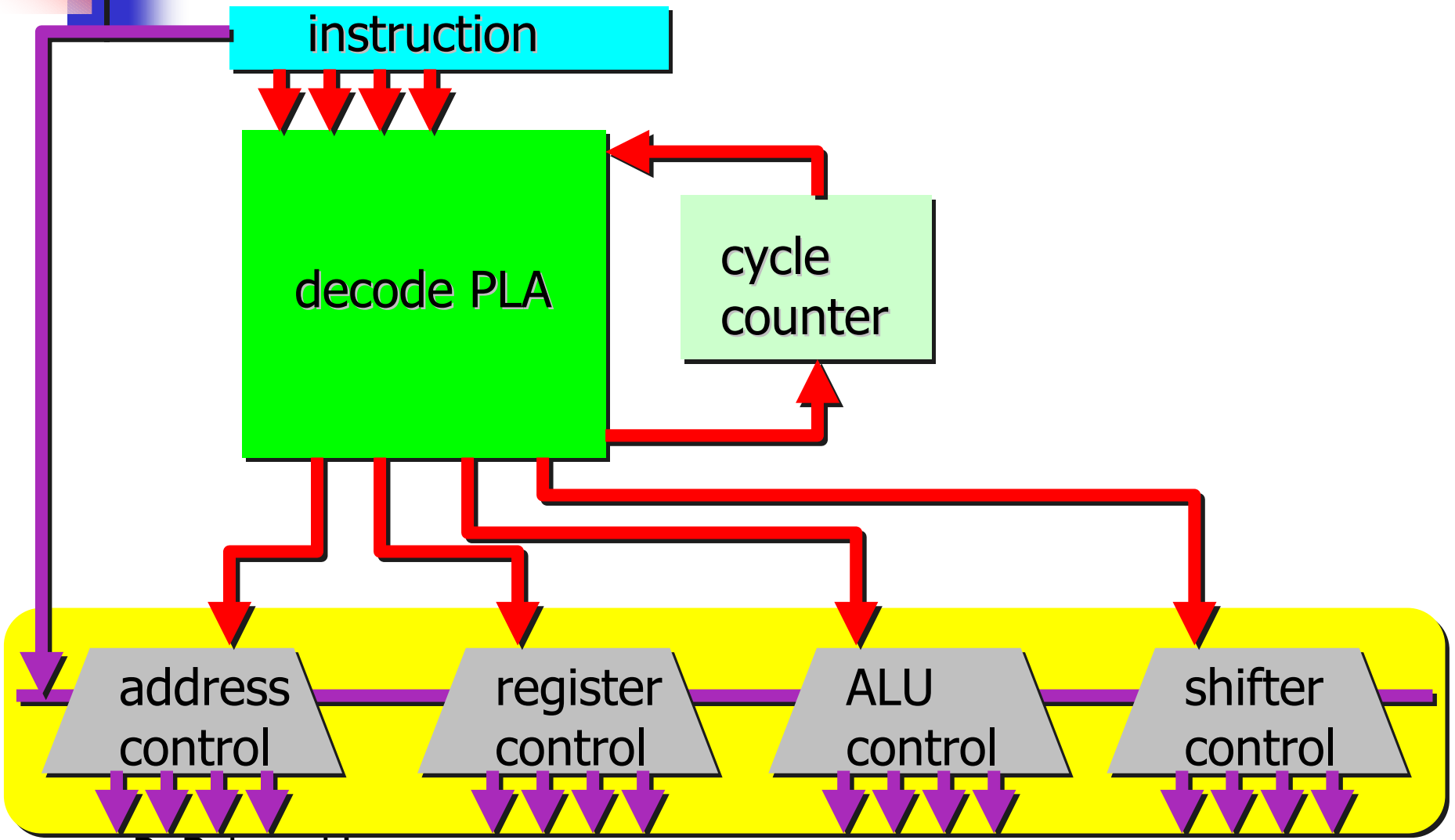
- an instruction decoder PLA
- distributed secondary control associated with main functional units
- decentralized control units for specific instructions

