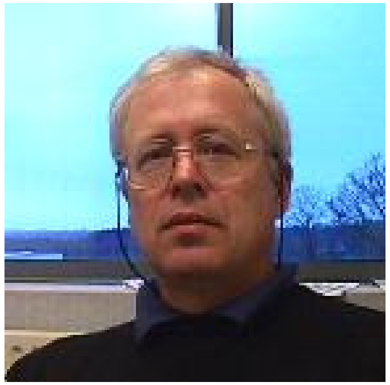
An introduction to processor design

P. Bakowski



bako@ieee.org

A simple processor

- a program counter : PC
- an accumulator
- an instruction register
- instruction decode and control logic
- an arithmetic-logic unit

A simple processor

- a program counter
- an accumulator : ACC
- an instruction register
- instruction decode and control logic
- an arithmetic-logic unit

A simple processor

- a program counter
- an accumulator
- an instruction register : IR
- instruction decode and control logic
- an arithmetic-logic unit

A simple processor

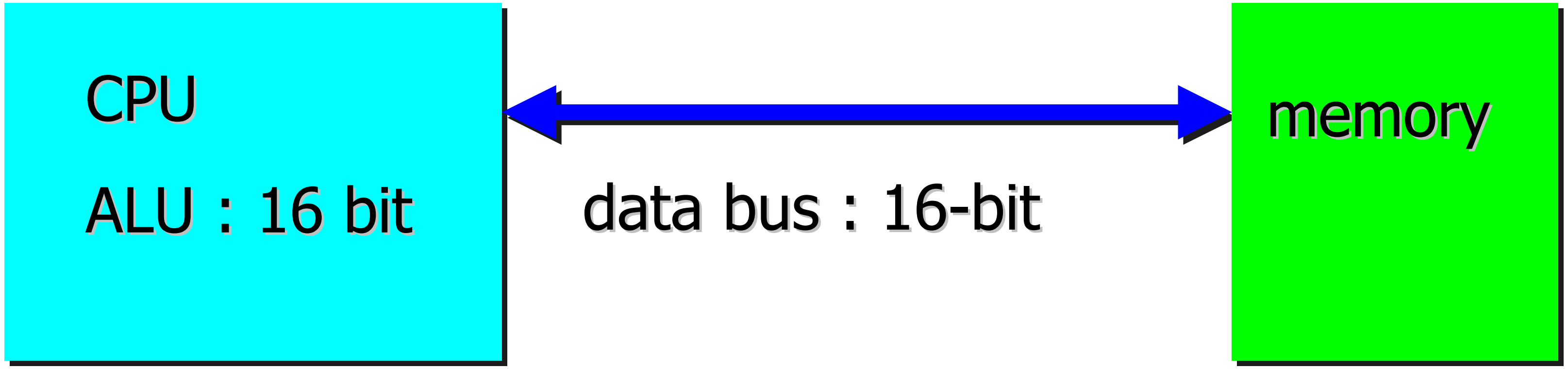
- a program counter
- an accumulator
- an instruction register
- instruction decode and control logic
- an arithmetic-logic unit

A simple processor

- a program counter
- an accumulator
- an instruction register
- instruction decode and control logic
- an arithmetic-logic unit : ALU

