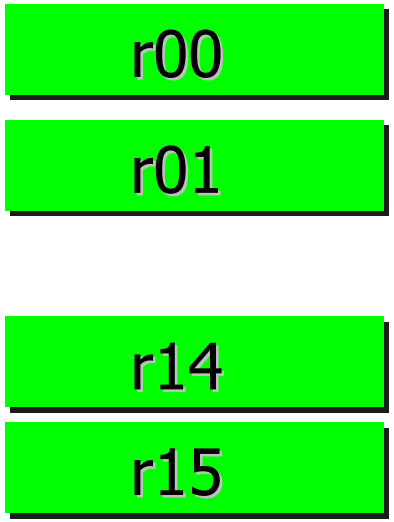
ARM processor organization

P. Bakowski

bako@ieee.org

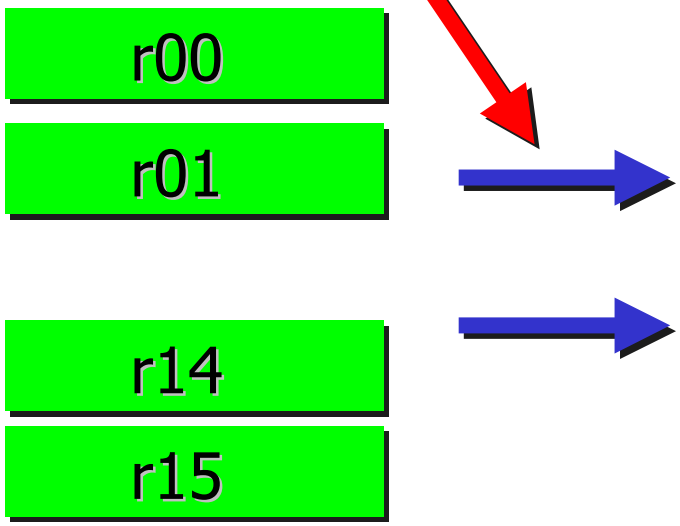
ARM register bank

The register bank, which stores the processor state.



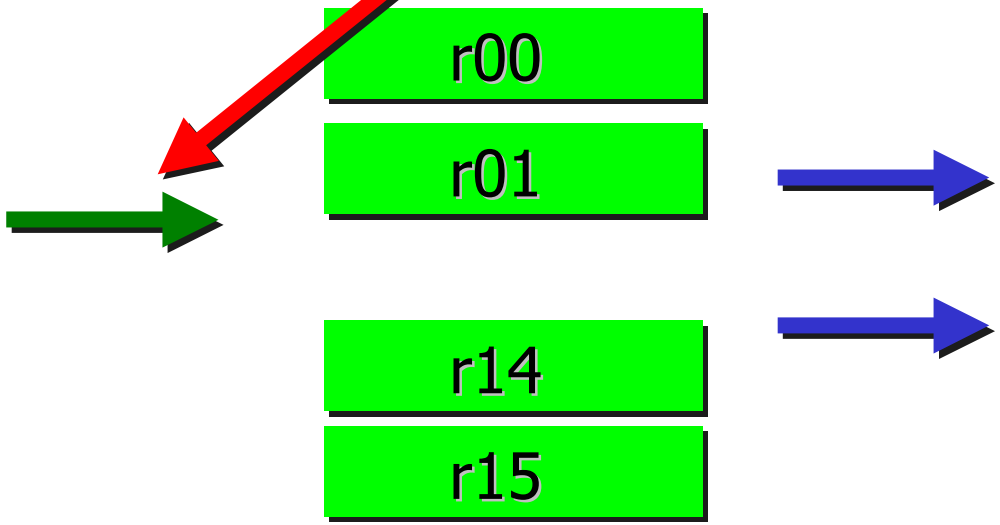
ARM register bank

It has two read ports and one write port which can each be used to access any register, plus an additional read port and an additional write port that give special access to r15, the program counter.



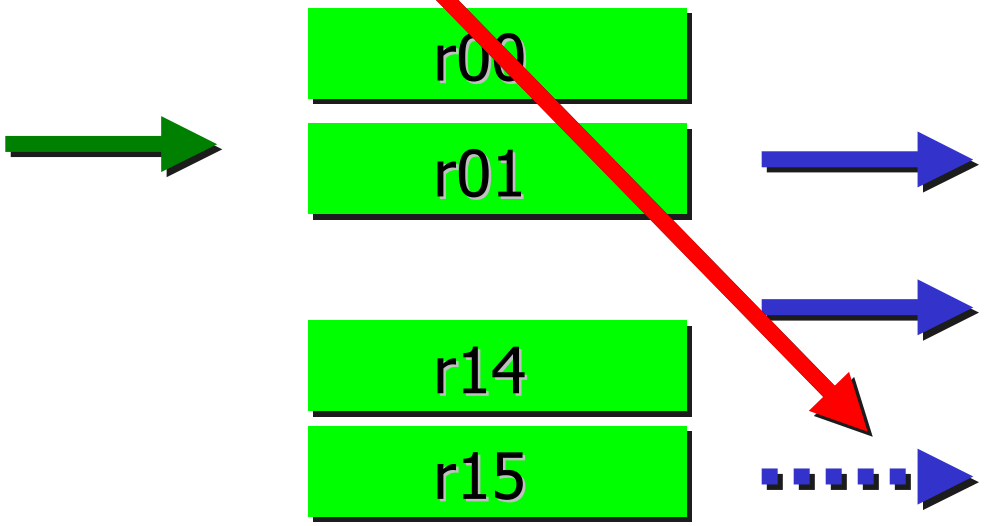
ARM register bank

It has two read ports and one write port which can each be used to access any register, plus an additional read port and an additional write port that give special access to r15, the program counter.



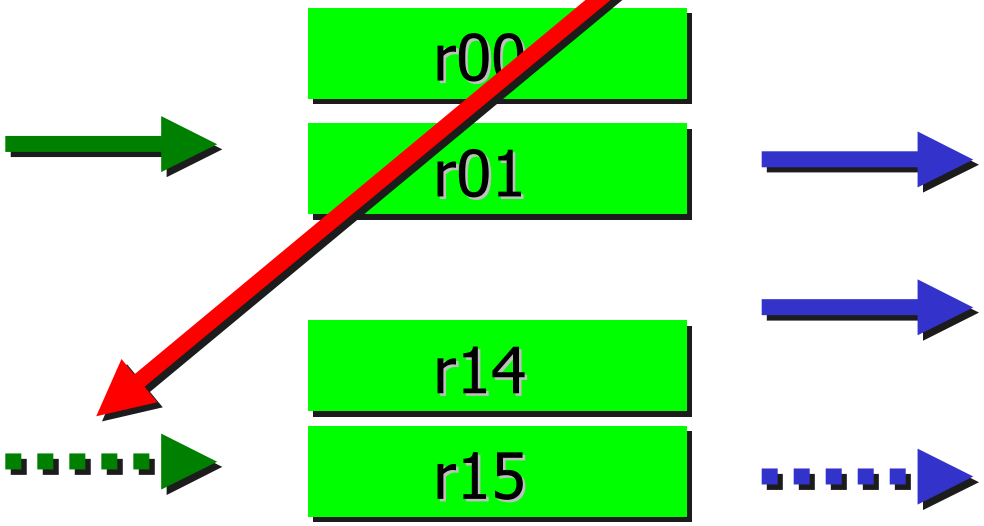
ARM register bank

It has two read ports and one write port which can each be used to access any register, plus an additional read port and an additional write port that give special access to r15, the program counter.



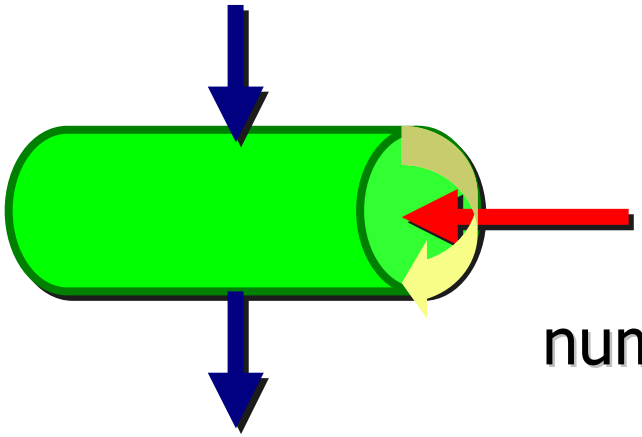
ARM register bank

It has two read ports and one write port which can each be used to access any register, plus an additional read port and an additional write port that give special access to r15, the program counter.



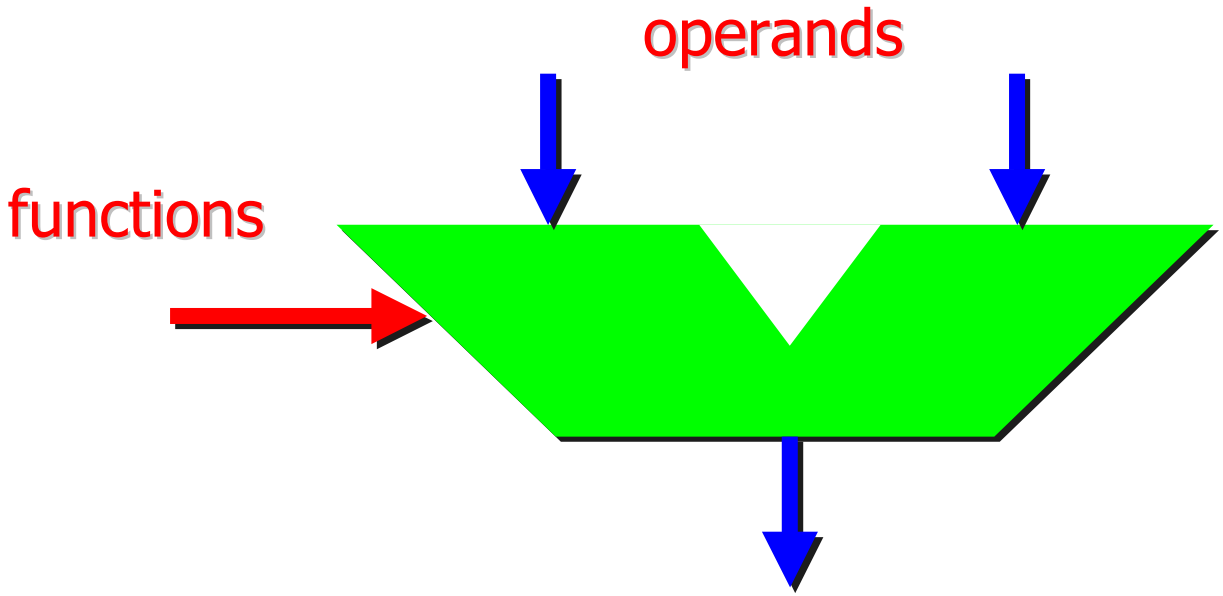
ARM barrel shifter

The barrel shifter, which can shift or rotate *one* operand by any number of bits.



ARM ALU

The ALU, which performs the arithmetic and logic functions required by the instruction set.

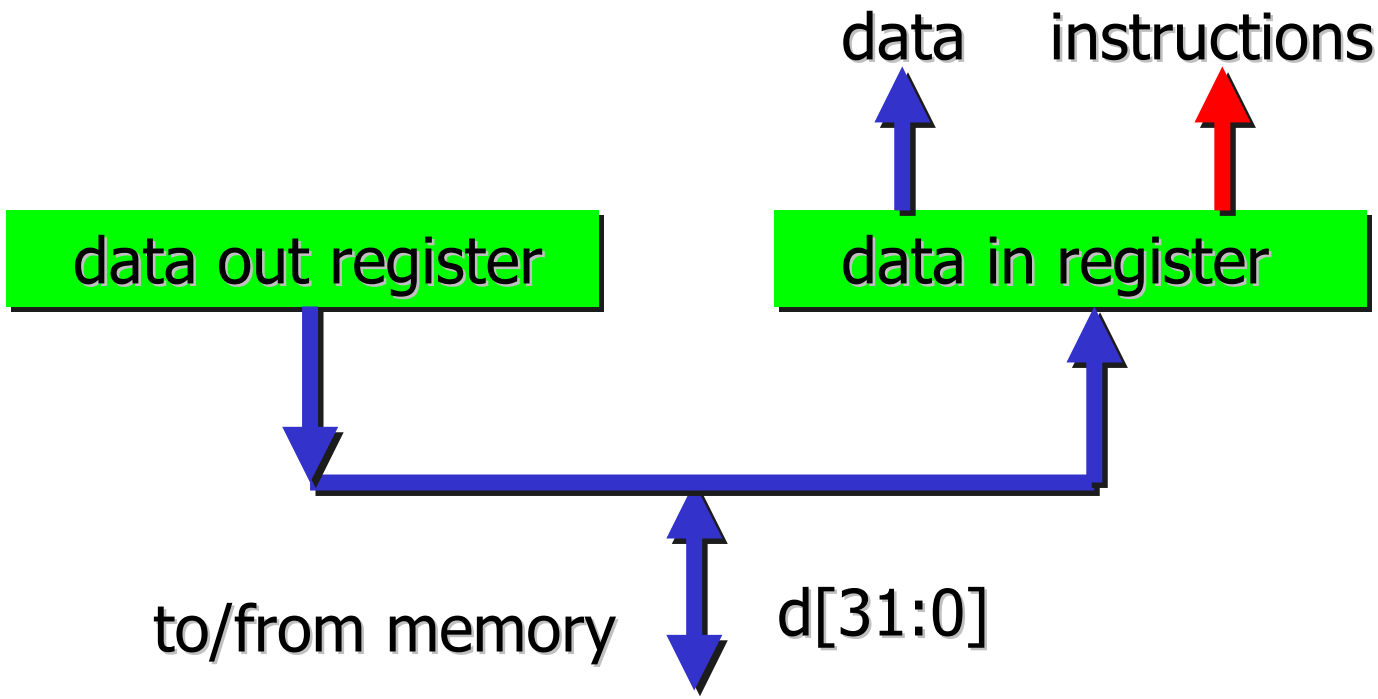


ARM 3-stage pipeline

The address register and incrementer, which select and hold addresses and generate sequential addresses when required.

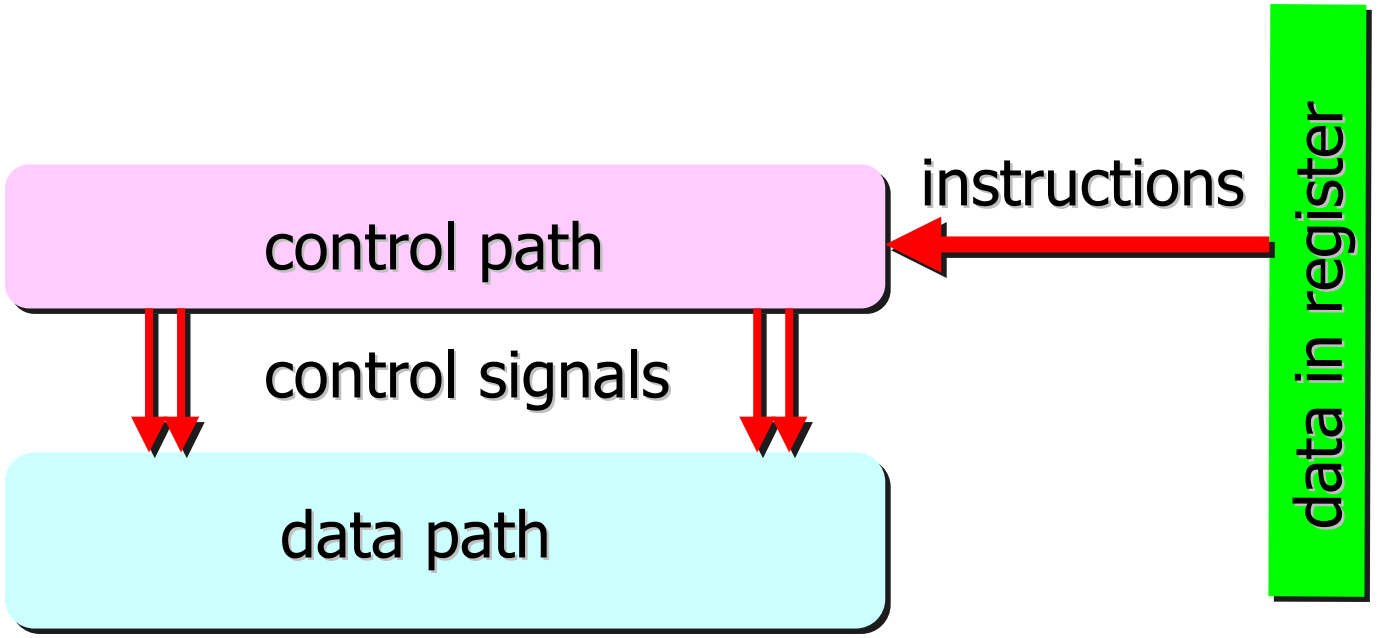
ARM 3-stage pipeline

The data registers, which hold data passing to and from memory.



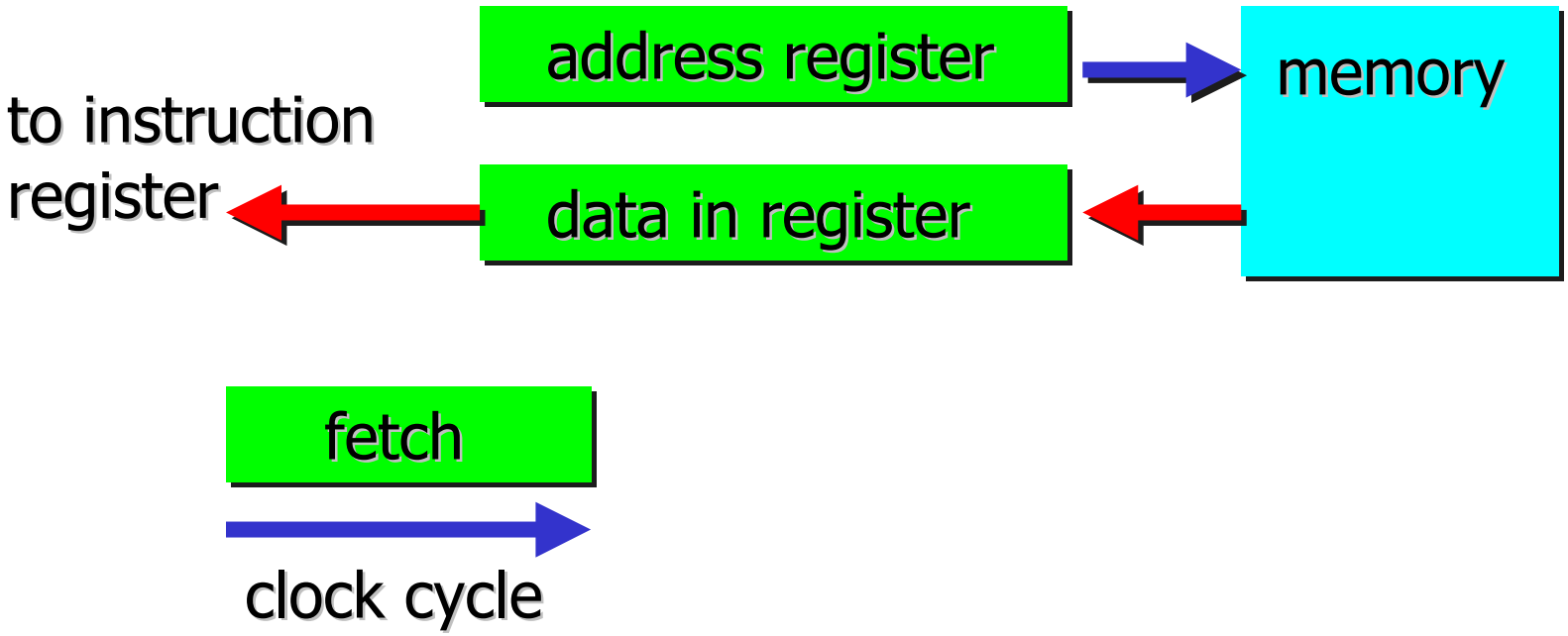
ARM 3-stage pipeline

The instruction decoder and associated control logic.