ARM instruction decoder PLA

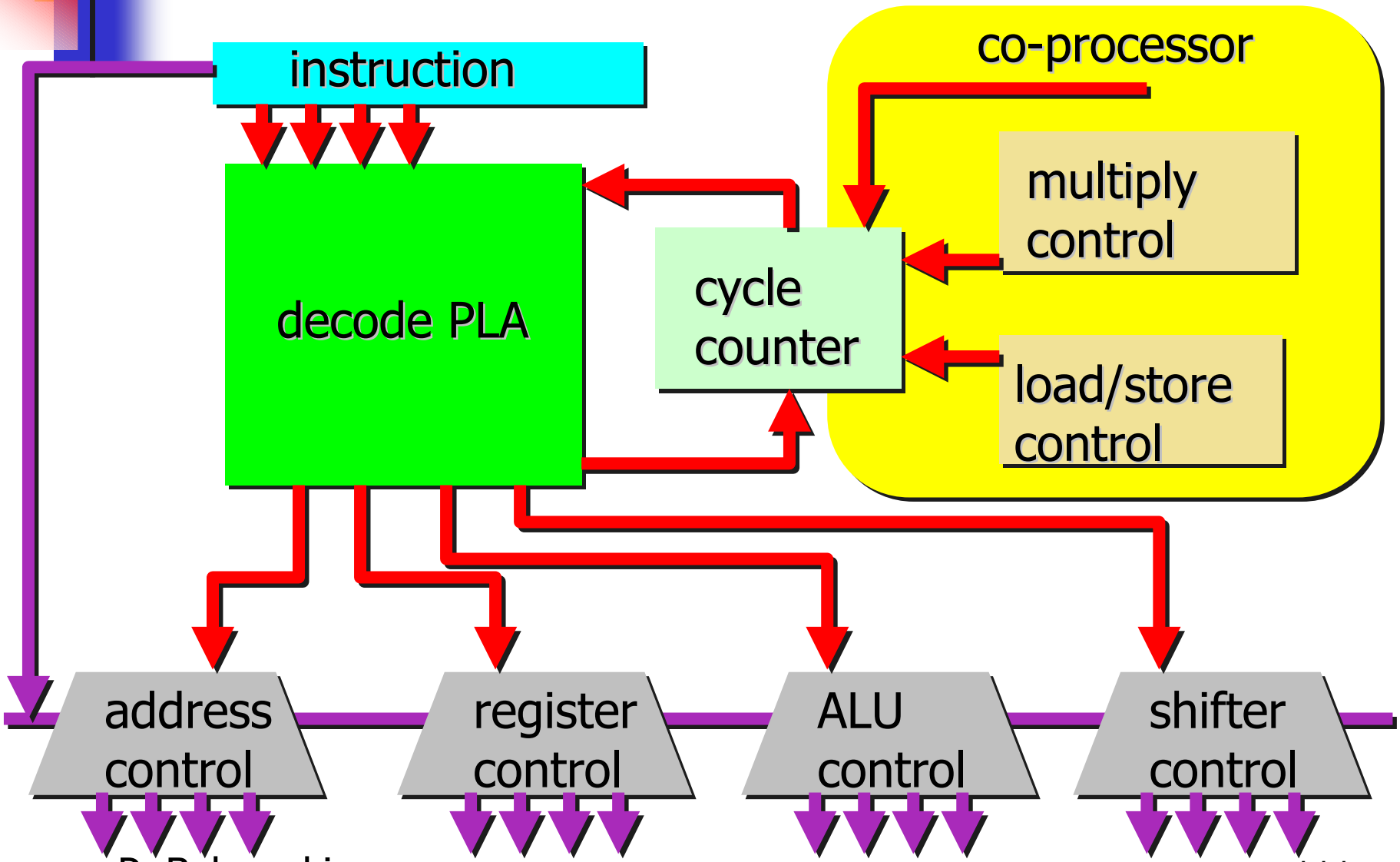


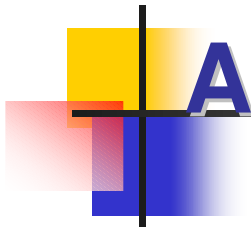
Class of instruction: ALU, load/store, branch, co-processor,..

ARM secondary control



ARM decentralized control

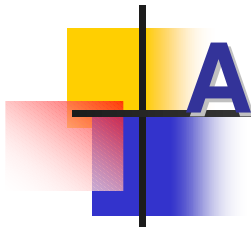




ARM hard cores versus soft cores

There are two kinds of supports for the implementation of ARM cores:

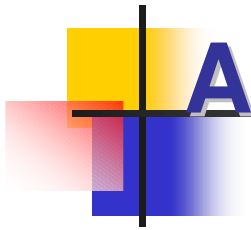
- a hard macrocell that is delivered as physical layout ready to be incorporated into the final design
- a soft macrocell that is delivered as a synthesizable design expressed in a hardware description language such as Verilog HDL or VHDL