A 16-bit processor

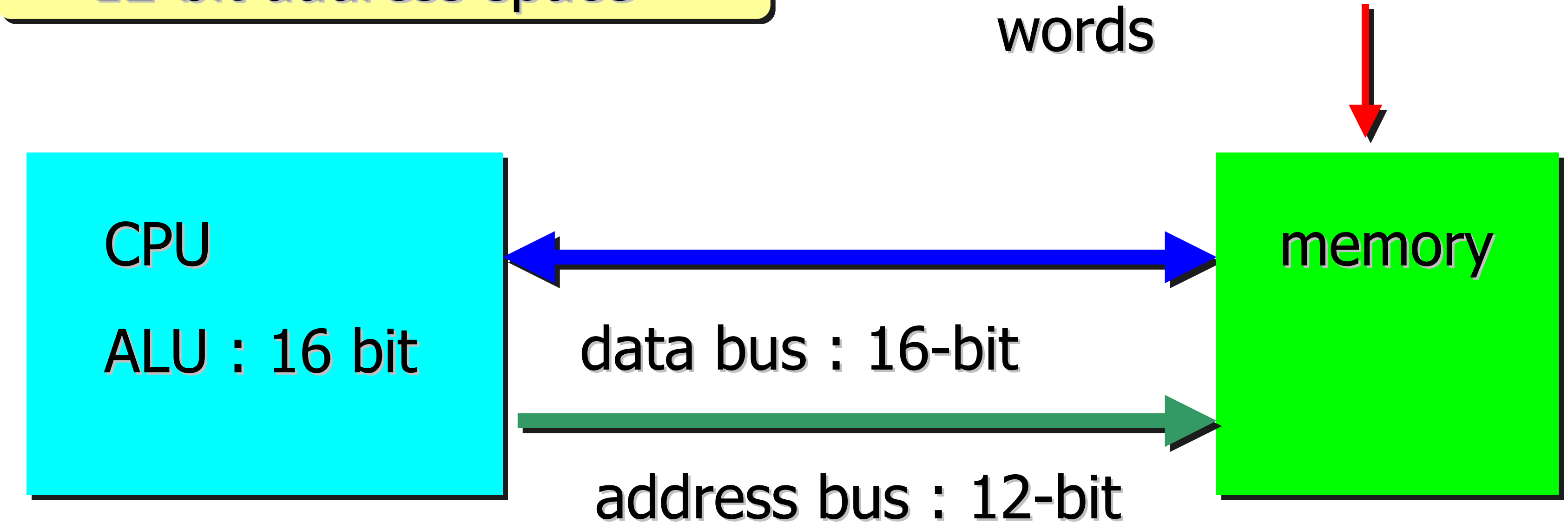
- 16-bit processor
- 12-bit address space



A 16-bit processor

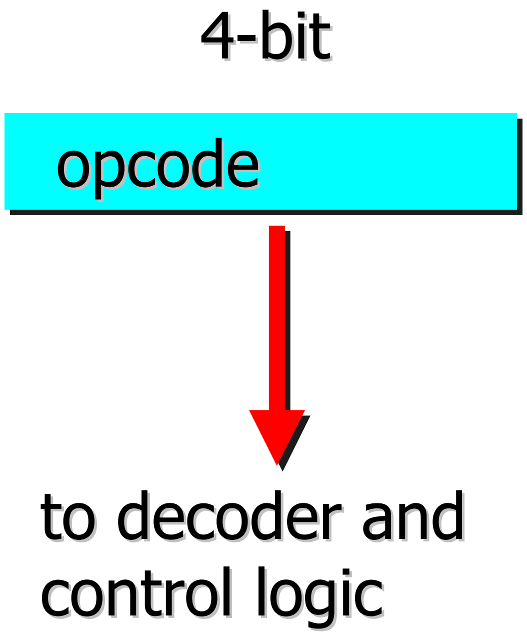
- 16-bit processor
- 12-bit address space