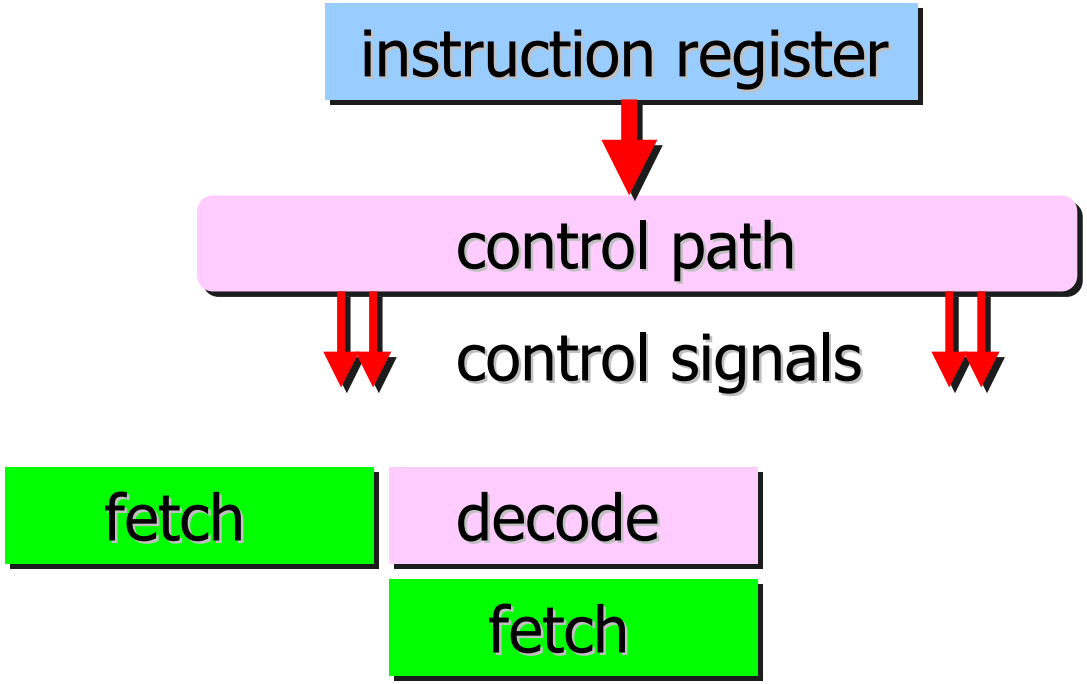
Three stage pipeline : ARM 1,2,3

FETCH; the instruction is fetched from memory and placed in the instruction pipeline.



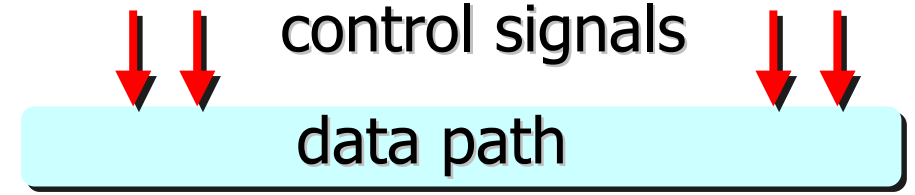
Three stage pipeline : ARM 1,2,3

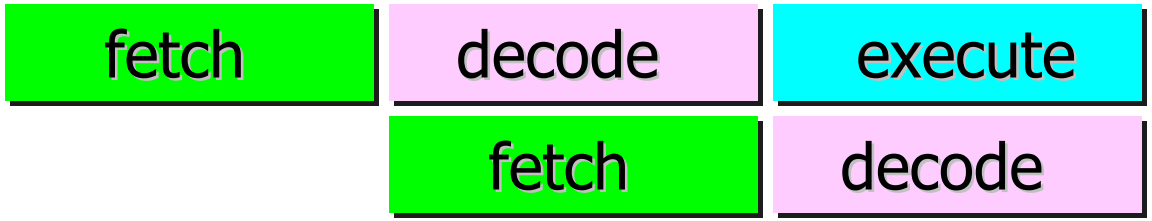
DECODE; the instruction is decoded and the datapath control signals prepared for the next cycle.



Three stage pipeline : ARM 1,2,3

EXECUTE; the instruction controls the datapath; the register bank is read, an operand shifted the ALU result generated and written back into a destination register.

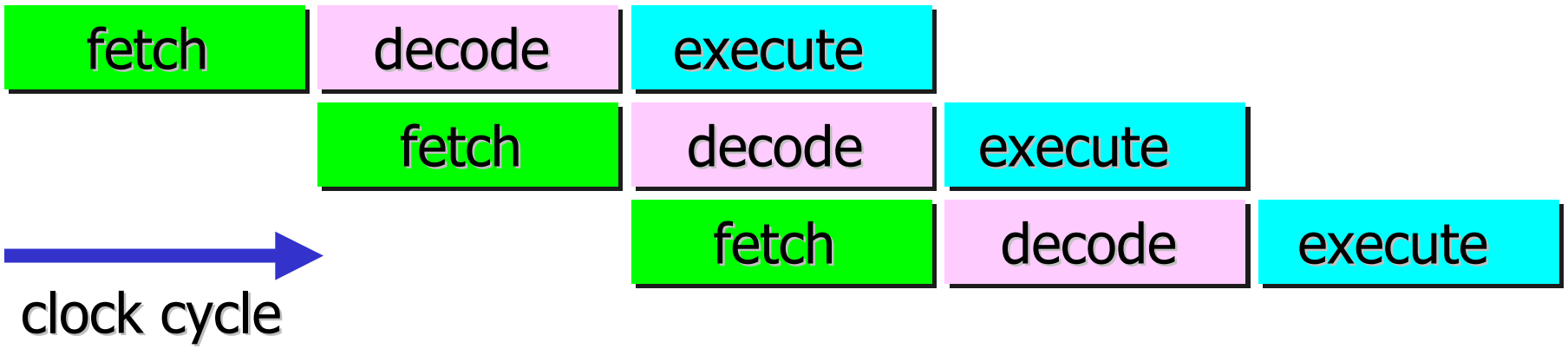




Three stage pipeline : ARM 1,2,3

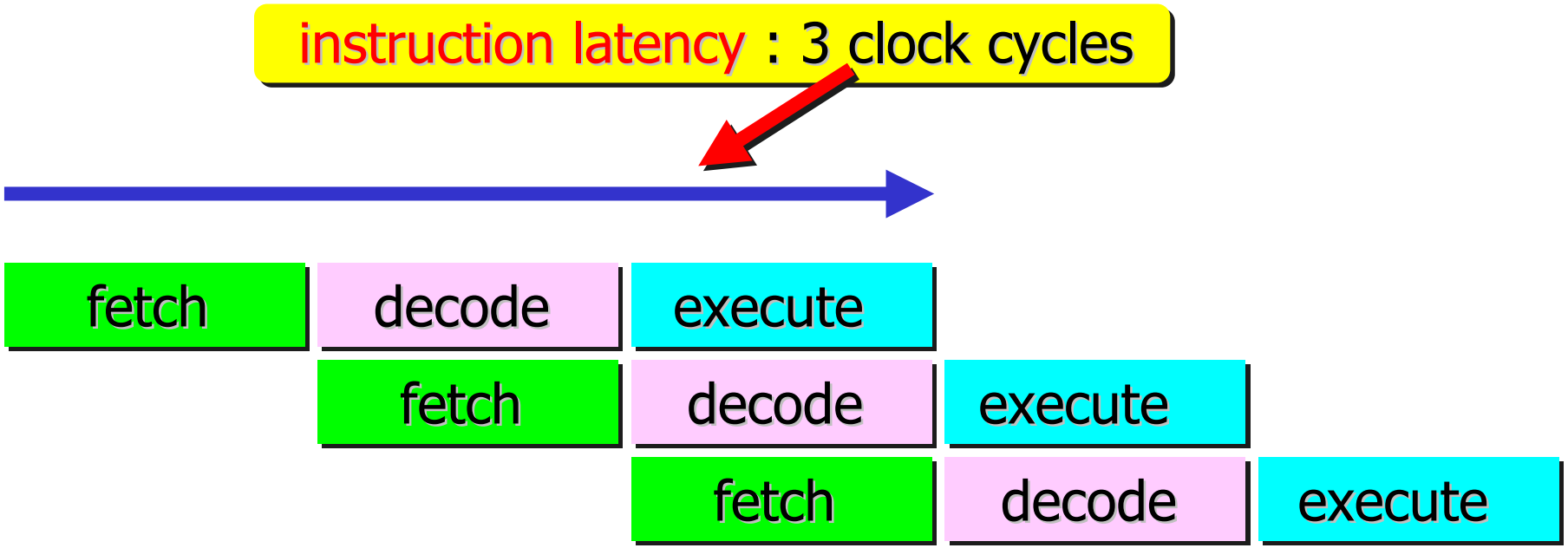
instruction throughput : 1 instruction per clock cycle

instruction latency : 3 clock cycles

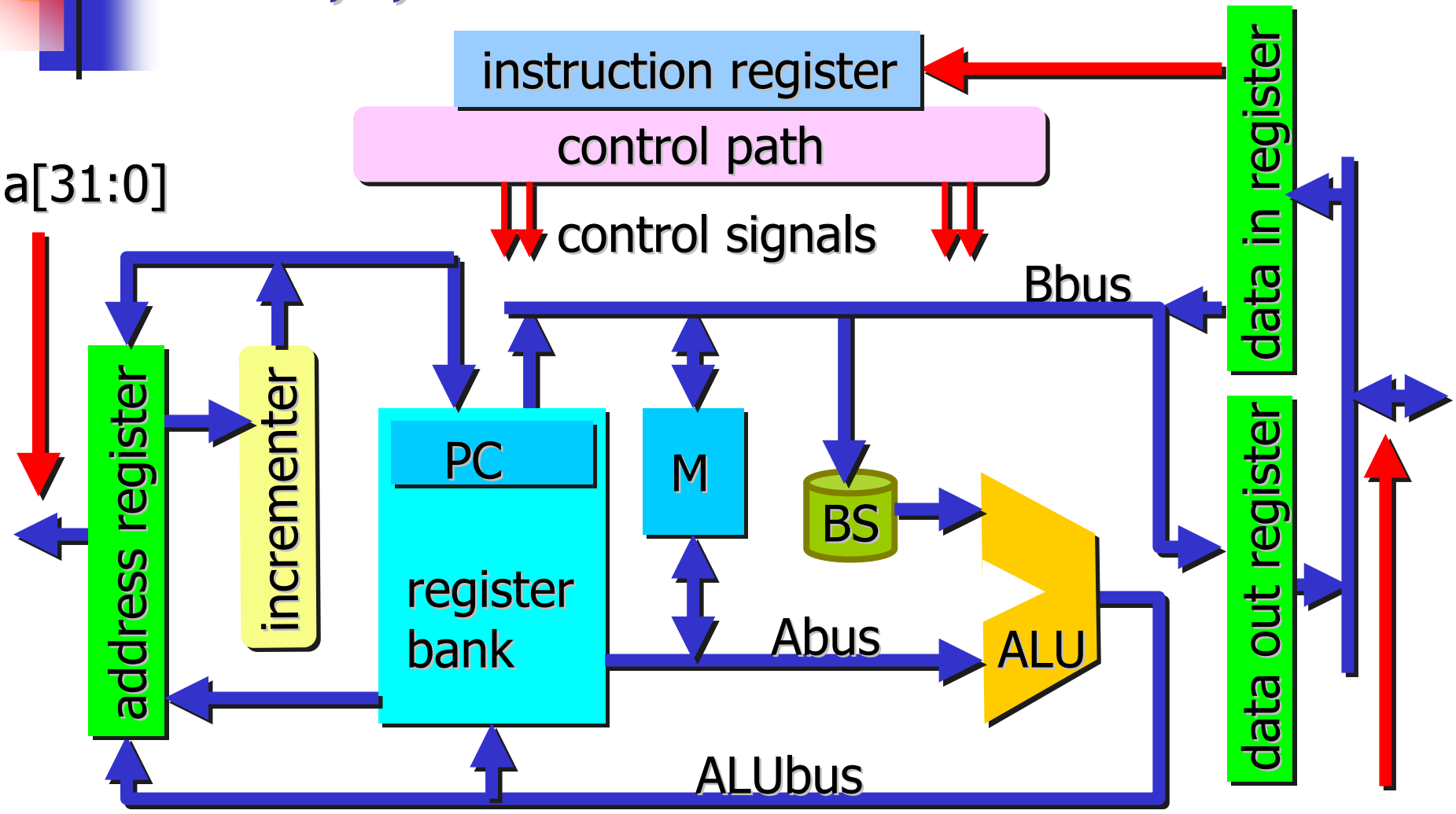


Three stage pipeline : ARM 1,2,3

instruction throughput : 1 instruction per clock cycle

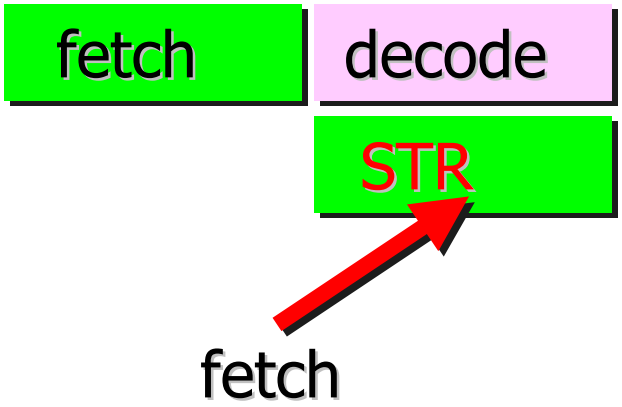


ARM 1,2,3 architecture



M – multiplier, BS – barrel shifter

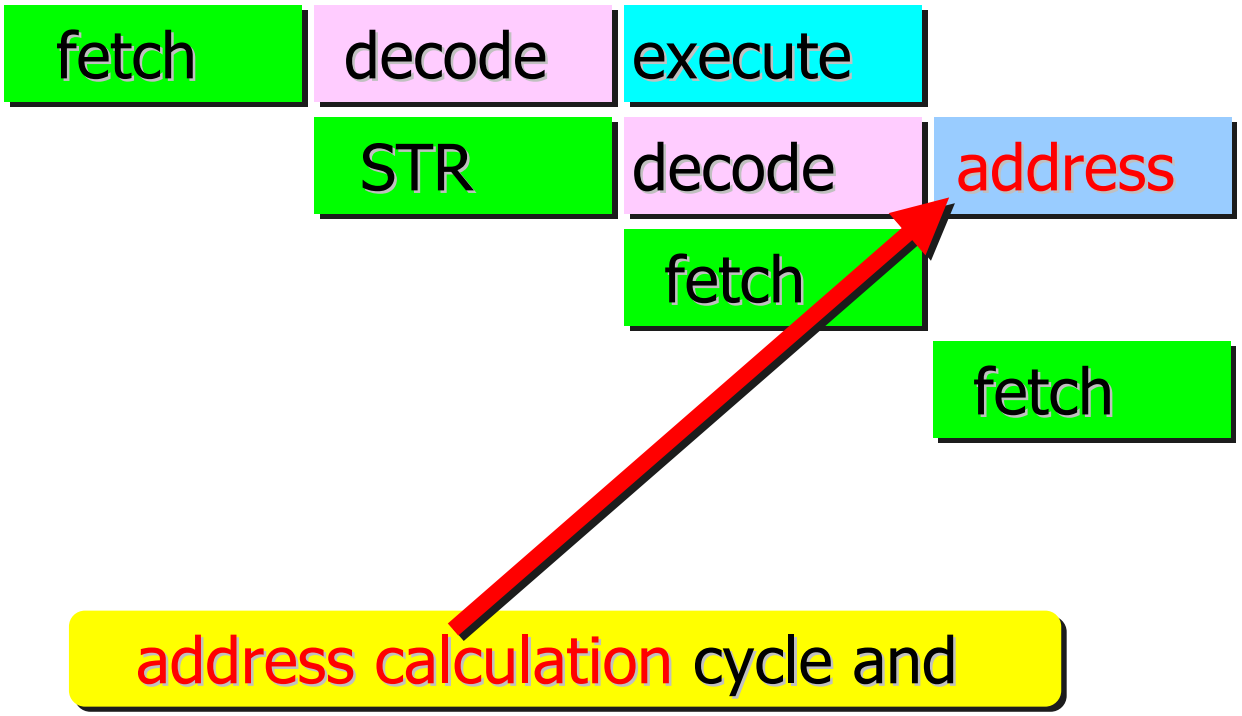
Multi cycle execution



STR – store instruction needs two execution cycles:

- address calculation cycle and
- data transfer cycle

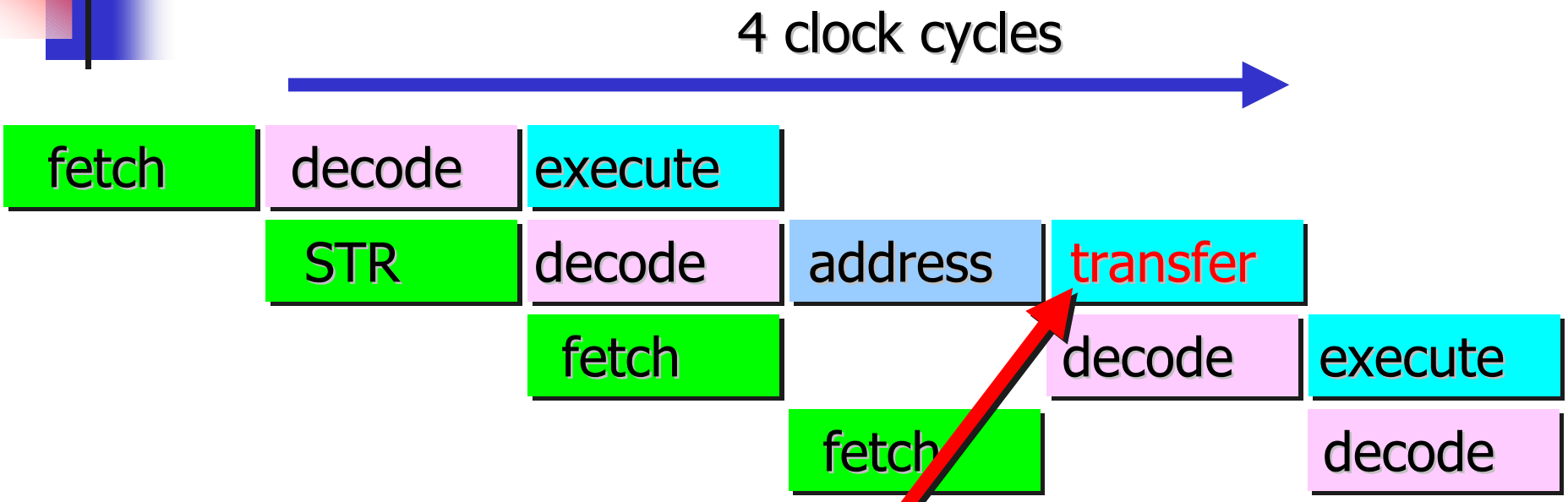
Multi cycle execution



address calculation cycle and

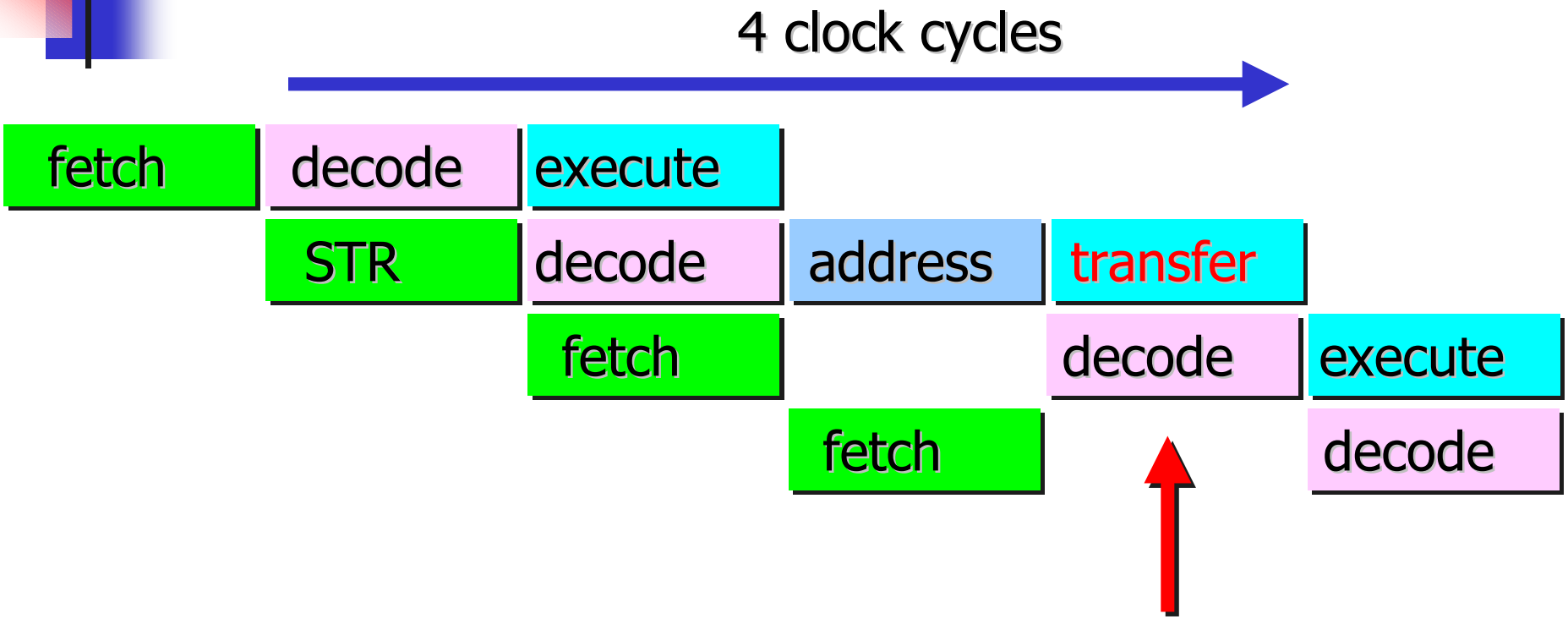
- data transfer cycle

Multi cycle execution



- address calculation cycle and
- data transfer cycle

Multi cycle execution



- address calculation cycle and
- data transfer cycle

Attention: only one memory transfer per clock cycle

Processor performance

The time T, required to execute a given program is given by:

$$T=(N_{inst}*CPI)/f_{clk}$$

N_{inst} - *number of instructions in the program*

CPI - clock cycles per instruction

f_{clk} - *clock frequency*

Processor performance

The time T, required to execute a given program is given by:

$$T=(N_{inst}*CPI)/f_{clk}$$

N_{inst} - *number of instructions in the program*

CPI - clock cycles per instruction

f_{clk} - *clock frequency*

Processor performance

The time T, required to execute a given program is given by:

$$T=(N_{inst}*CPI)/f_{clk}$$

N_{inst} - *number of instructions in the program*

CPI - clock cycles per instruction (throughput)

f_{clk} - *clock frequency*

Processor performance

The time T, required to execute a given program is given by:

$$T=(N_{inst}*CPI)/f_{clk}$$

N_{inst} - *number of instructions in the program*

CPI - clock cycles per instruction

f_{clk} - *clock frequency*

Processor performance

Since N_{insi} is constant for a given program there are only two ways to increase performance:

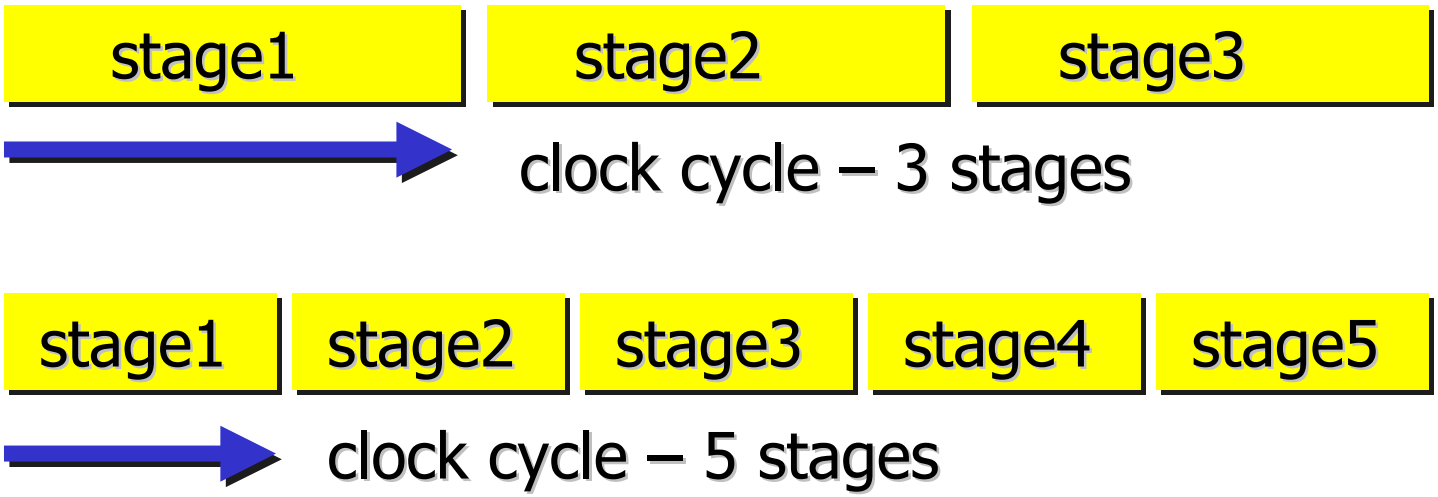
- increase the clock rate, f_{clk}.
- reduce the average number of clock cycles per instruction, *CPI.*

Clock rate increase

Increase the clock rate, f_{clk}. This requires the logic in each pipeline stage to be simplified and, therefore, the number of pipeline stages to be increased.

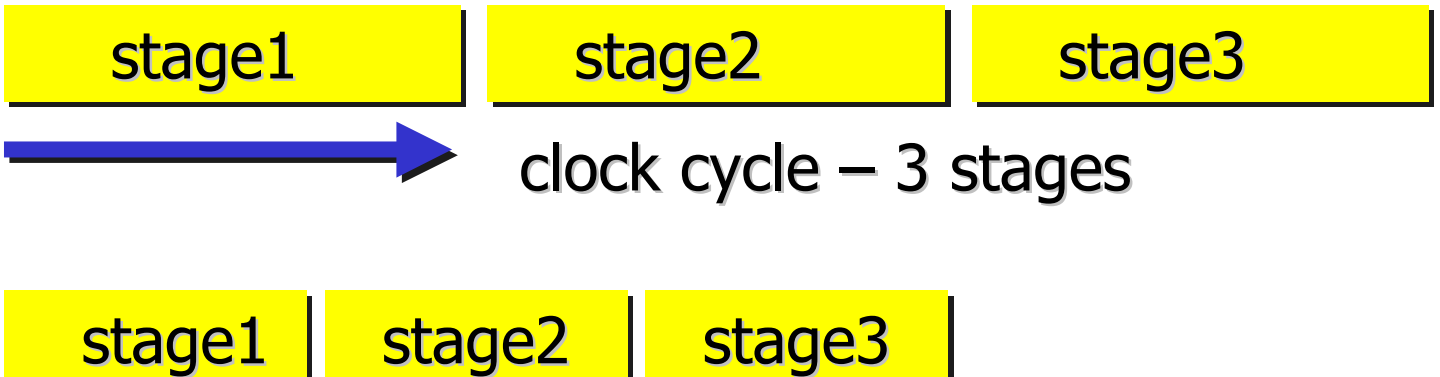
Clock rate increase

Increase the clock rate, f_{clk}. This requires the logic in each pipeline stage to be simplified and, therefore, the number of pipeline stages to be increased.



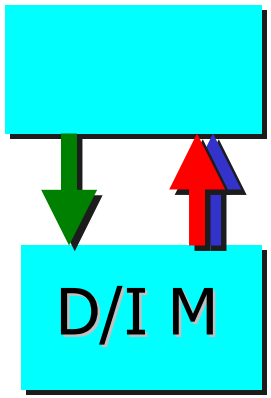
Clock rate increase

Note that the clock rate to be used depends heavily on the implementation technology.

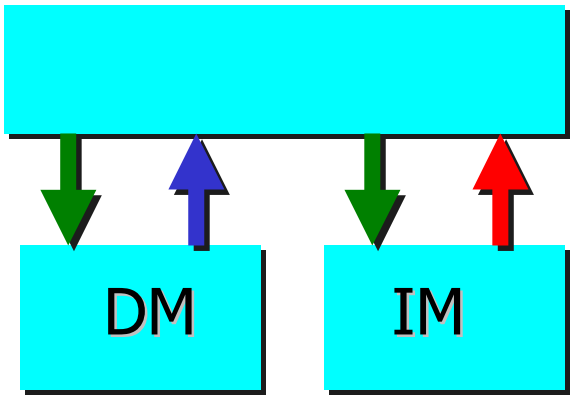


More hardware resources

Reduce the average number of clock cycles per instruction, *CPI.* This requires the introduction of more parallelism that means more hardware resources to be used in a given clock cycle.



data read or
instruction fetch



data read and
instruction fetch

5-stage pipeline organization

Higher performance ARM cores employ a 5-stage pipeline and have separate instruction and data memories.

Breaking instruction execution down into five stages rather than three reduces the maximum work which must be completed in a clock cycle, and hence allows a higher clock frequency to be used.

The separate instruction and data memories seen as separate caches connected to a unified instruction and data main memory allow a significant reduction in the core's CPI.

5-stage pipeline organization

Higher performance ARM cores employ a 5-stage pipeline and have separate instruction and data memories.

Breaking instruction execution down into five stages rather than three reduces the maximum work which must be completed in a clock cycle, and hence allows a higher clock frequency to be used.

The separate instruction and data memories seen as separate caches connected to a unified instruction and data main memory allow a significant reduction in the core's CPI.

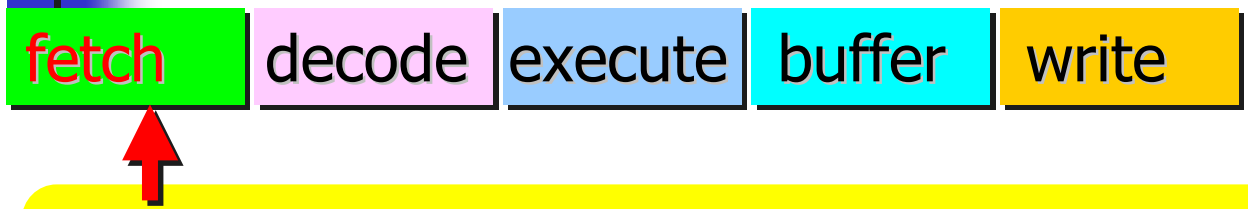
5-stage pipeline organization

Higher performance ARM cores employ a 5-stage pipeline and have separate instruction and data memories.

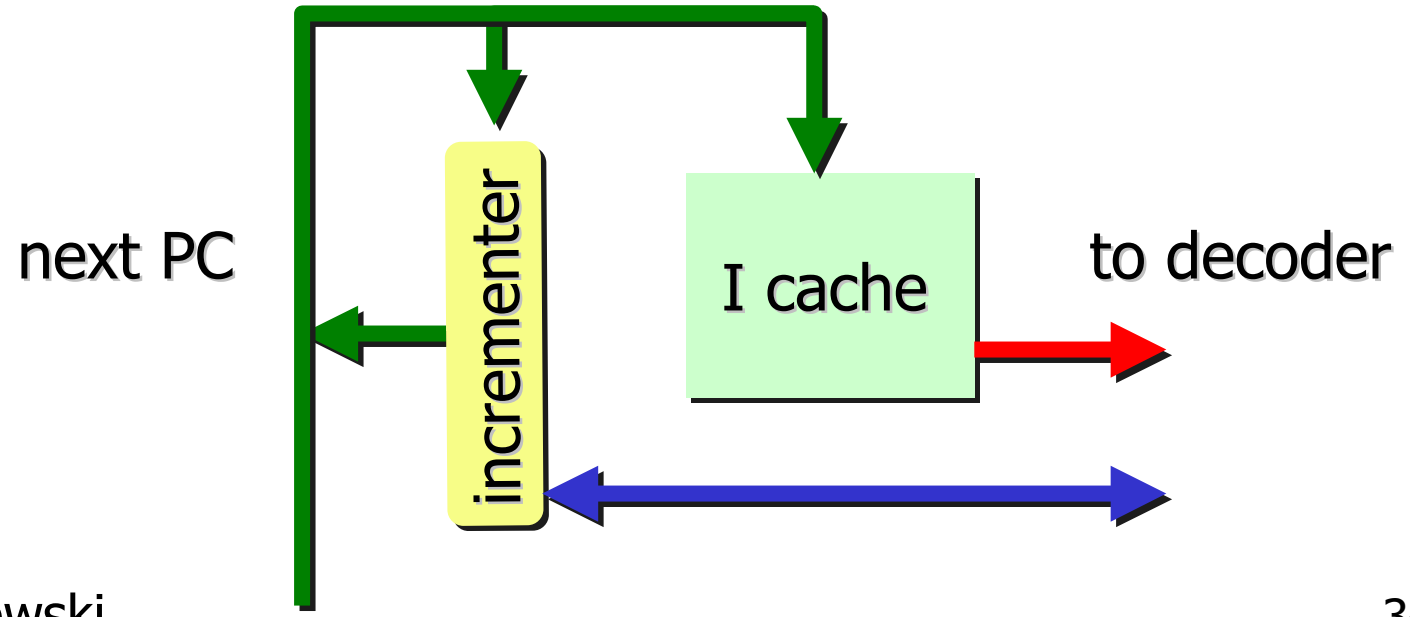
Breaking instruction execution down into five stages rather than three reduces the maximum work which must be completed in a clock cycle, and hence allows a higher clock frequency to be used.

The separate instruction and data memories seen as separate caches connected to a unified instruction and data main memory allow a significant reduction in the core's CPI.

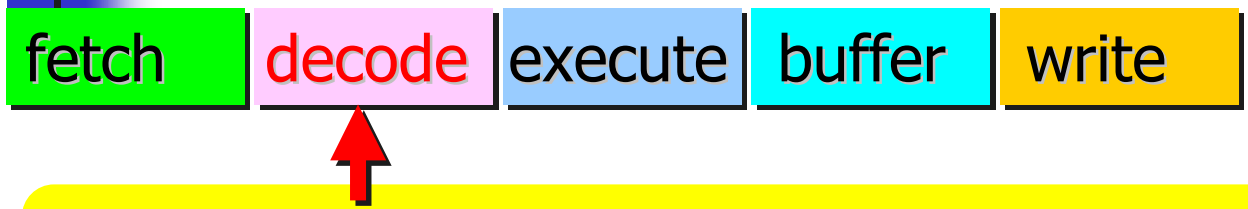
Fetch stage



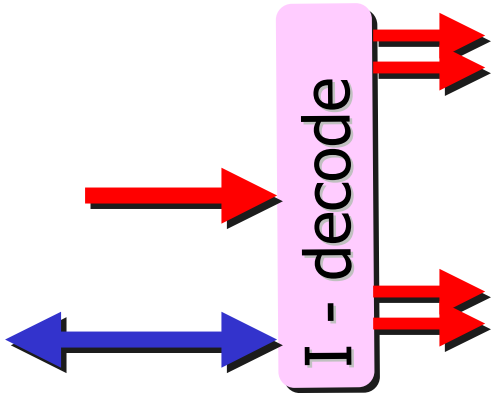
FETCH - the instruction is fetched from memory and placed in the instruction cache.



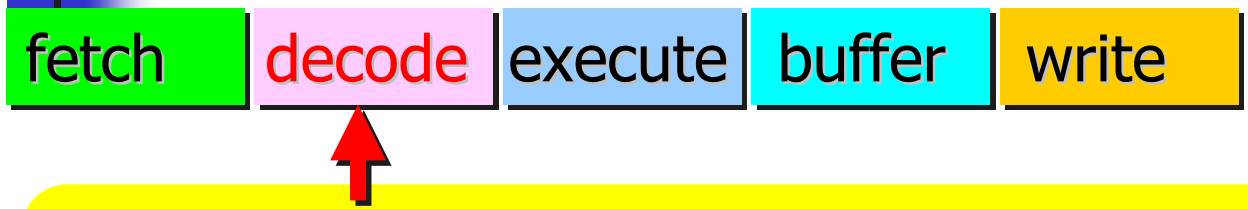
Decode stage



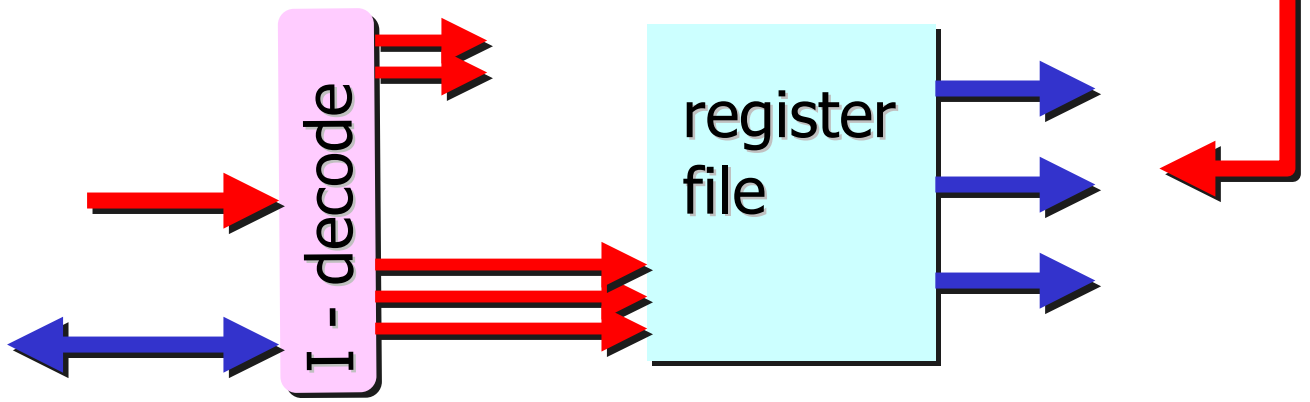
DECODE - the instruction is decoded and register operands read from the register file.



Decode stage



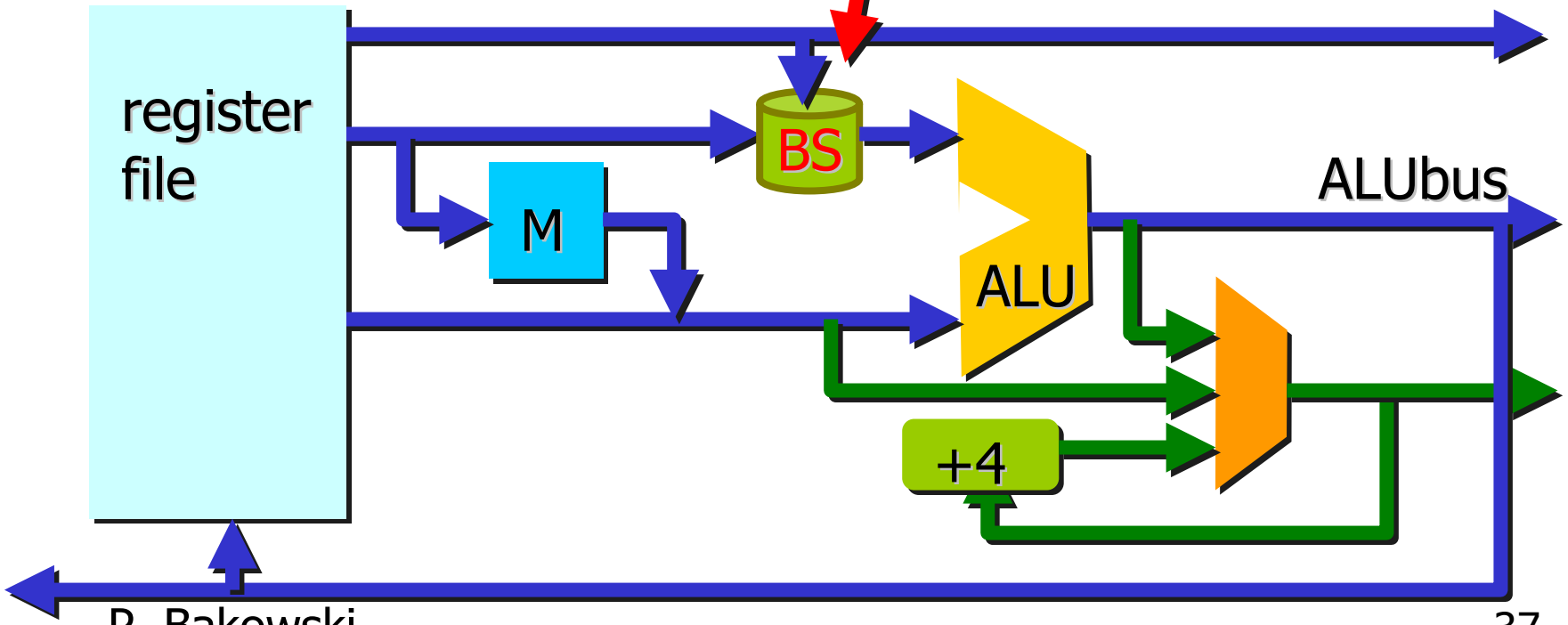
There are three operand read ports in the register file, so most instructions can obtain all their operands in one cycle.