ARM hard cores versus soft cores

There are two kinds of supports for the implementation of ARM cores:

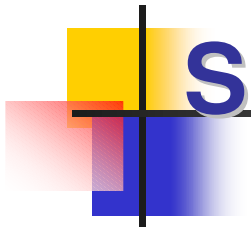
- a hard macrocell that is delivered as physical layout ready to be incorporated into the final design
- a soft macrocell that is delivered as a synthesizable design expressed in a hardware description language such as Verilog HDL or VHDL



ARM hard cores versus soft cores

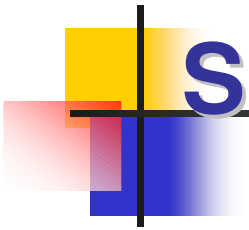
There are two kinds of supports for the implementation of ARM cores:

- a hard macrocell that is delivered as physical layout ready to be incorporated into the final design
- a soft macrocell that is delivered as a synthesizable design expressed in a hardware description language such as Verilog HDL or VHDL



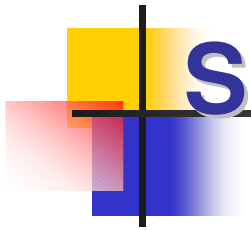
Summary

- ARM clocking scheme
- ARM data path timing
- ARM adder design
- ARM ALU design
- ARM barrel shifter
- ARM multipliers
- ARM register bank
- ARM data path layout
- ARM control path



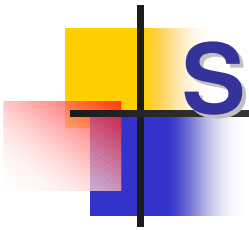
Summary

- ARM clocking scheme
- ARM data path timing
- ARM adder design
- ARM ALU design
- ARM barrel shifter
- ARM multipliers
- ARM register bank
- ARM data path layout
- ARM control path



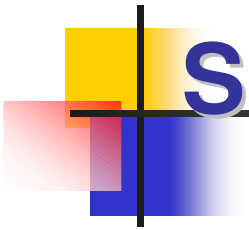
Summary

- ARM clocking scheme
- ARM data path timing
- ARM adder design
- ARM ALU design
- ARM barrel shifter
- ARM multipliers
- ARM register bank
- ARM data path layout
- ARM control path



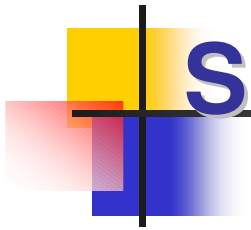
Summary

- ARM clocking scheme
- ARM data path timing
- ARM adder design
- ARM ALU design
- ARM barrel shifter
- ARM multipliers
- ARM register bank
- ARM data path layout
- ARM control path



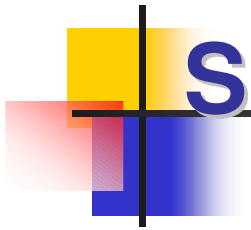
Summary

- ARM clocking scheme
- ARM data path timing
- ARM adder design
- ARM ALU design
- ARM barrel shifter
- ARM multipliers
- ARM register bank
- ARM data path layout
- ARM control path



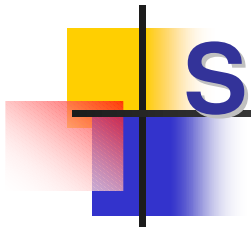
Summary

- ARM clocking scheme
- ARM data path timing
- ARM adder design
- ARM ALU design
- ARM barrel shifter
- ARM multipliers
- ARM register bank
- ARM data path layout
- ARM control path



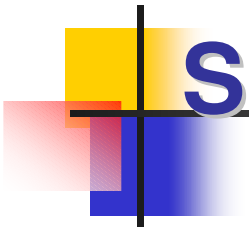
Summary

- ARM clocking scheme
- ARM data path timing
- ARM adder design
- ARM ALU design
- ARM barrel shifter
- ARM multipliers
- ARM register bank
- ARM data path layout
- ARM control path



Summary

- ARM clocking scheme
- ARM data path timing
- ARM adder design
- ARM ALU design
- ARM barrel shifter
- ARM multipliers
- ARM register bank
- ARM data path layout
- ARM control path



Summary

- ARM clocking scheme
- ARM data path timing
- ARM adder design
- ARM ALU design
- ARM barrel shifter
- ARM multipliers
- ARM register bank
- ARM data path layout
- ARM control path