4 096 individually addressable 16-bit words



Instruction format

instruction format : a 16-bit word



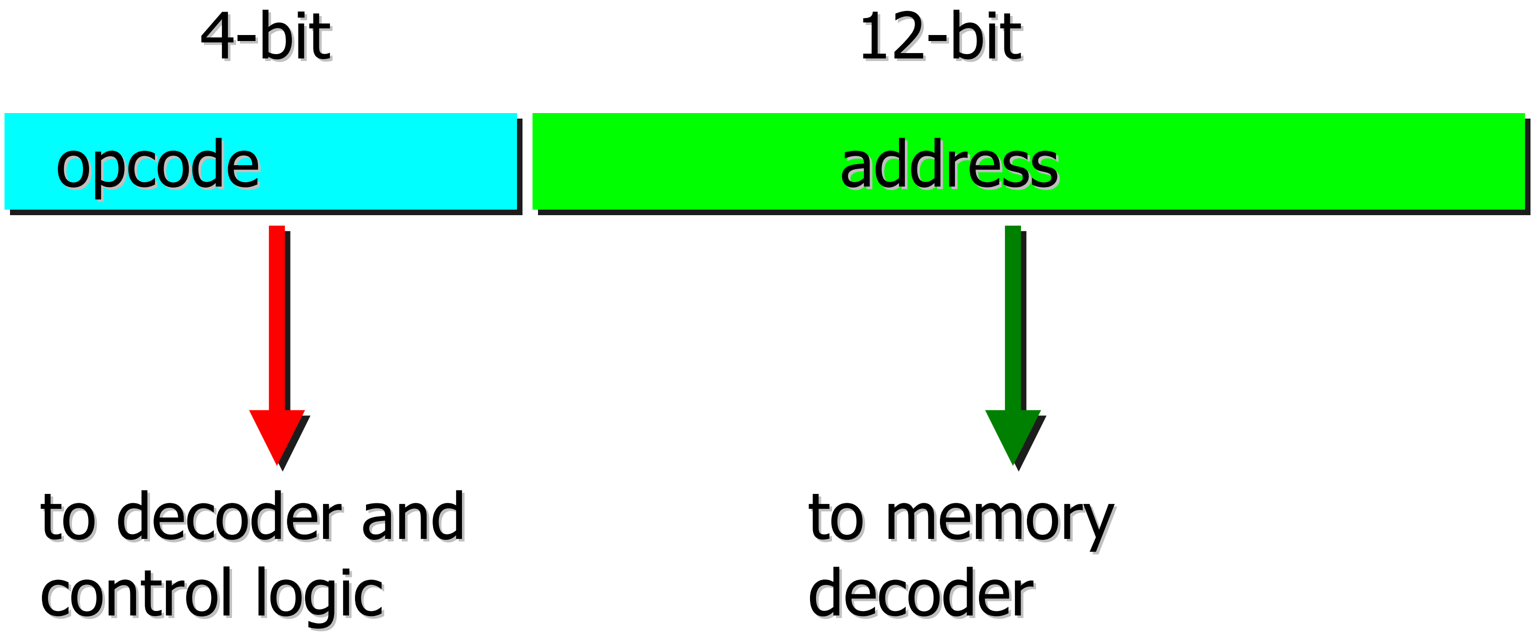
4-bit

opcode

to decoder and control logic

Instruction format

instruction format : a 16-bit word



4-bit

12-bit

opcode