Execute stage

fetch | decode | execute | buffer | write

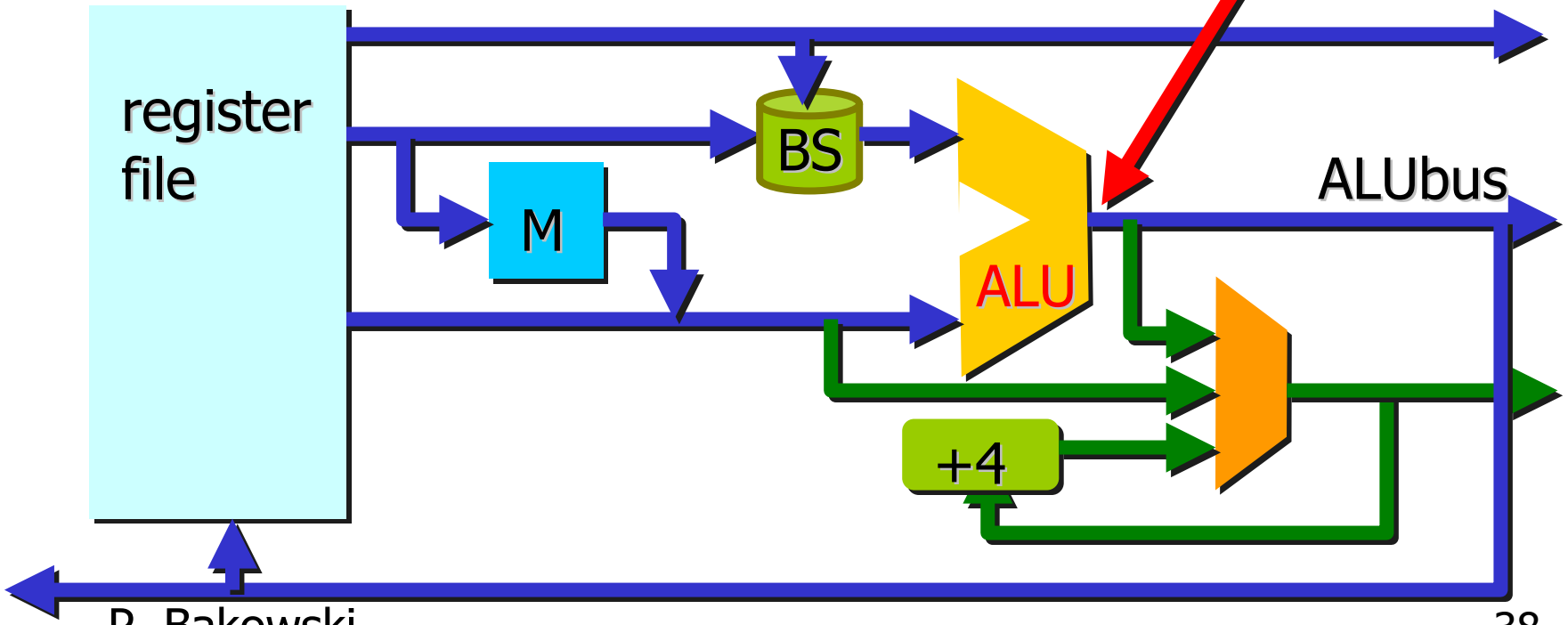
EXECUTE - an operand is shifted and the ALU result generated.



Execute stage

fetch | decode | execute | buffer | write

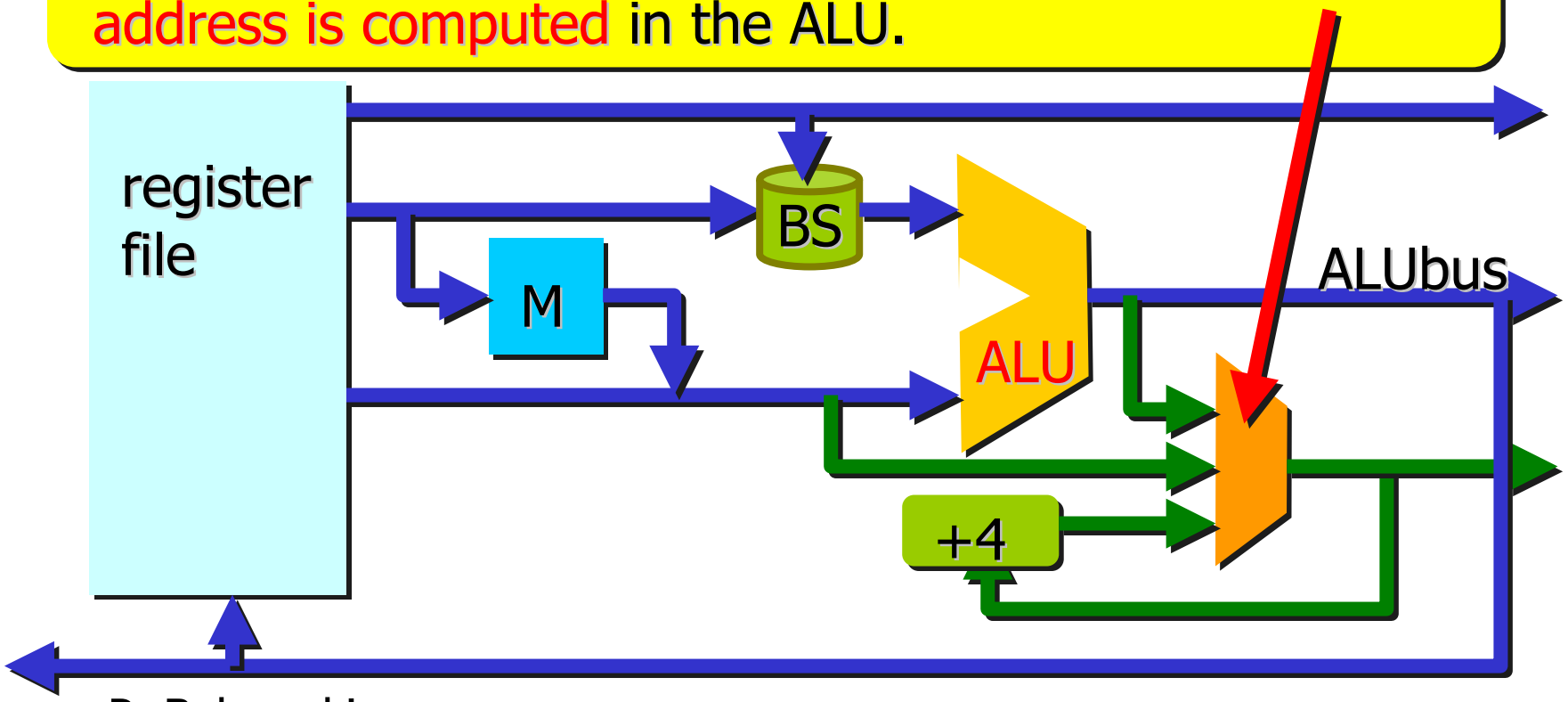
EXECUTE - an operand is shifted and the ALU result generated.



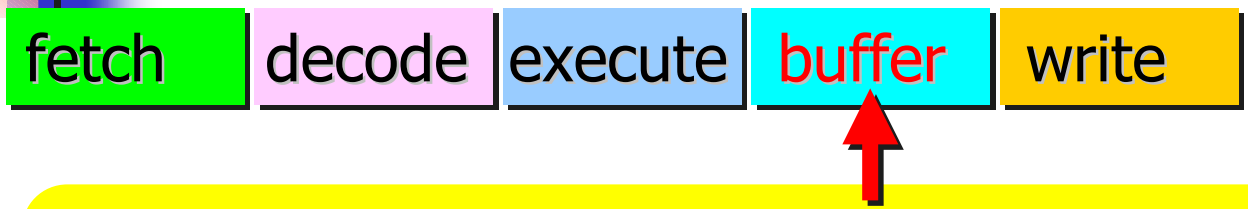
Execute stage

fetch | decode | execute | buffer | write

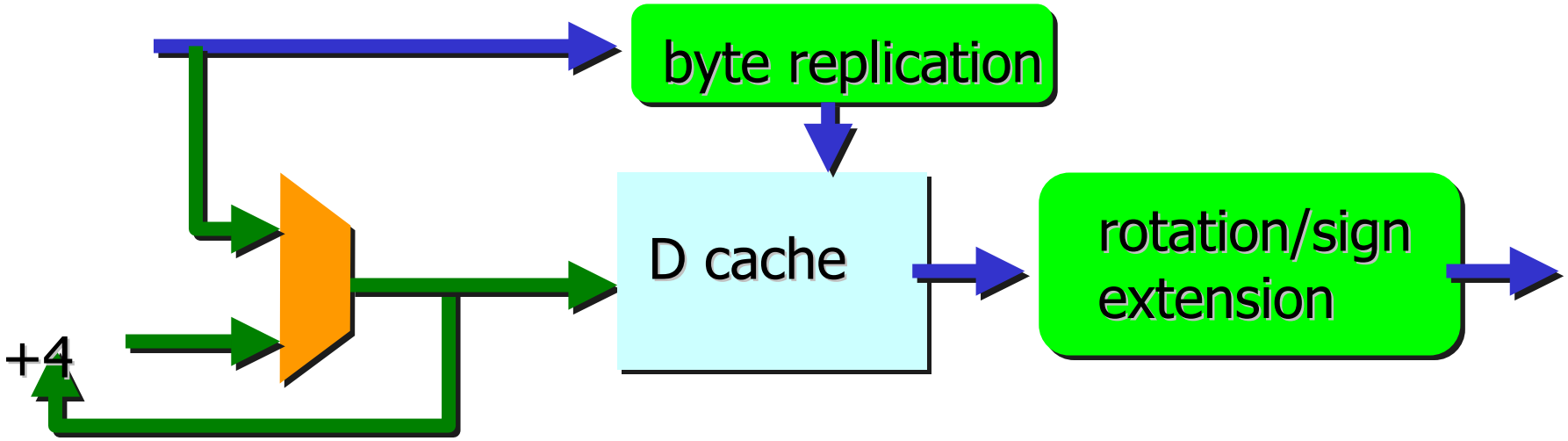
If the instruction is a load or store the memory address is computed in the ALU.



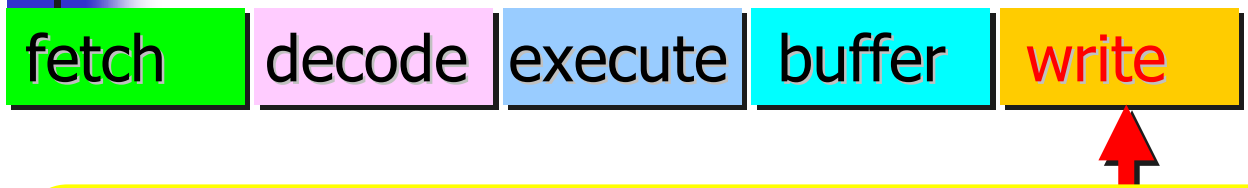
Buffer stage



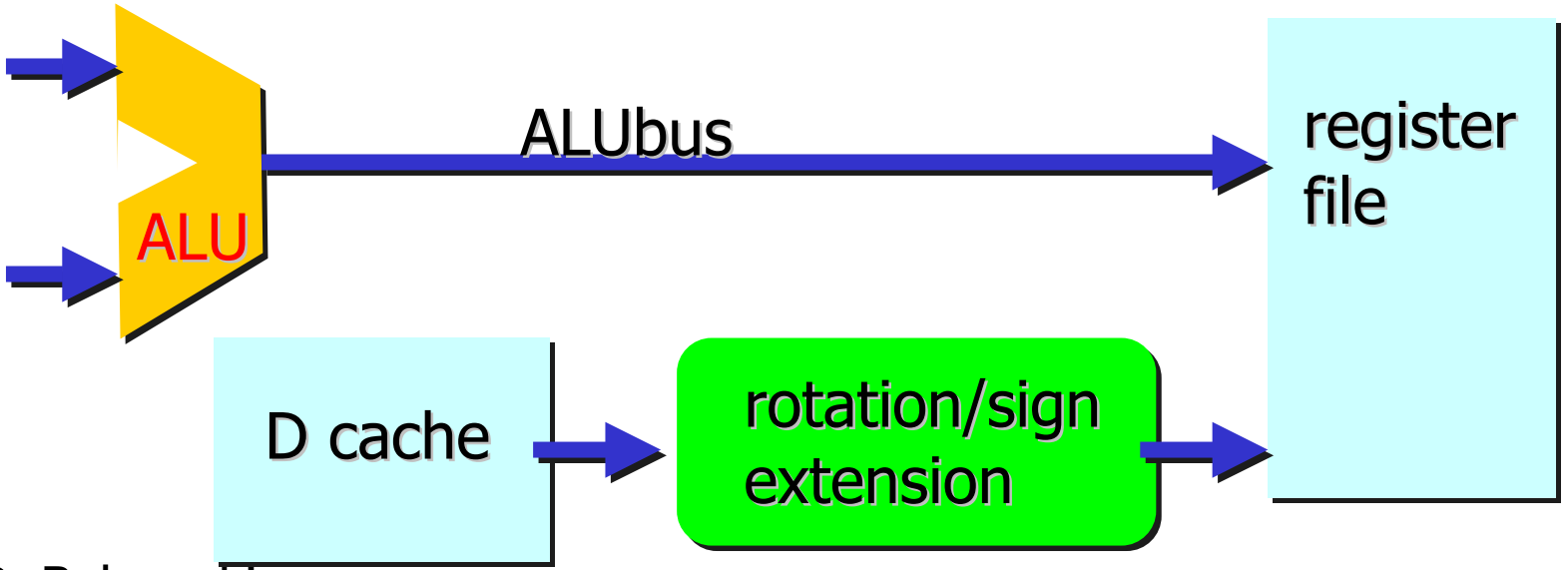
BUFFER data - data memory is accessed if required. Otherwise the ALU result is simply buffered for one clock cycle to give the same pipeline flow for all instructions.



Write back stage



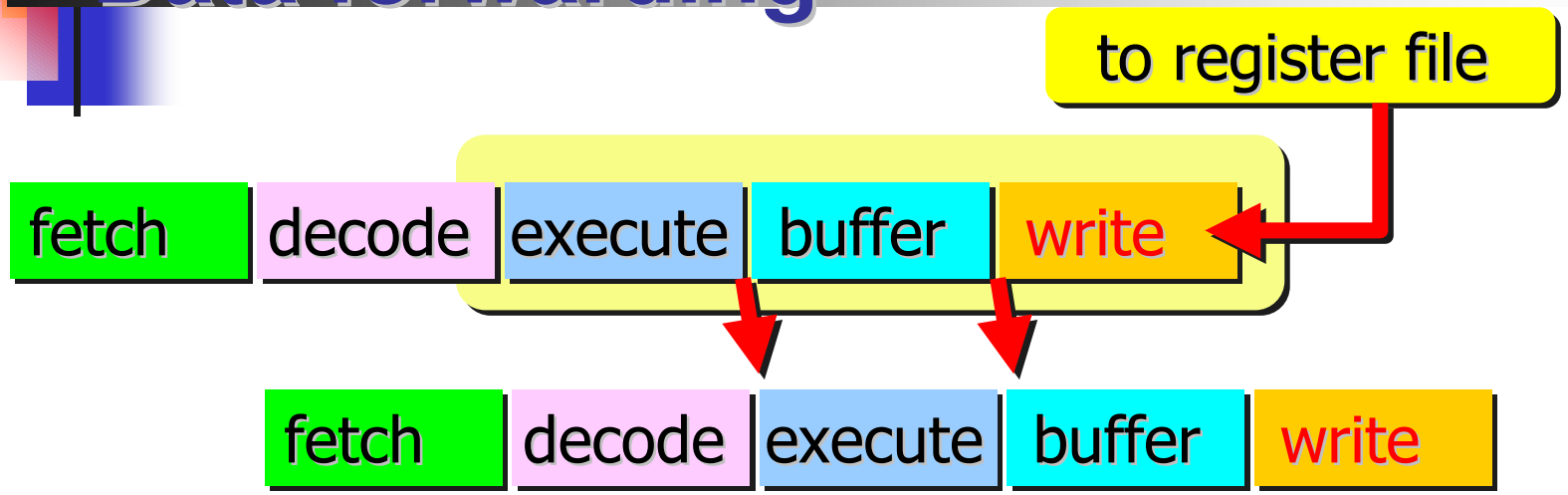
WRITE-back; the results generated by the instruction are written back to the register file, including any data loaded from memory.



Data forwarding

In the 5-stage pipeline instruction execution is spread across three pipeline stages, the only way to resolve data dependencies without stalling the pipeline is to introduce *forwarding* paths.

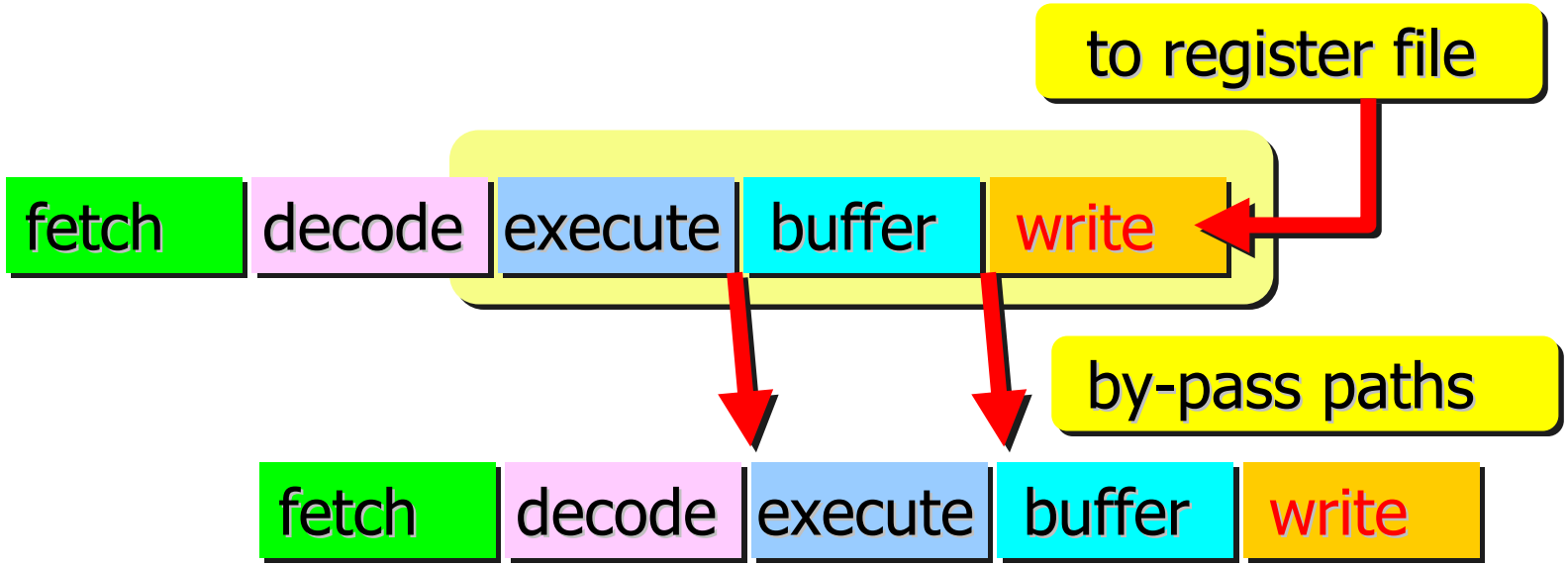
Data forwarding



Data dependencies arise when an instruction needs to use the result of one of its predecessors before that result has returned to the register file.

Data forwarding

Forwarding paths (by-pass) allow the intermediate results to be passed between stages as soon as they are available, in the 5-stage ARM pipeline each of the three source operands can be forwarded from any of three intermediate result registers



PC organization - compatibility

The programming behavior of the PC implemented through r15 is based on the operational characteristics of the 3-stage ARM pipeline.

Basically the 5-stage pipeline reads the instruction operands one stage earlier and that is incompatible with 3-stage design.

PC organization - compatibility

The programming behavior of the PC implemented through r15 is based on the operational characteristics of the 3-stage ARM pipeline.

Basically the 5-stage pipeline reads the instruction operands one stage earlier and that is incompatible with 3-stage design.

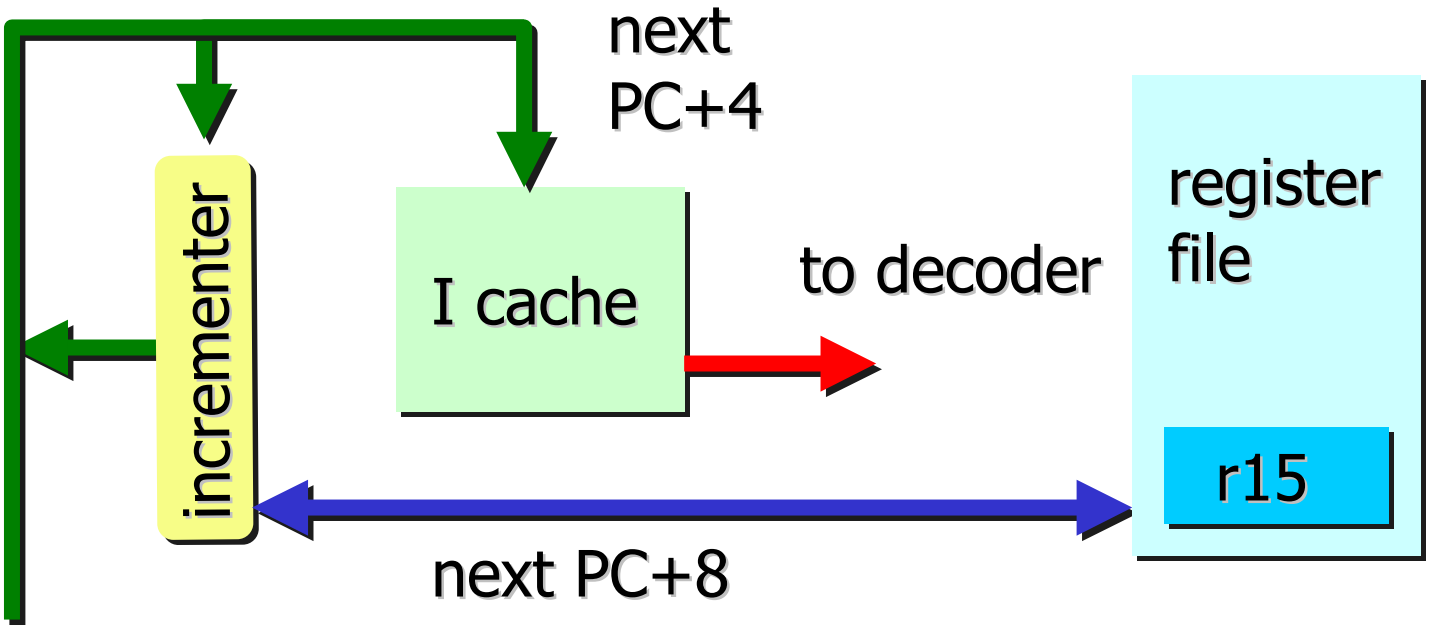
PC organisation - solution

This problem is resolved by the incrementation of the PC value from the fetch stage in the decode stage, bypassing the pipeline register between the two stages.

PC+4 for the next instruction is equal to PC+8 for the current instruction (4 bytes farther), so the correct r15 value is obtained without additional hardware.

PC organisation - solution

PC+4 for the next instruction is equal to PC+8 for the current instruction (4 bytes farther), so the correct r15 value is obtained without additional hardware.



ARM programming model

The Instruction Set Architecture (ISA) defines the operations that the programmer can use to change the state of the system incorporating the processor.

This state usually comprises the values of the data items in the visible registers and the memory.

Each instruction performs a defined transformation from the state before the instruction is executed to the state after it has completed.

ARM programming model

The Instruction Set Architecture (ISA) defines the operations that the programmer can use to change the state of the system incorporating the processor.

This state usually comprises the values of the data items in the visible registers and the memory.

Each instruction performs a defined transformation from the state before the instruction is executed to the state after it has completed.

ARM programming model

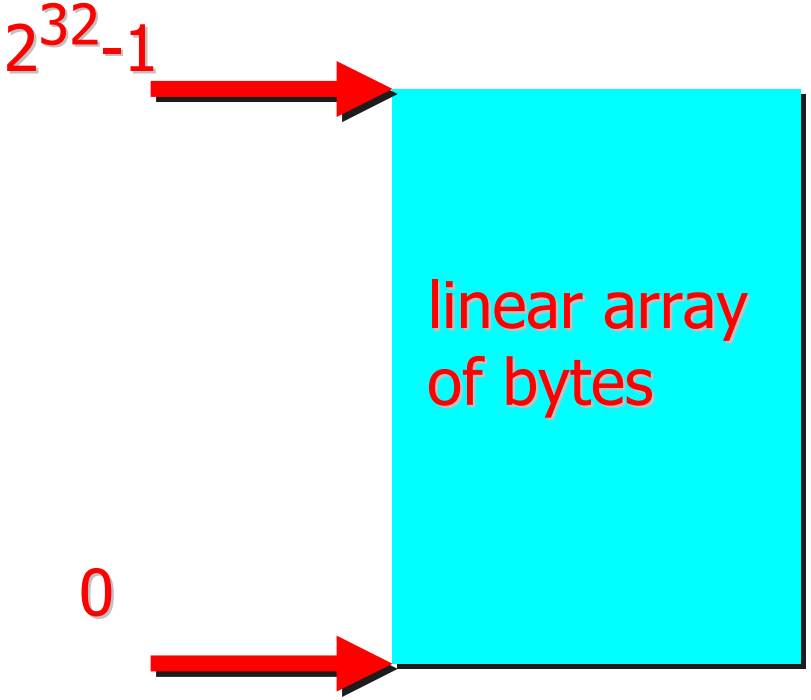
The Instruction Set Architecture (ISA) defines the operations that the programmer can use to change the state of the system incorporating the processor.

This state usually comprises the values of the data items in the visible registers and the memory.

Each instruction performs a defined transformation from the state before the instruction is executed to the state after it has completed.

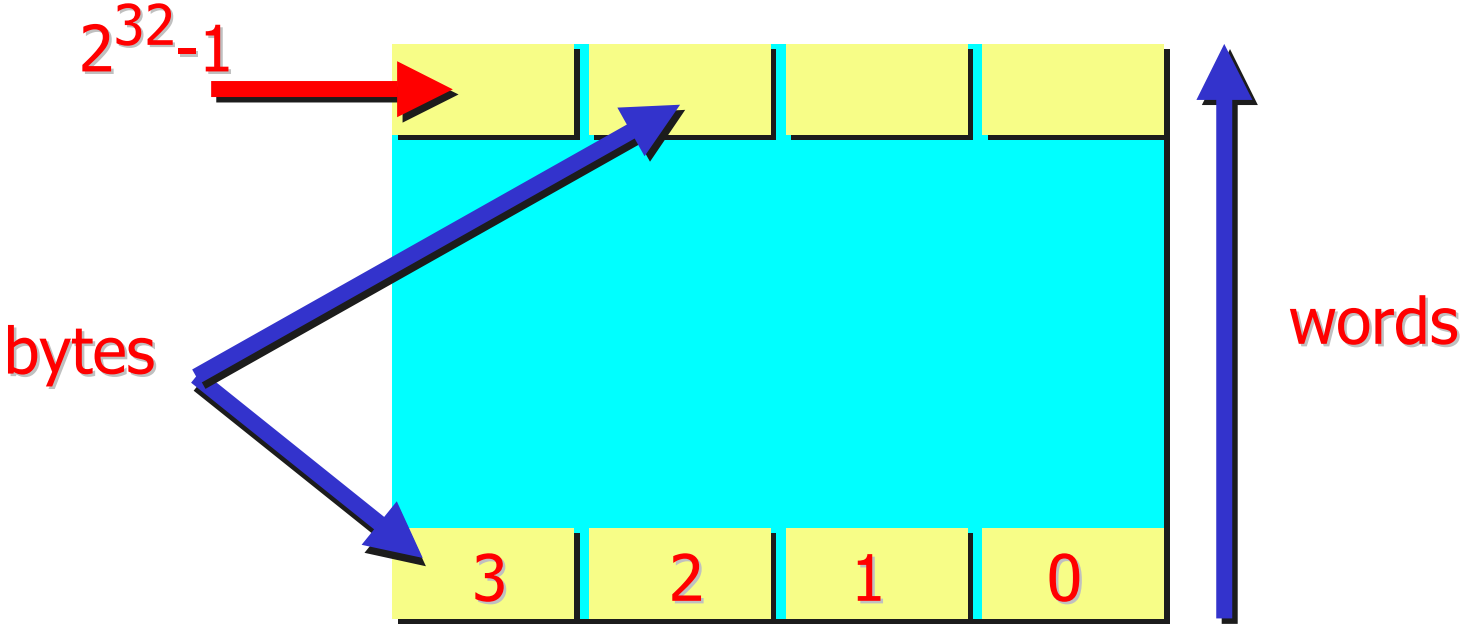
ARM memory subsystem

ARM memory may be viewed as a linear array of bytes numbered from zero up to $2^{32}-1$.



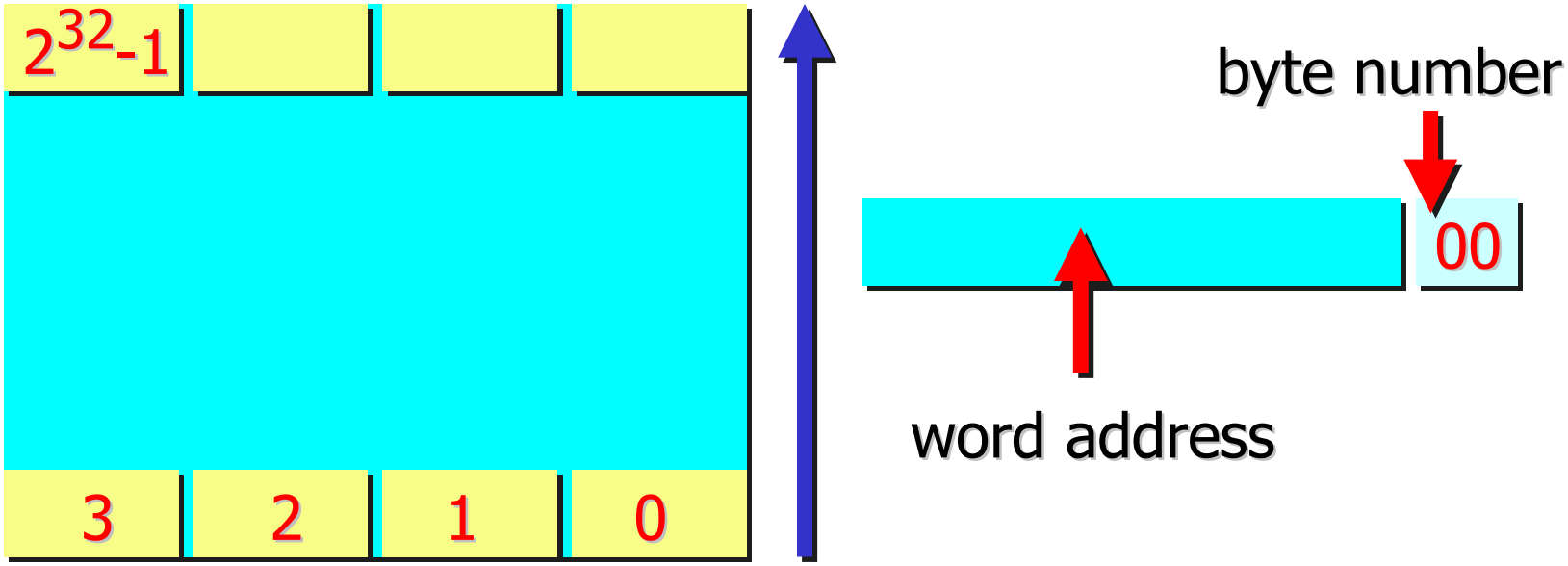
ARM memory subsystem

Data items may be 8-bit bytes, 16-bit half-words or 32-bit words.



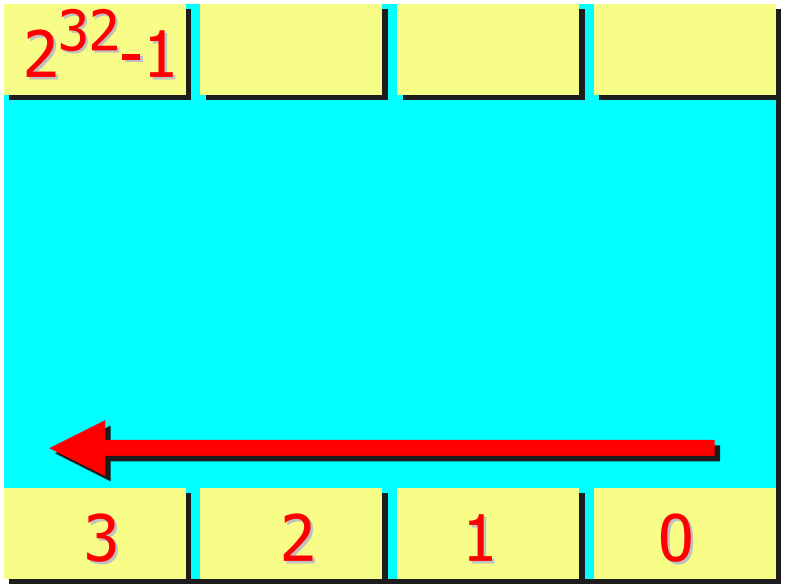
ARM memory subsystem

Words are always aligned on 4-byte boundaries (the two least significant address bits are zero) and half-words are aligned on even byte boundaries.



ARM memory subsystem

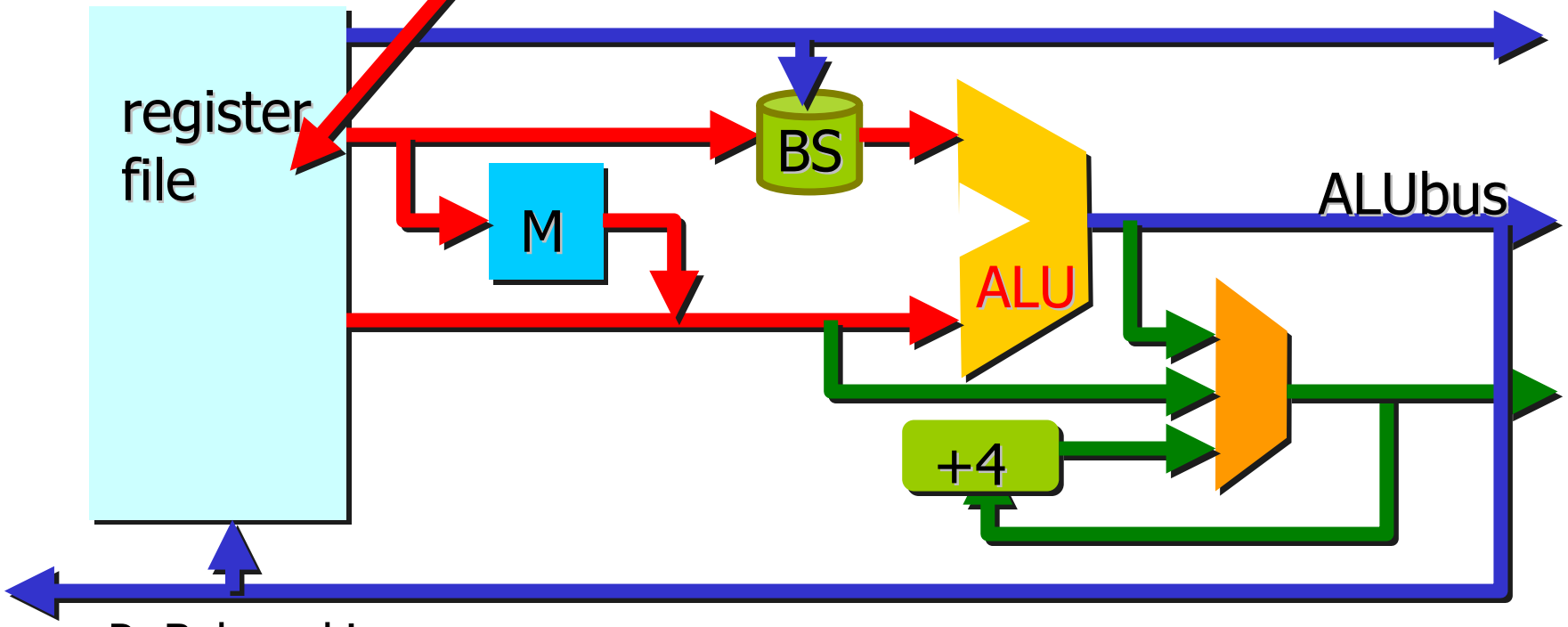
Words are always aligned on 4-byte boundaries (the two least significant address bits are zero) and half-words are aligned on even byte boundaries.



little endian organization

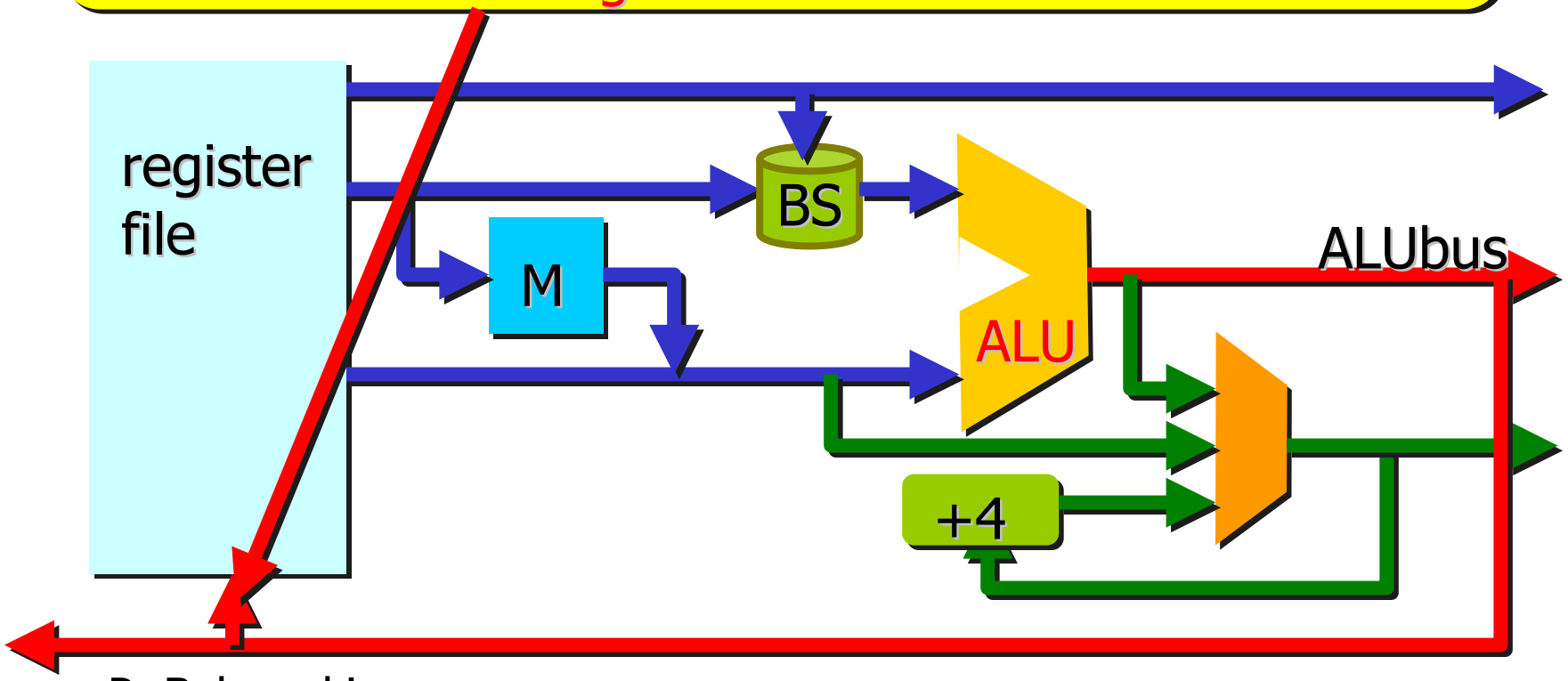
ARM load-store architecture

The processing instruction (add, subtract, and so on) take the values from the registers and always place the results into a register.