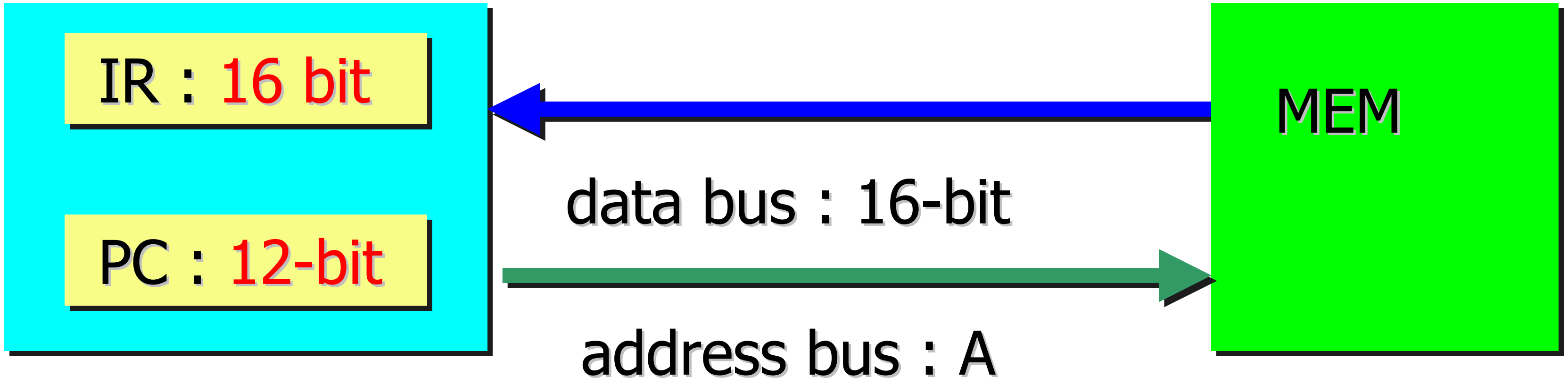
address

to decoder and
control logic

to memory
decoder

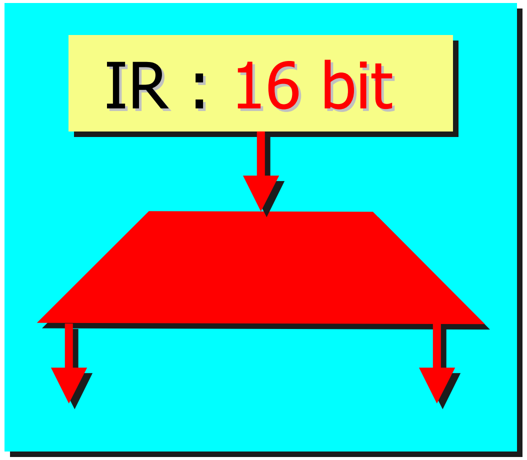
Instruction fetch

FETCH: load new instruction to Instruction Register



Instruction decode

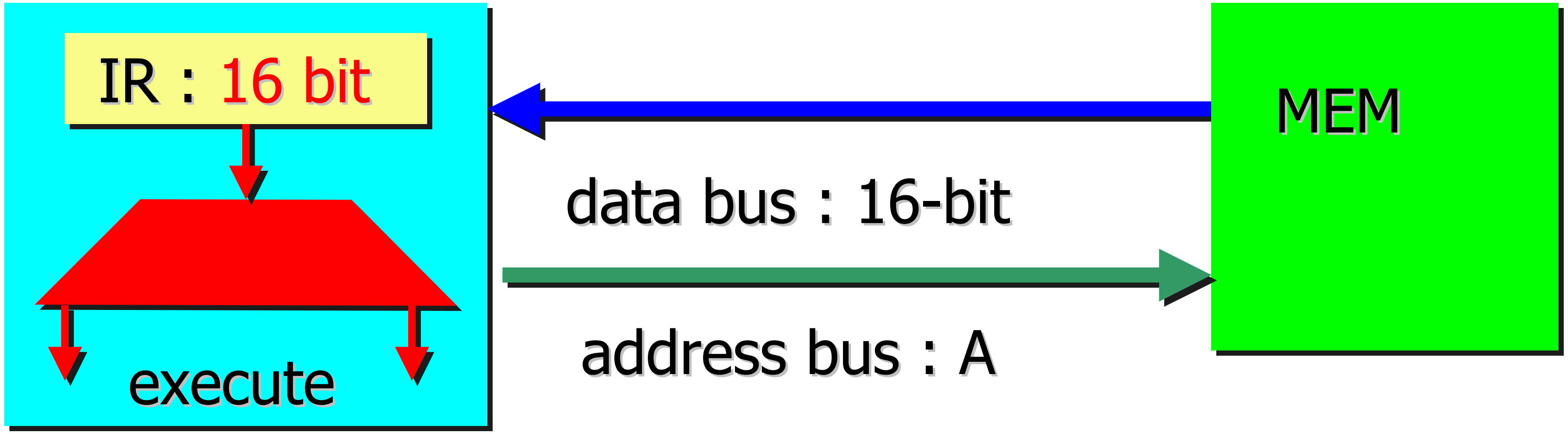
DECODE: decode new instruction



selection and control signals generation

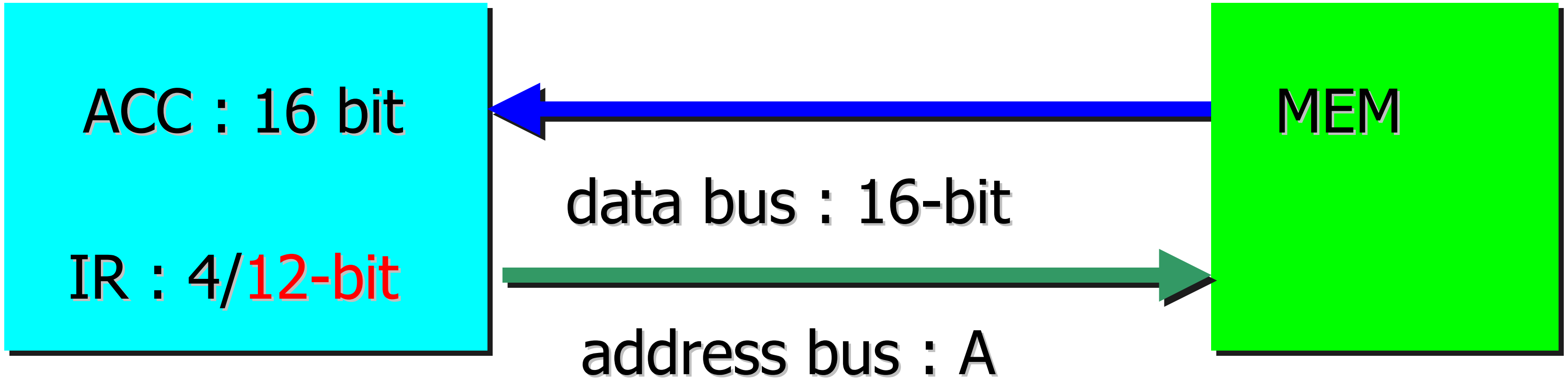
Instruction execute

EXECUTE: execute the instruction depending on opcode



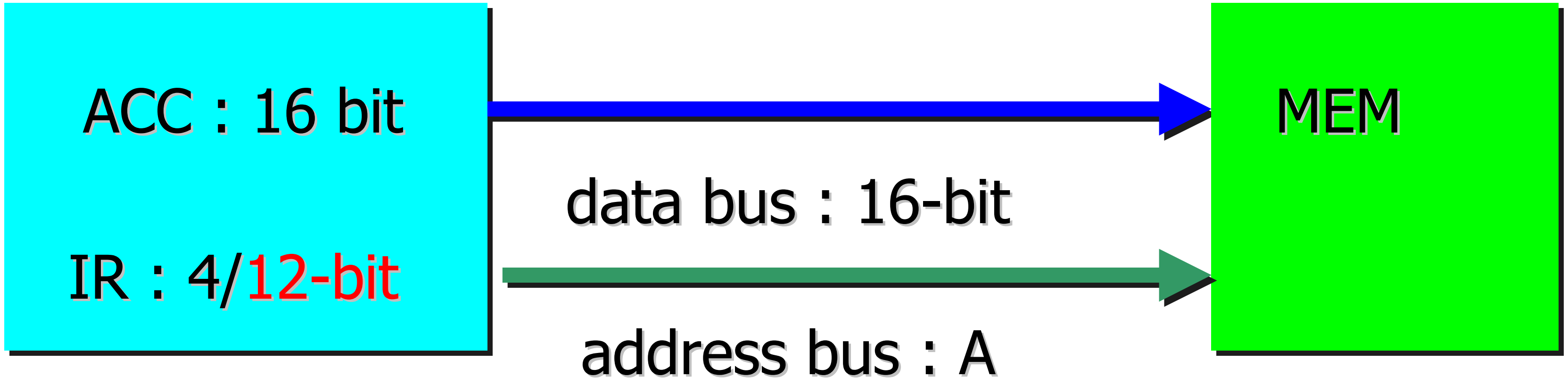
Execution : load instruction

instruction	opcode	function
LDA A	0000	ACC <= MEM(A)