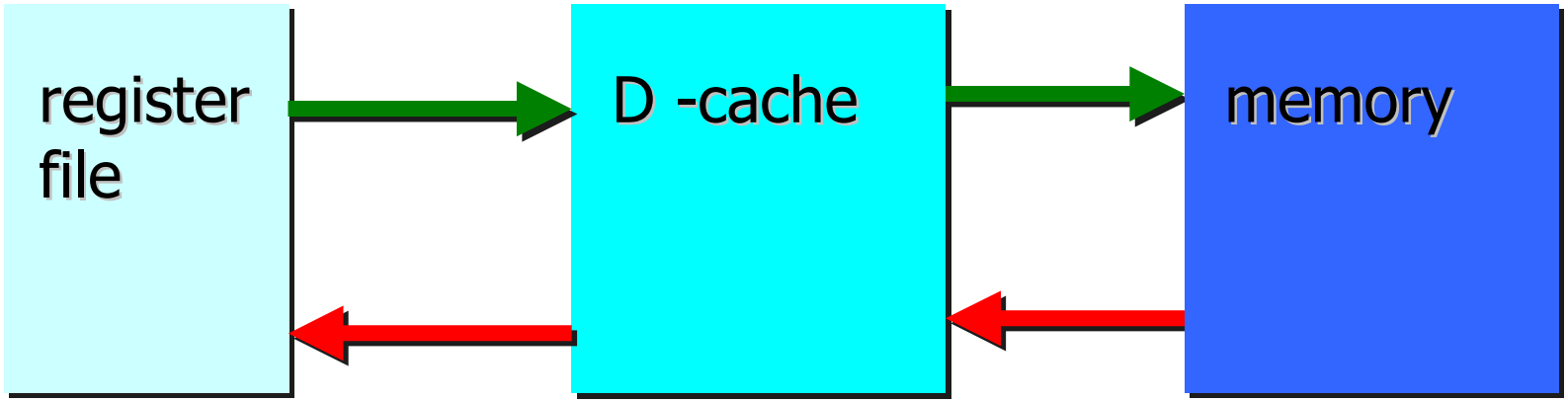
ARM load-store architecture

The processing instruction (add, subtract, and so on) take the values from the registers and always place the results into a register.



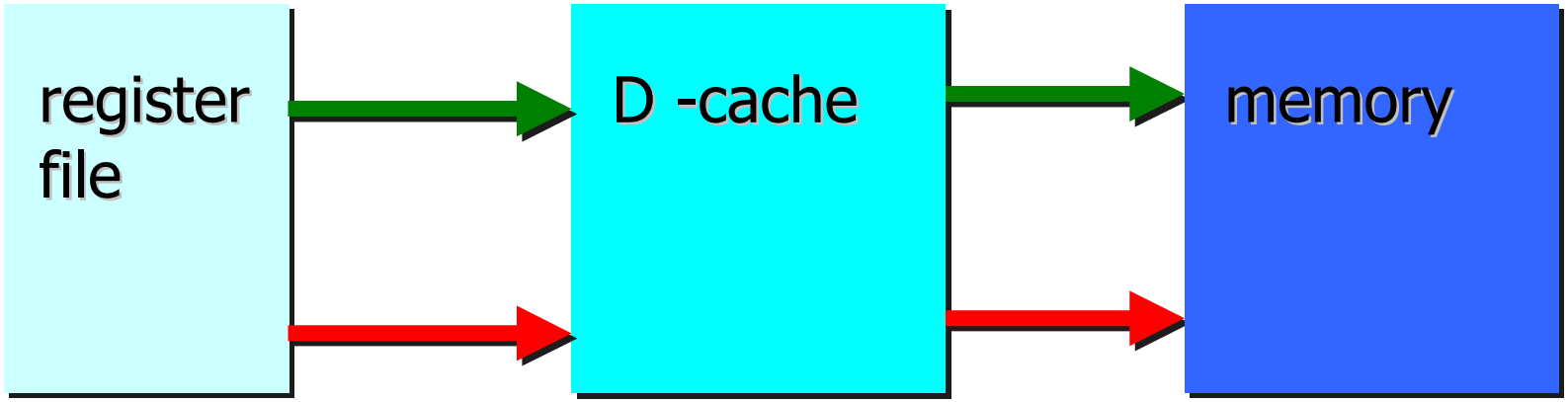
ARM load-store architecture

The only instructions which apply to memory state are ones which copy memory values into register (load instructions) or copy register values into memory (store instructions).



ARM load-store architecture

The only instructions which apply to memory state are ones which copy memory values into register (load instructions) or copy register values into memory (store instructions).



ARM instructions

In general the ARM instructions fall into one of the following three categories:

- data processing instructions
- data transfer instructions
- control flow instructions

ARM instructions

In general the ARM instructions fall into one of the following three categories:

- data processing instructions
- data transfer instructions
- control flow instructions

ARM instructions

In general the ARM instructions fall into one of the following three categories:

- data processing instructions
- data transfer instructions
- control flow instructions

ARM instructions

In general the ARM instructions fall into one of the following three categories:

- data processing instructions
- data transfer instructions
- control flow instructions

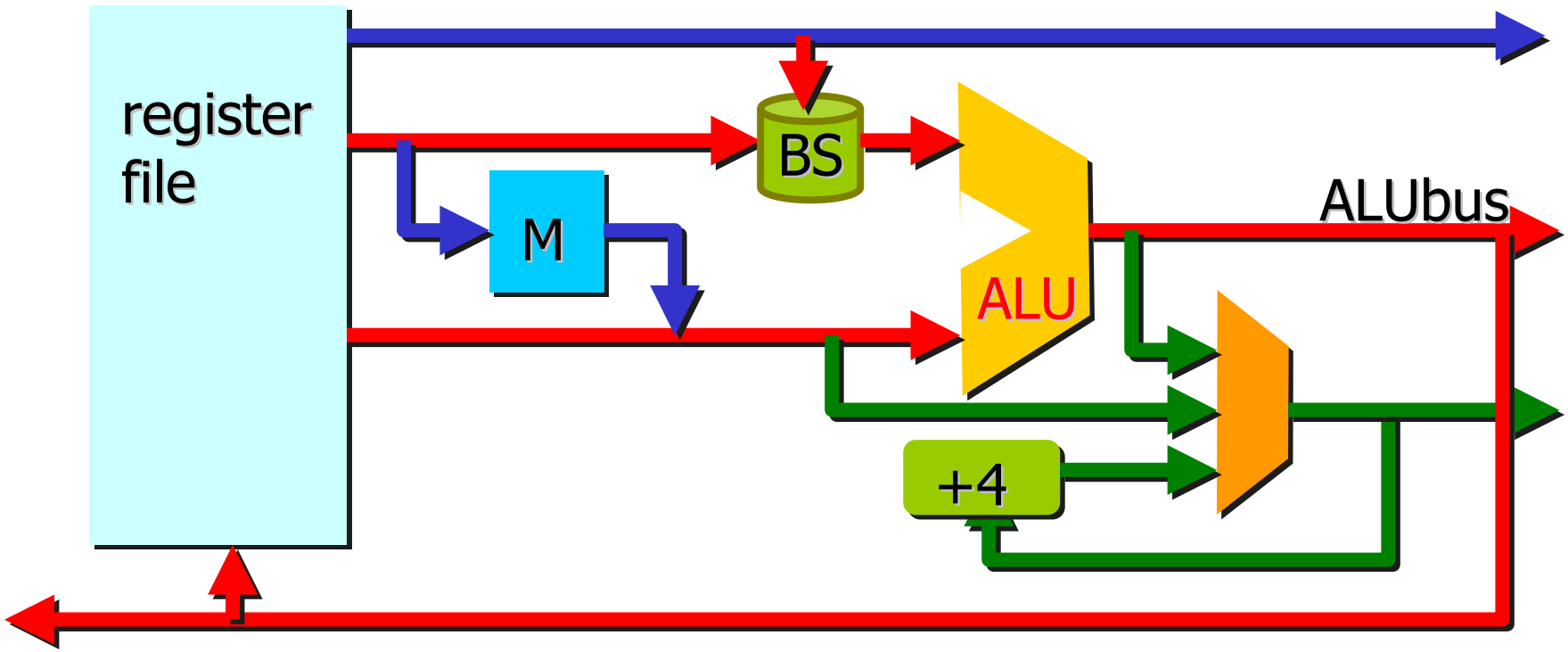
ARM data processing

Data processing instructions:

these use and change only register values;

ARM data processing

For example, an instruction can add two registers and place the result in a register.



ARM data transfer

Data transfer instructions copy memory values into registers (load instructions) or copy register values into memory (store instructions);

An additional form, useful only in systems code, exchanges a memory value with a register value.

ARM data transfer

Data transfer instructions copy memory values into registers (load instructions) or copy register values into memory (store instructions);

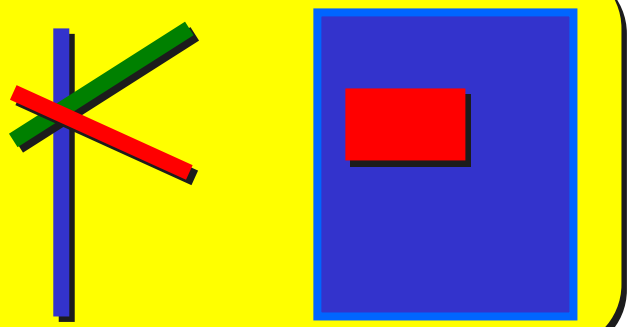
An additional form, useful only in systems code, exchanges a memory value with a register value.

ARM data transfer

Data transfer instructions copy memory values into registers (load instructions) or copy register values into memory (store instructions);

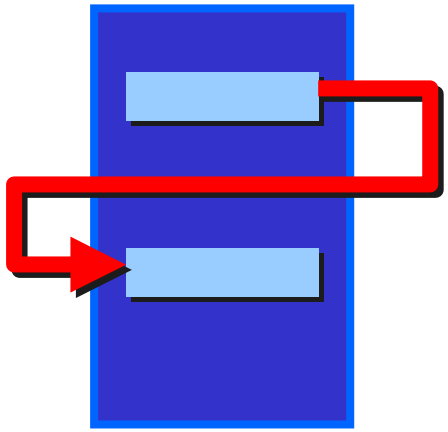
An additional form, useful only in systems code, exchanges a memory value with a register value.

e.g. test and set instruction



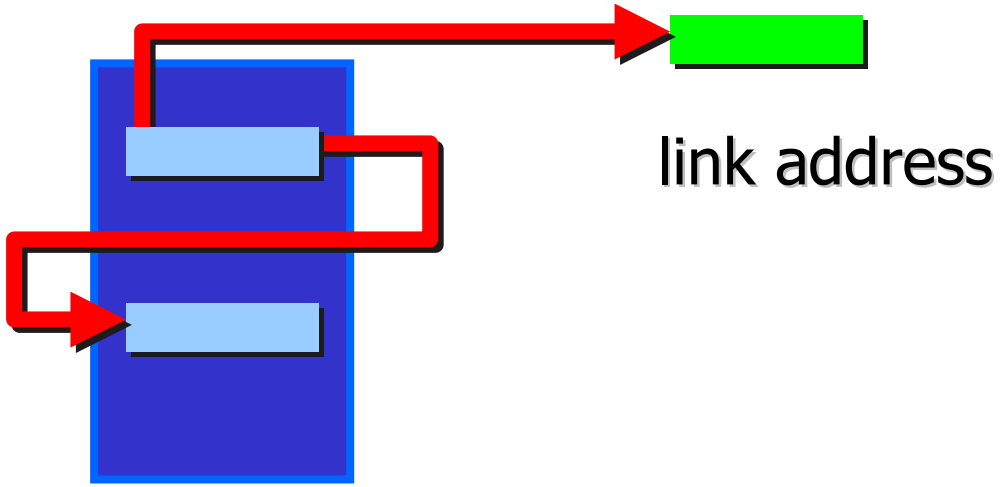
ARM control flow

Control flow instructions cause execution to switch to a different address, either permanently (branch instructions) or saving a return address to resume the original sequence (branch and link instructions) or trapping into system code (supervisor calls).



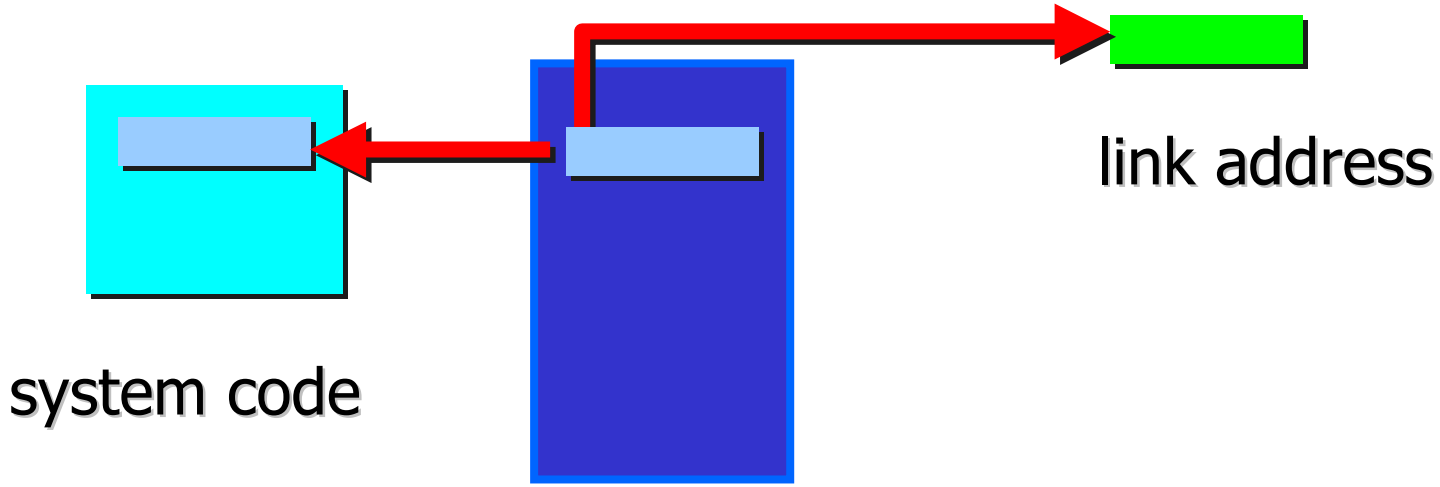
ARM control flow

Control flow instructions cause execution to switch to a different address, either permanently (branch instructions) or saving a return address to resume the original sequence (branch and link instructions) or trapping into system code (supervisor calls).



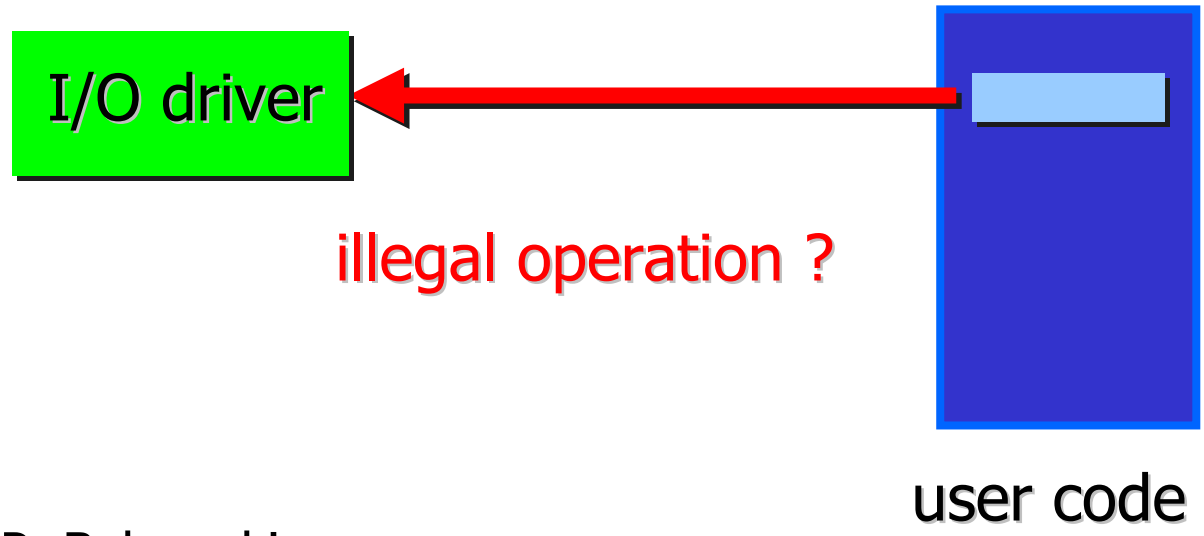
ARM control flow

Control flow instructions cause execution to switch to a different address, either permanently (branch instructions) or saving a return address to resume the original sequence (branch and link instructions) or trapping into system code (supervisor calls).



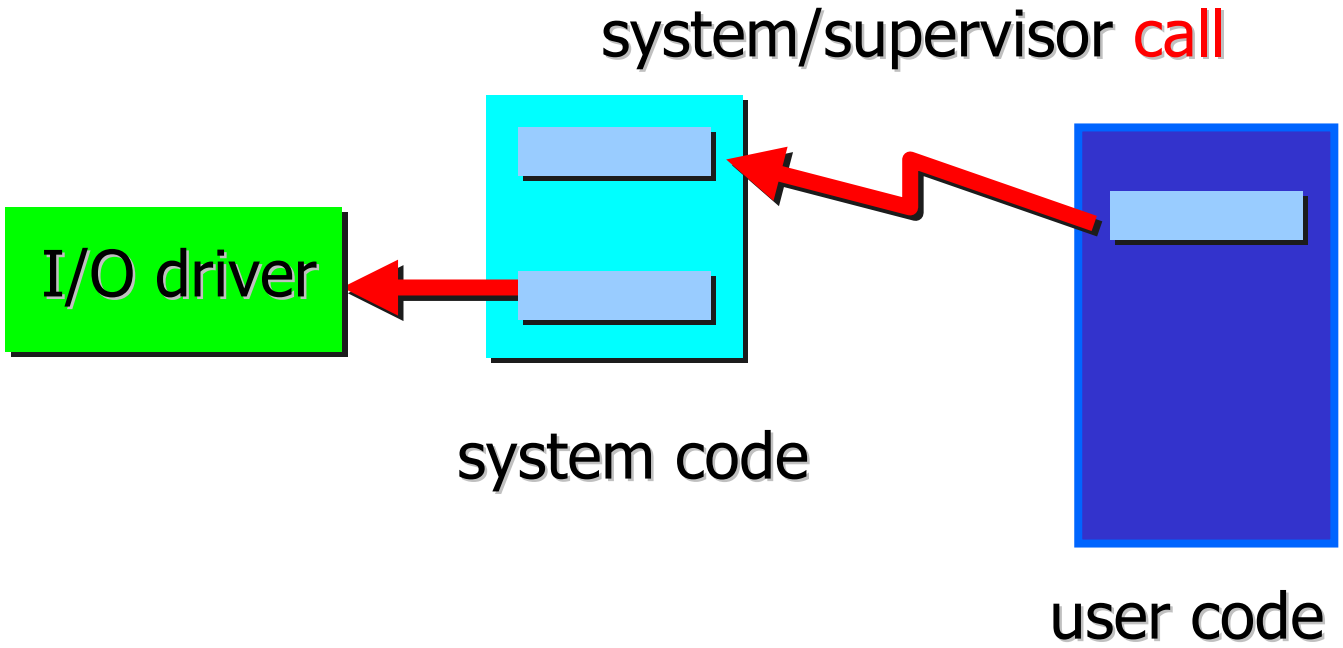
ARM supervisor mode

The ARM processor supports a protected supervisor mode. The protection mechanism ensures that user code cannot gain supervisor privileges without appropriate checks being carried out to ensure that the code is not attempting illegal operations.



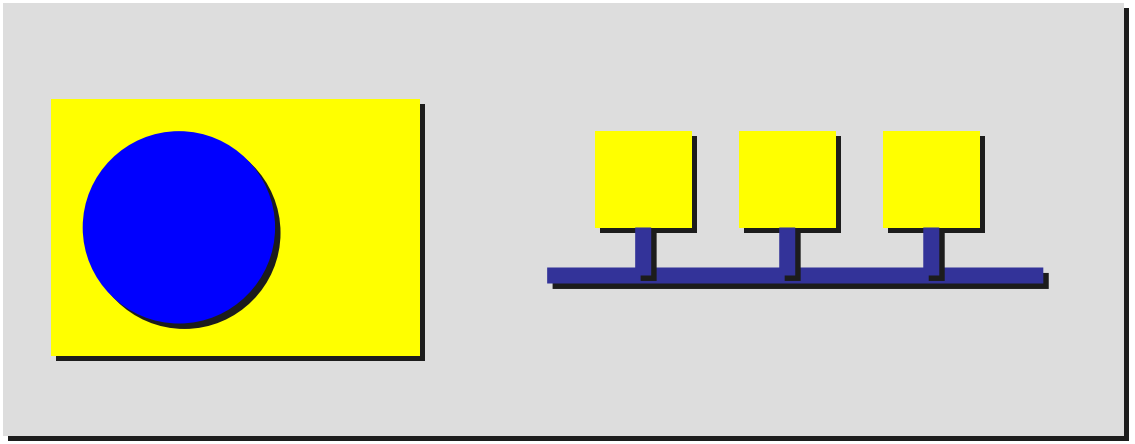
ARM supervisor mode

The upshot of this for the user-level programmer is that system-level functions can only be accessed through specified supervisor call.