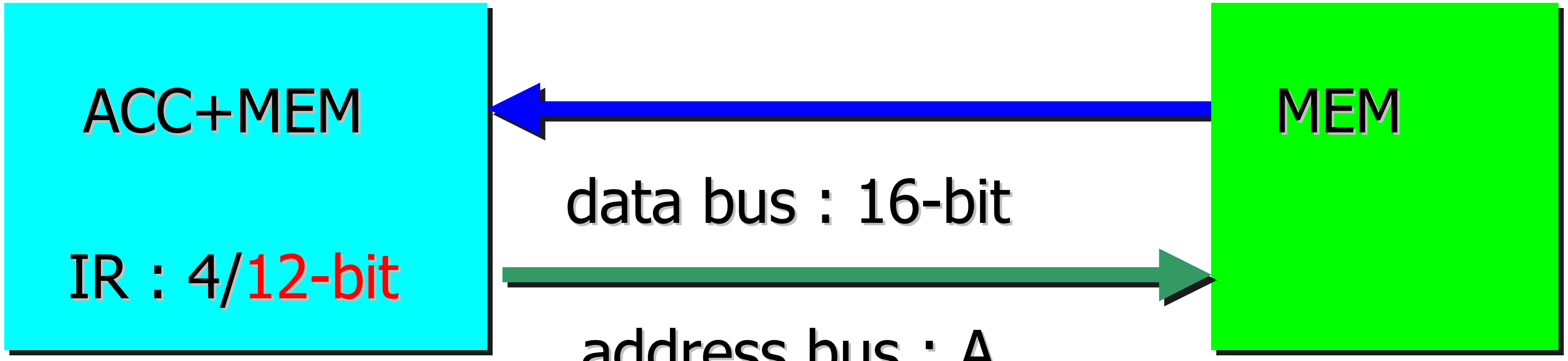
Execution : store instruction

instruction	opcode	function
STO A	0001	MEM(A) <= ACC



Execution : add instruction

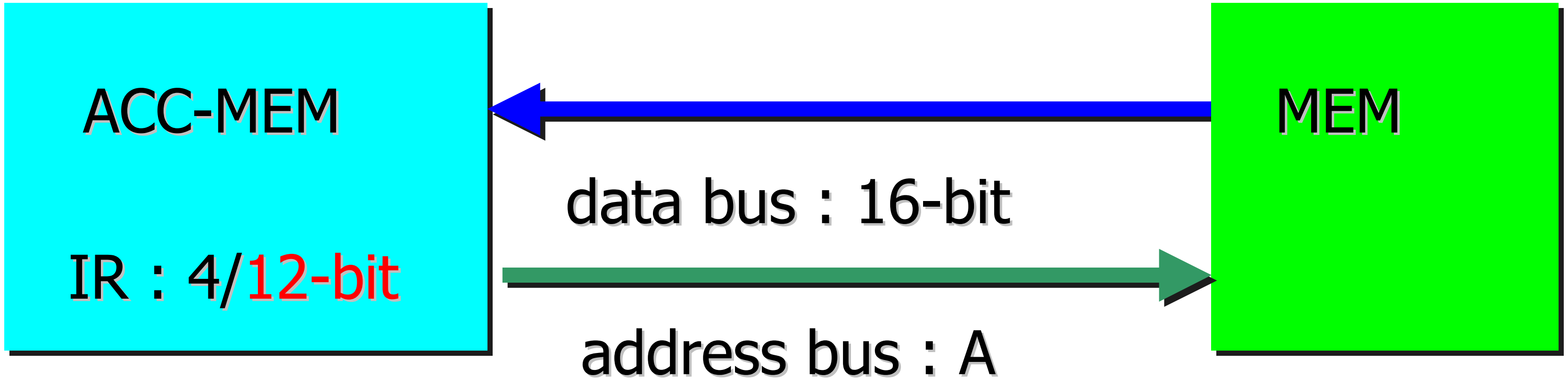
instruction	opcode	function
ADD A	0010	ACC <= ACC+MEM(A)



P. Bakowski

Execution : subtract instruction

instruction	opcode	function
ADD A	0011	ACC <= ACC-MEM(A)

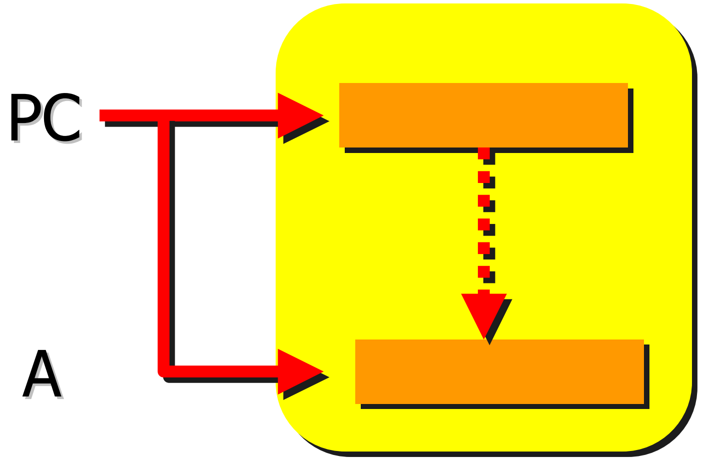


Execution : jump instruction

instruction	opcode	function
JMP A	0100	PC <= A

PC <= A

IR : 4/12-bit



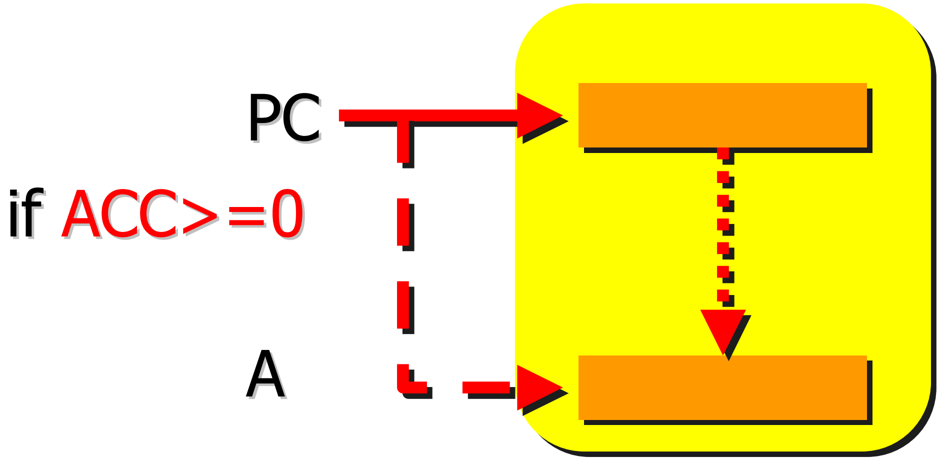
unconditional jump
to new address A

Jump if greater or equal instruction

instruction	opcode	function
JGE A	0101	PC <= A (if ACC>=0)

if ACC>=0
PC <= A

IR : 4/12-bit



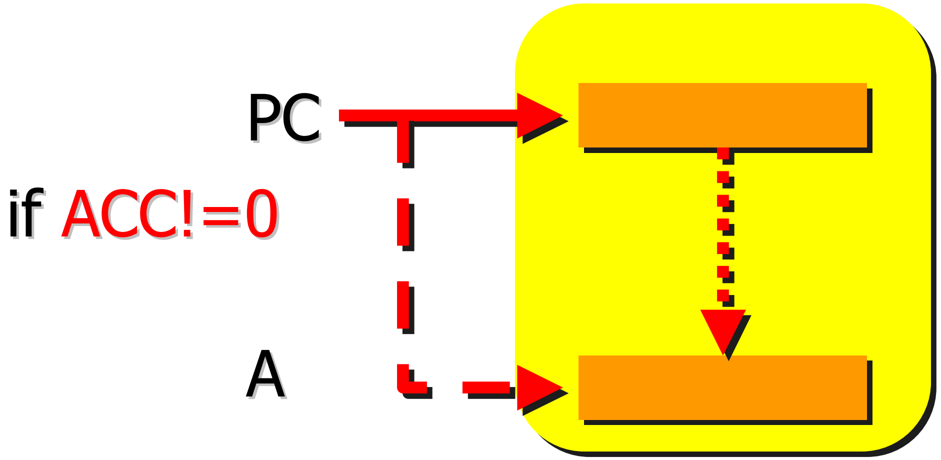
conditional jump to new address A

Jump if greater or equal instruction

instruction	opcode	function
JNE A	0110	PC <= A (if ACC!=0)