ARM I/O programming

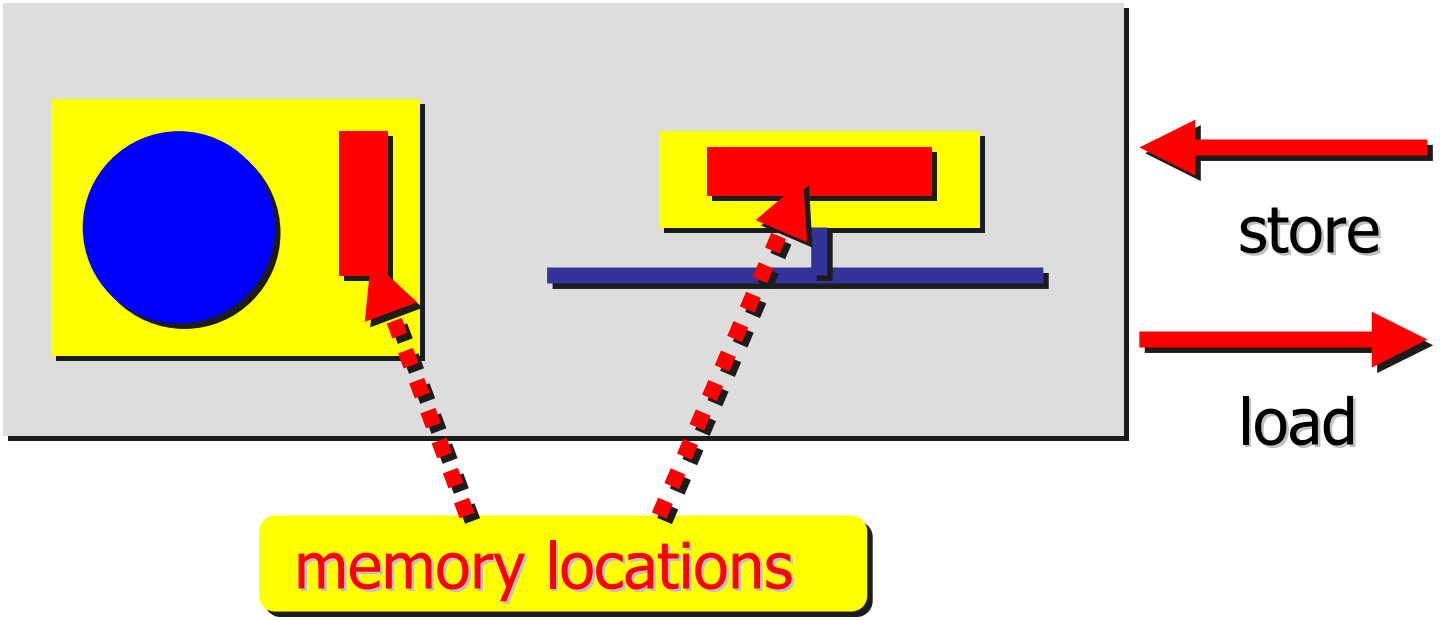
The ARM handles I/0 (input/output) peripherals (such as disk controllers, network interfaces, and so on) as memory-mapped devices with interrupt support.



memory-mapped devices

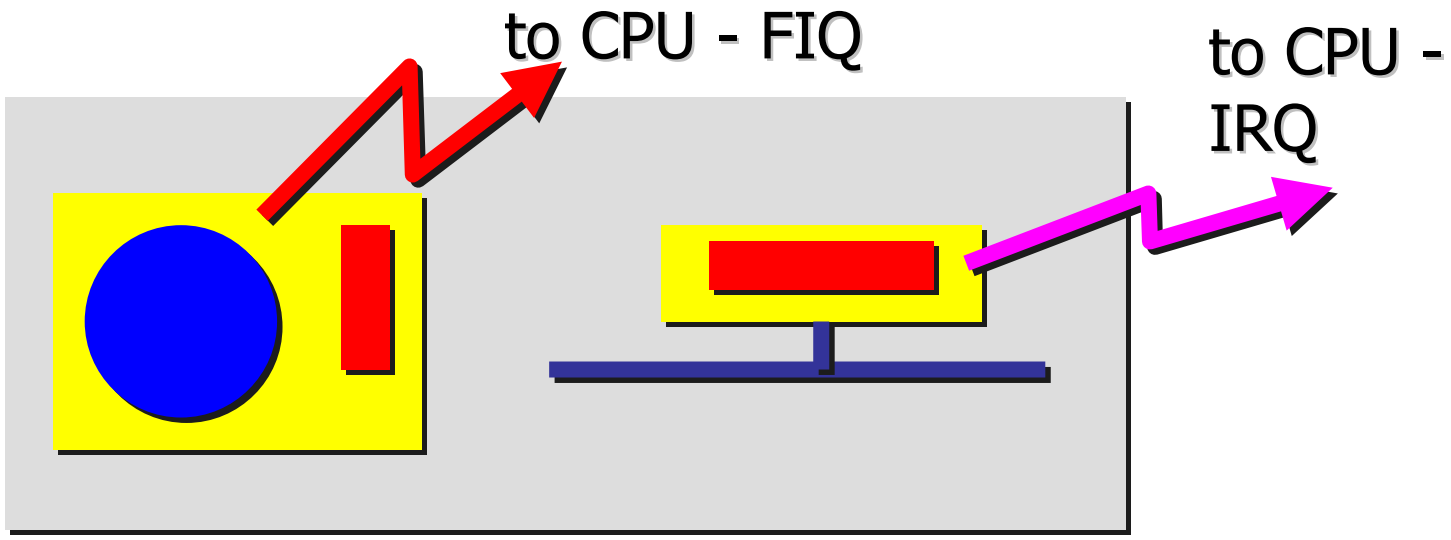
ARM I/O programming

The internal registers in these devices appear as addressable locations within the ARM's memory map and may be read and written using the same (load-store) instructions as any other memory locations.



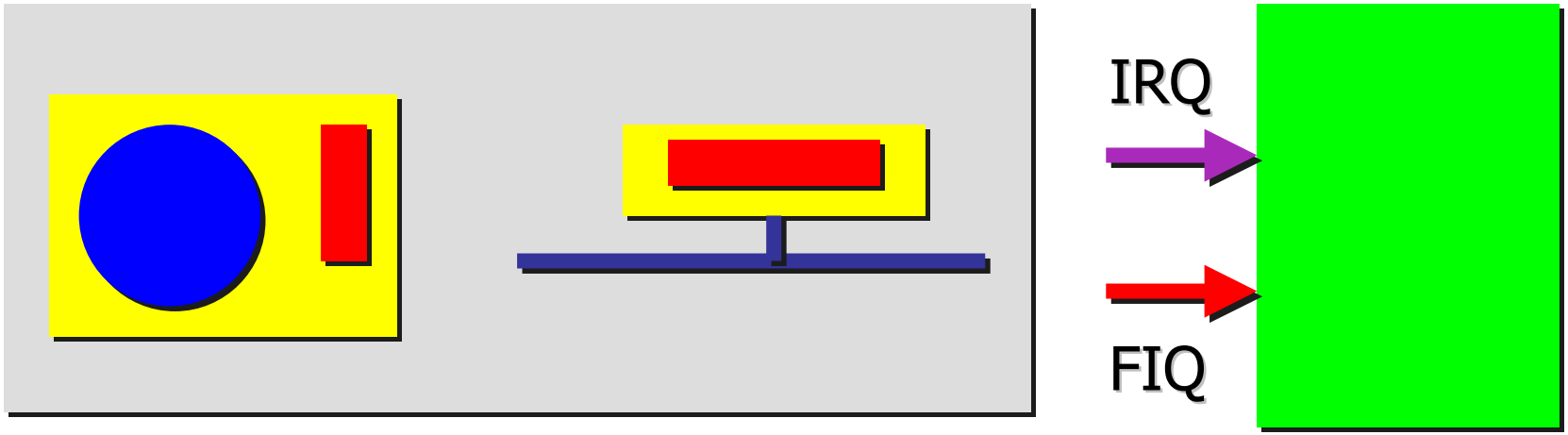
ARM I/O interruptions

Peripherals may attract the processor's attention by making an interrupt request using either the normal interrupt (IRQ) or the fast interrupt (FIQ) input.



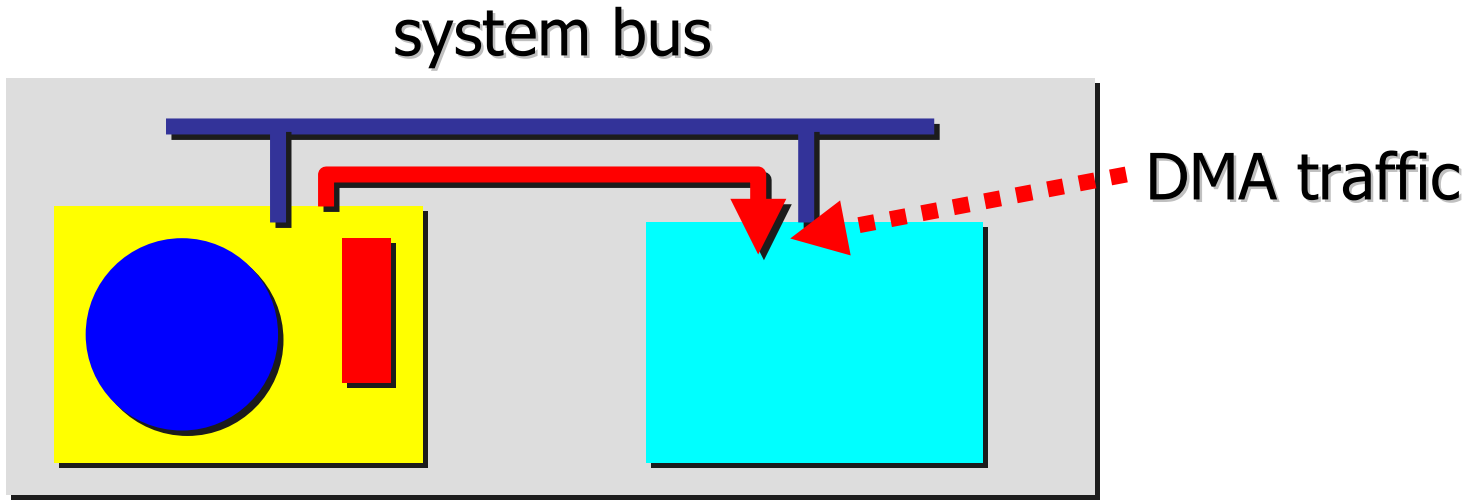
ARM I/O interruptions

Both interrupt inputs are level-sensitive and maskable. Normally most interrupt sources share the IRQ input, with just one or two time-critical sources connected to the higher-priority FIQ input.



ARM I/O interruptions

Some systems may include direct memory access (DMA) hardware external to the processor to handle high-bandwidth traffic.



ARM exceptions

The ARM architecture supports a range of :

- interrupts
- traps
- supervisor calls

all grouped under the general heading of exceptions.

ARM exceptions

The ARM architecture supports a range of :

- interrupts
- traps
- supervisor calls

all grouped under the general heading of exceptions.

ARM exceptions

The ARM architecture supports a range of :

- interrupts
- traps
- supervisor calls

all grouped under the general heading of exceptions.

ARM exceptions

The ARM architecture supports a range of :

- interrupts
- traps
- supervisor calls

all grouped under the general heading of exceptions.

ARM exceptions

The ARM architecture supports a range of :

- interrupts
- traps
- supervisor calls

all grouped under the general heading of exceptions.

ARM exceptions

The general way of exception handling is the same in all cases:

- the current state is saved by copying the PC into rl4_exc and the CPSR into SPSR_exc (where *exc* stands for the exception type);
- the processor operating mode is changed to the appropriate exception mode;
- the PC is forced to a value between 00_{16} and $1C_{16}$, the particular value depending on the type of exception.

ARM exceptions

The general way of exception handling is the same in all cases:

- the current state is saved by copying the PC into rl4_exc and the CPSR into SPSR_exc (where *exc* stands for the exception type);
- the processor operating mode is changed to the appropriate exception mode;
- the PC is forced to a value between 00_{16} and $1C_{16}$, the particular value depending on the type of exception.

ARM exceptions

The general way of exception handling is the same in all cases:

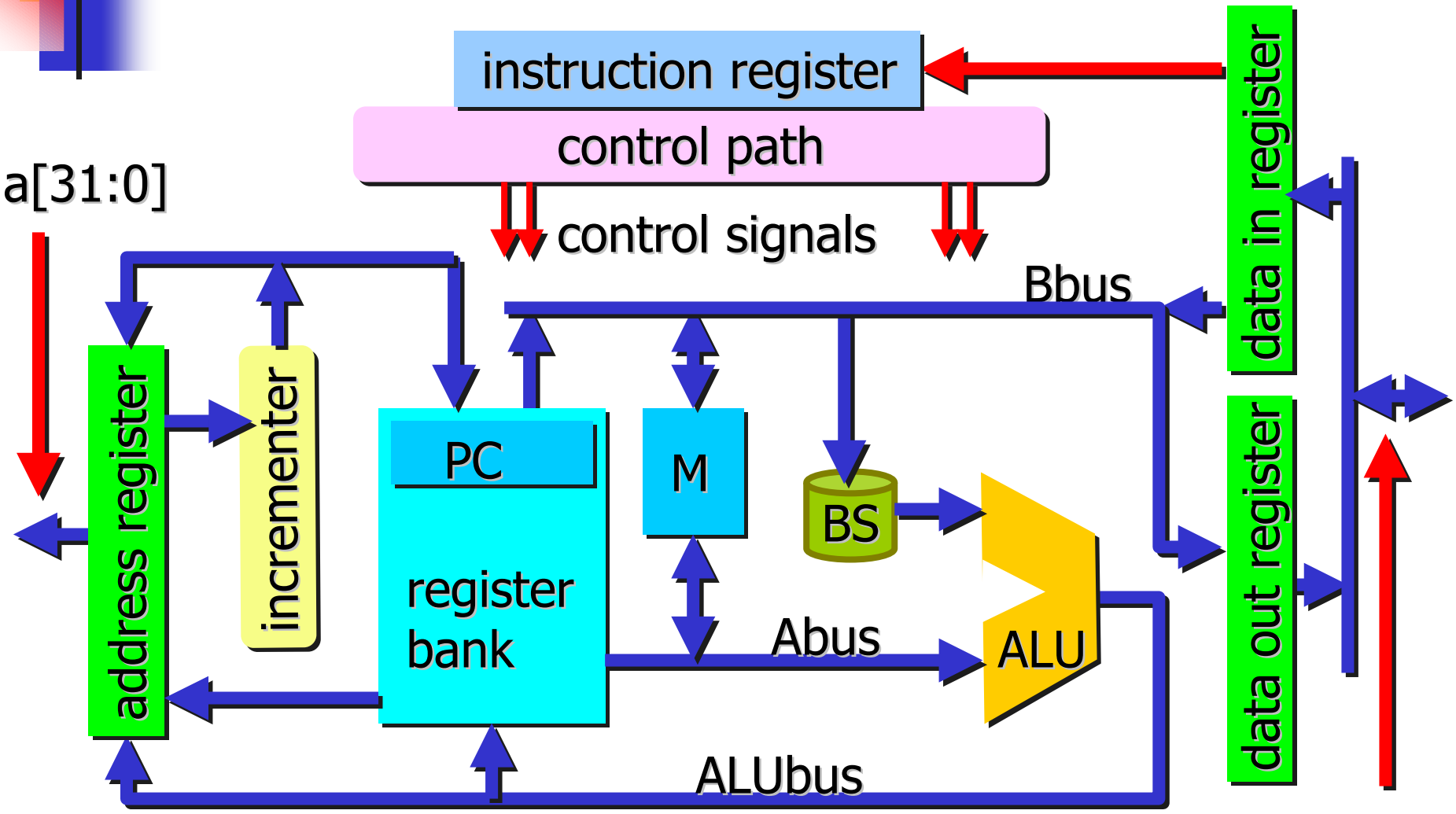
- the current state is saved by copying the PC into rl4_exc and the CPSR into SPSR_exc (where *exc* stands for the exception type);
- the processor operating mode is changed to the appropriate exception mode;
- the PC is forced to a value between 00_{16} and $1C_{16}$, the particular value depending on the type of exception.

ARM exceptions

The general way of exception handling is the same in all cases:

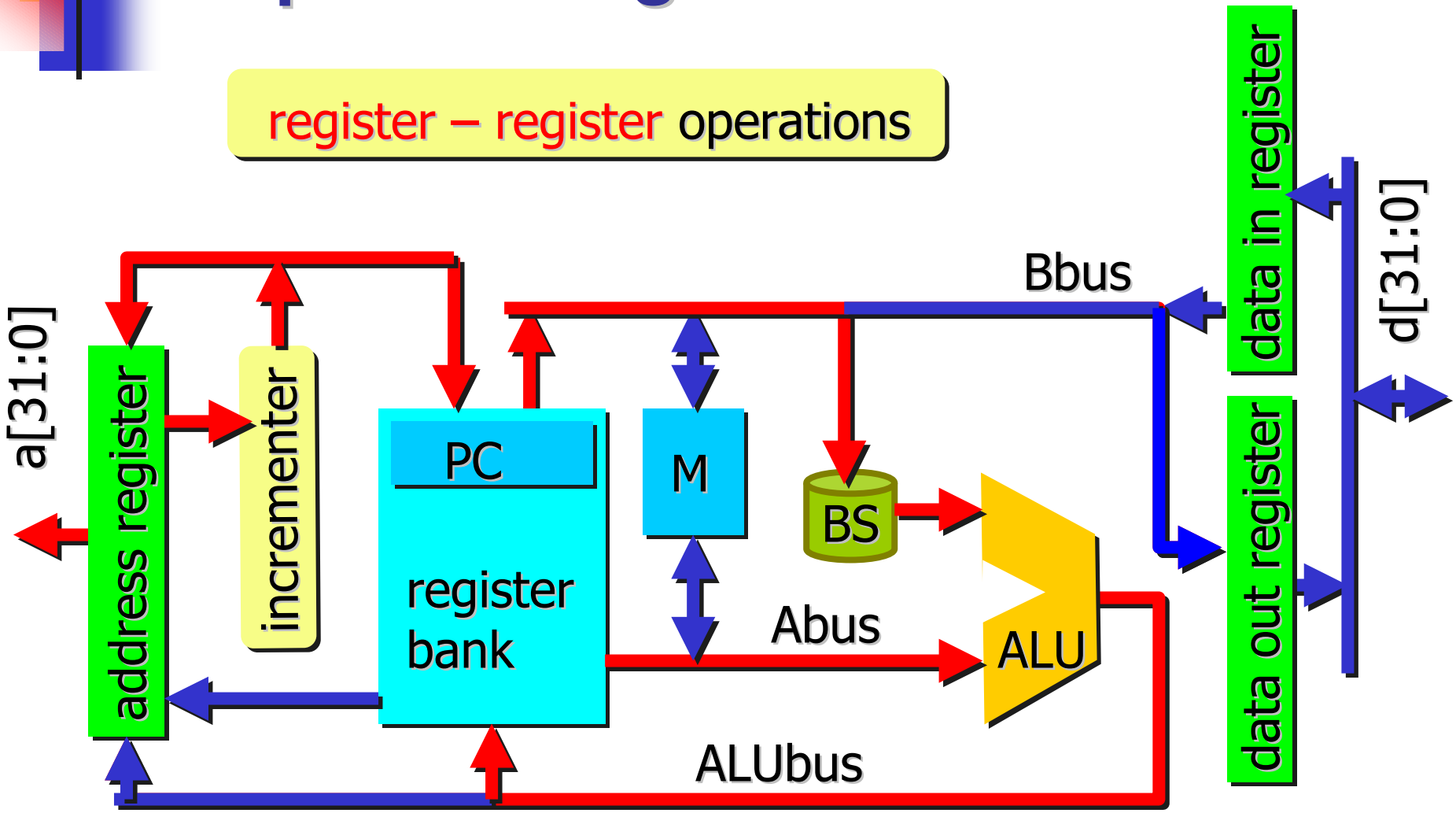
- the current state is saved by copying the PC into rl4_exc and the CPSR into SPSR_exc (where *exc* stands for the exception type);
- the processor operating mode is changed to the appropriate exception mode;
- the PC is forced to a value between 00_{16} and $1C_{16}$, the particular value depending on the type of exception.

ARM instruction execution

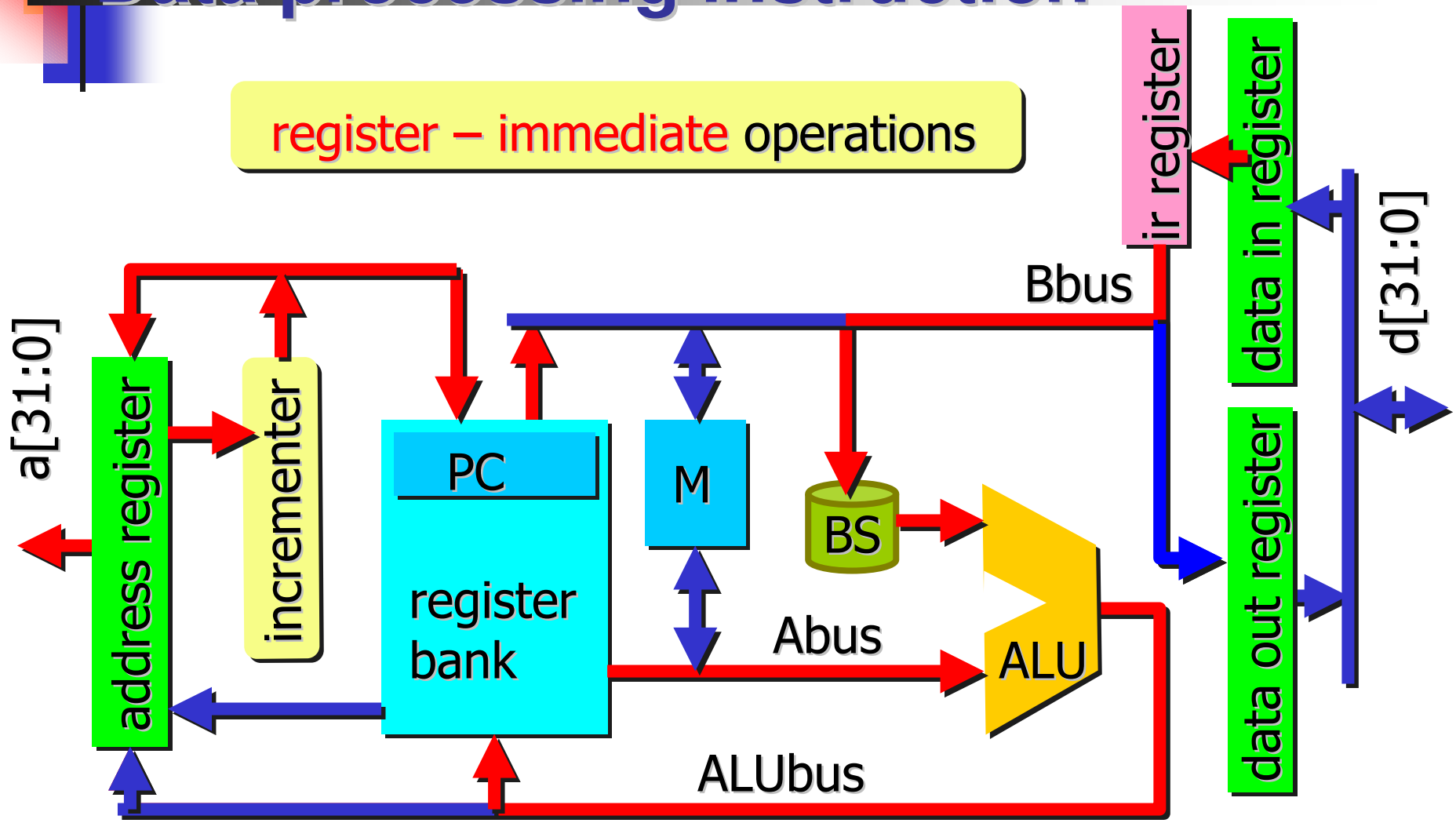


M – multiplier, BS – barrel shifter

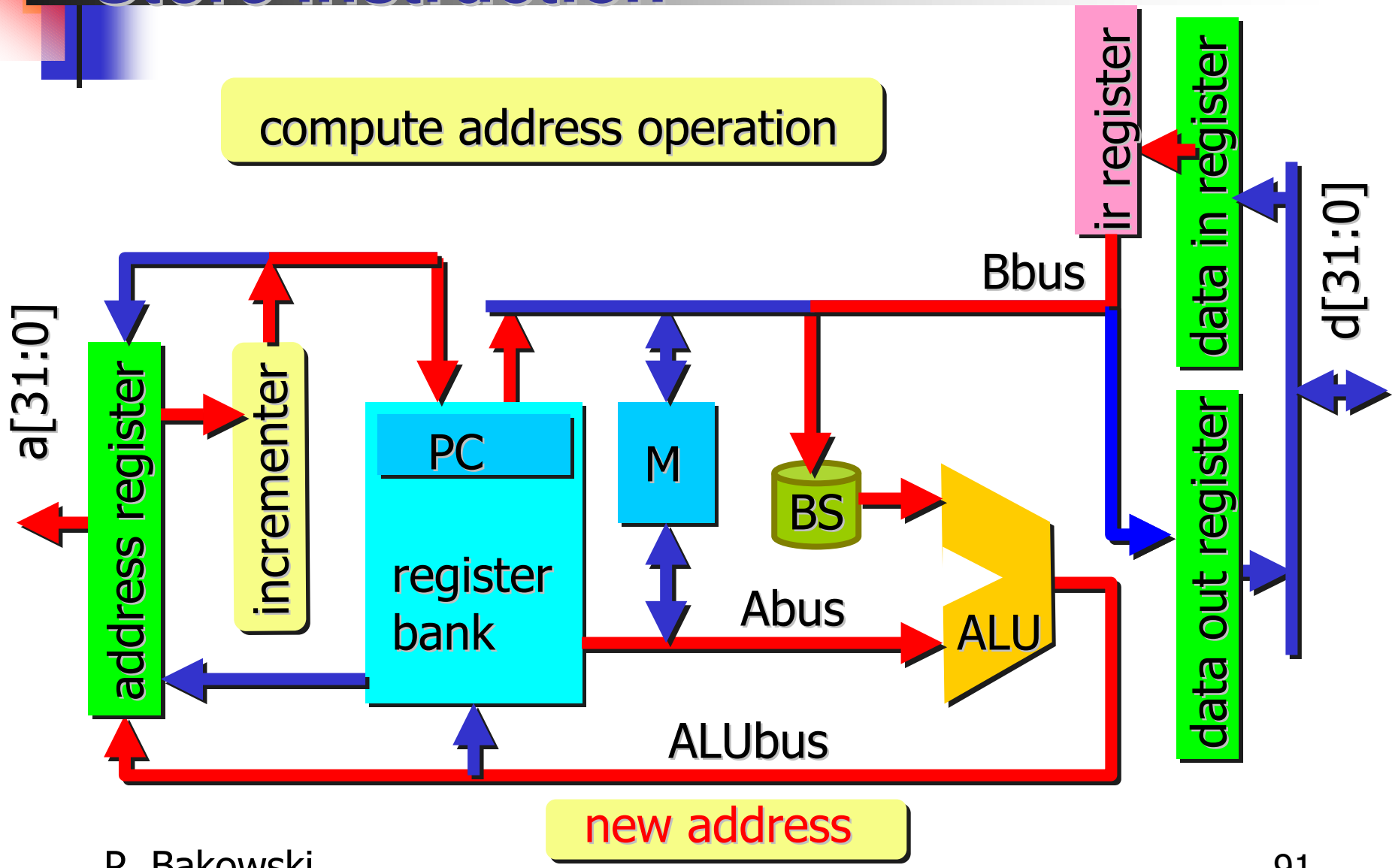
Data processing instruction



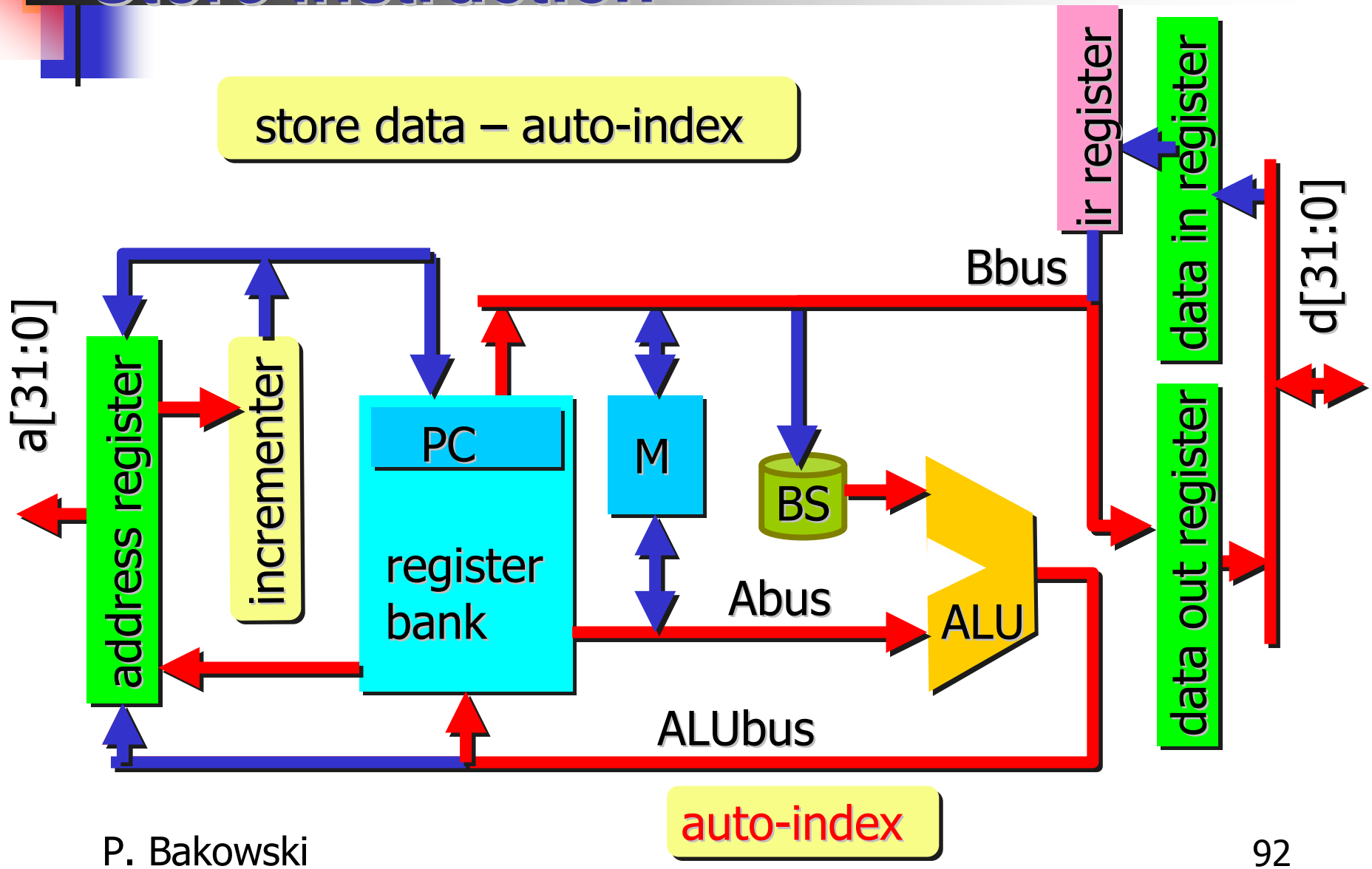
Data processing instruction



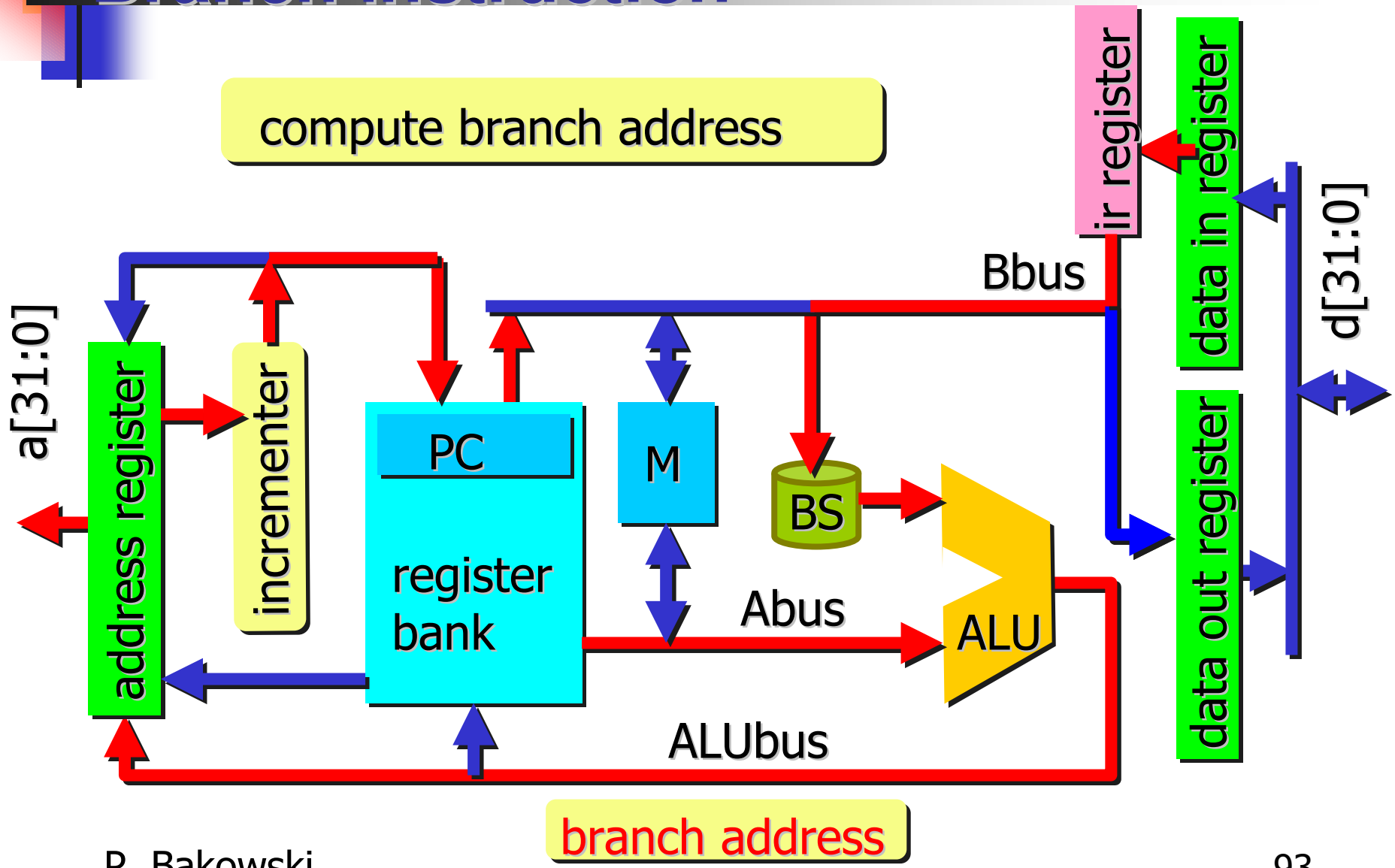
Store instruction



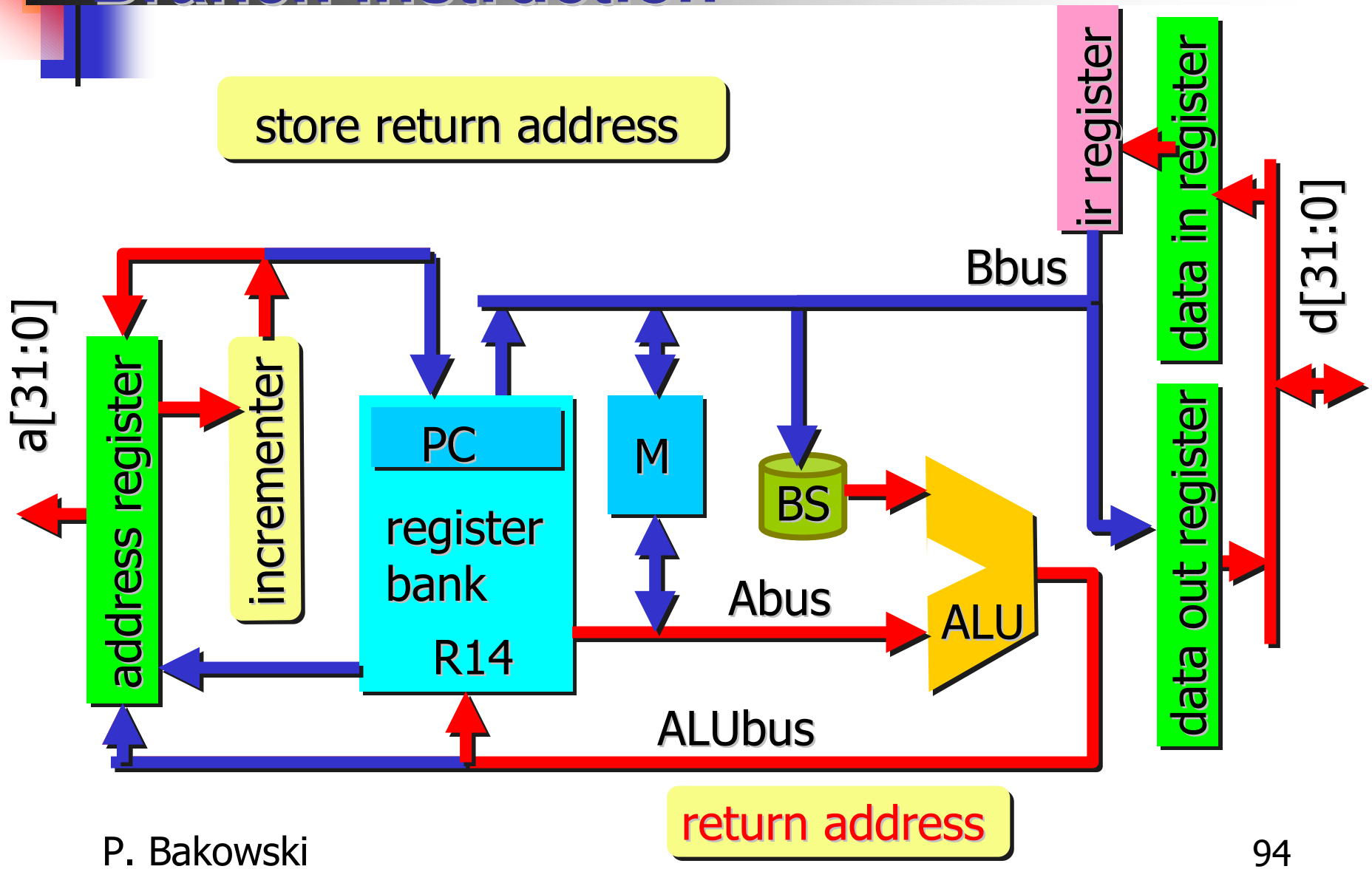
Store instruction

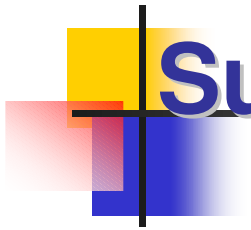


Branch instruction



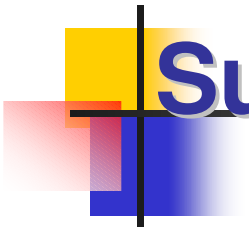
Branch instruction





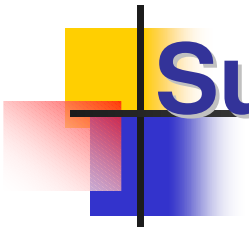
Summary

- ARM register bank
- ARM barrel shifter and ALU
- ARM 3-stage and 5-stage pipelines
- ARM programming model
- ARM instructions



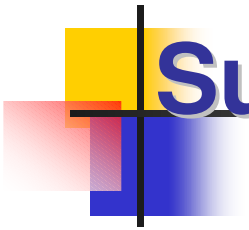
Summary

- ARM register bank
- ARM barrel shifter and ALU
- ARM 3-stage and 5-stage pipelines
- ARM programming model
- ARM instructions



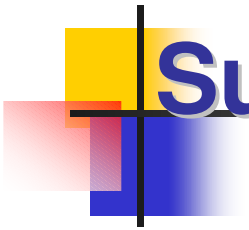
Summary

- ARM register bank
- ARM barrel shifter and ALU
- ARM 3-stage and 5-stage pipelines
- ARM programming model
- ARM instructions



Summary

- ARM register bank
- ARM barrel shifter and ALU
- ARM 3-stage and 5-stage pipelines
- ARM programming model
- ARM instructions



Summary

- ARM register bank
- ARM barrel shifter and ALU
- ARM 3-stage and 5-stage pipelines
- ARM programming model
- ARM instructions