if ACC!=0
PC <= A

IR : 4/12-bit



conditional (if ACC not zero) jump to new address A

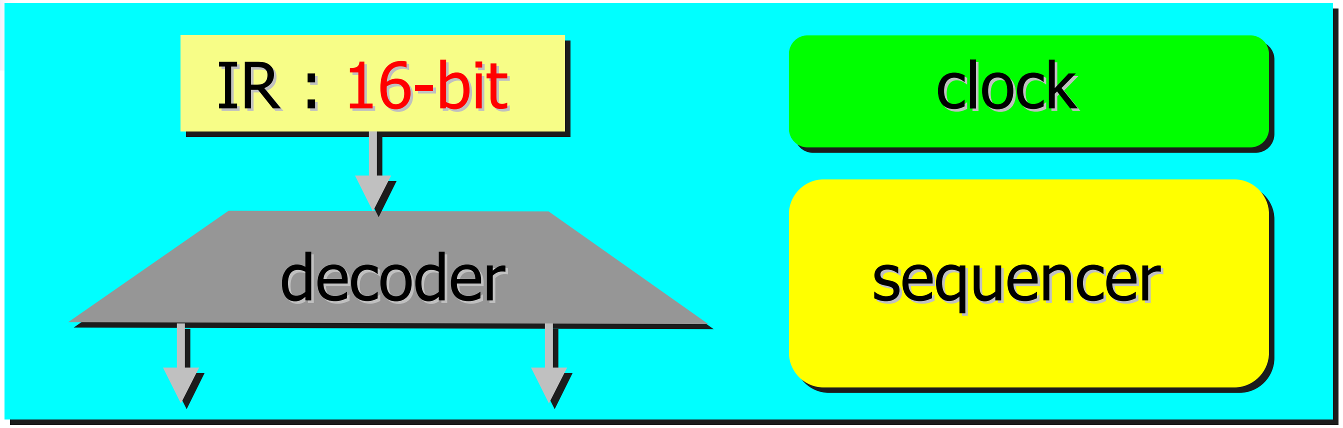
Execution : stop instruction

instruction	opcode	function
STP	0111	PC <= PC

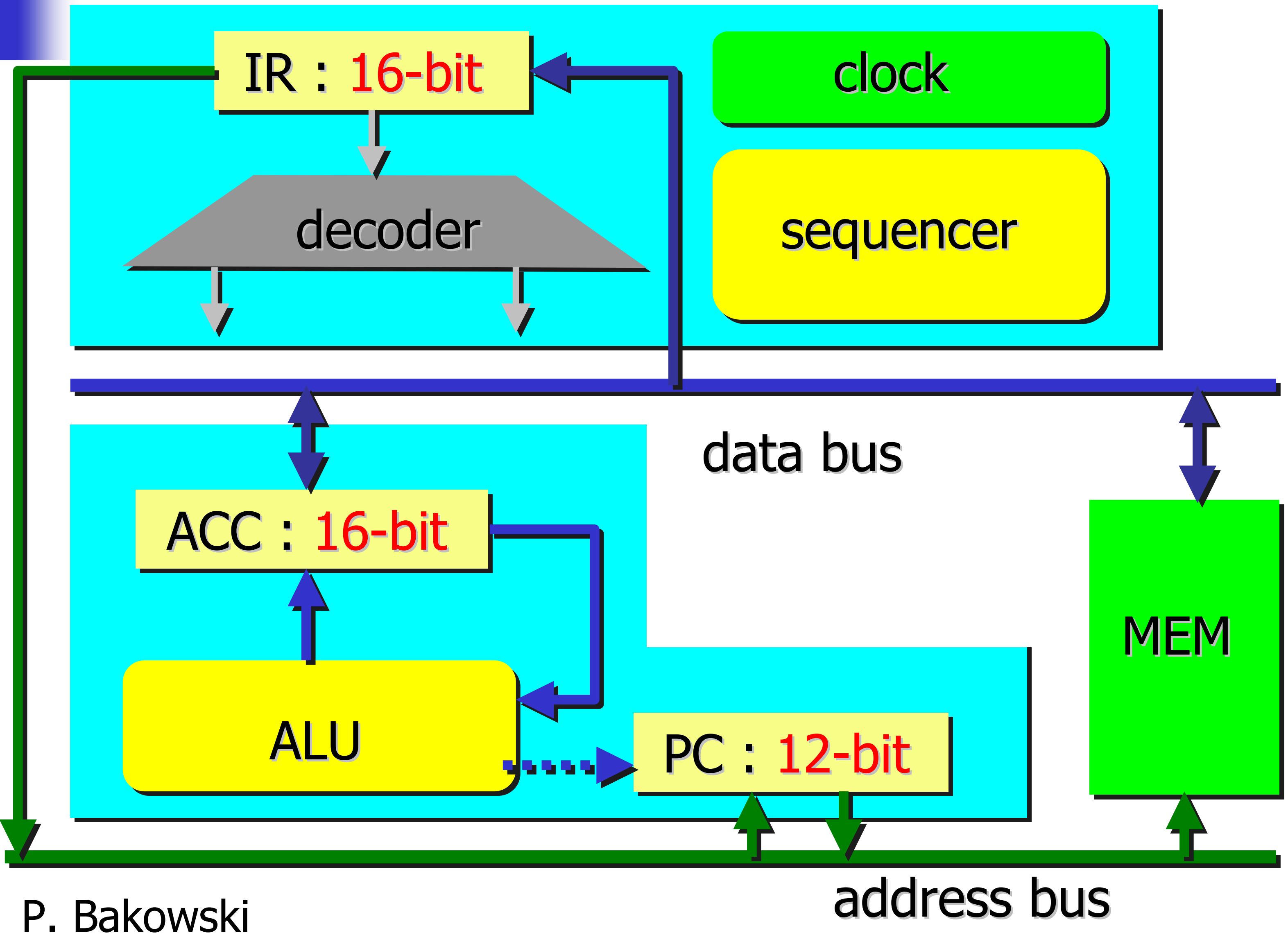
PC <= PC

IR : 4/12-bit

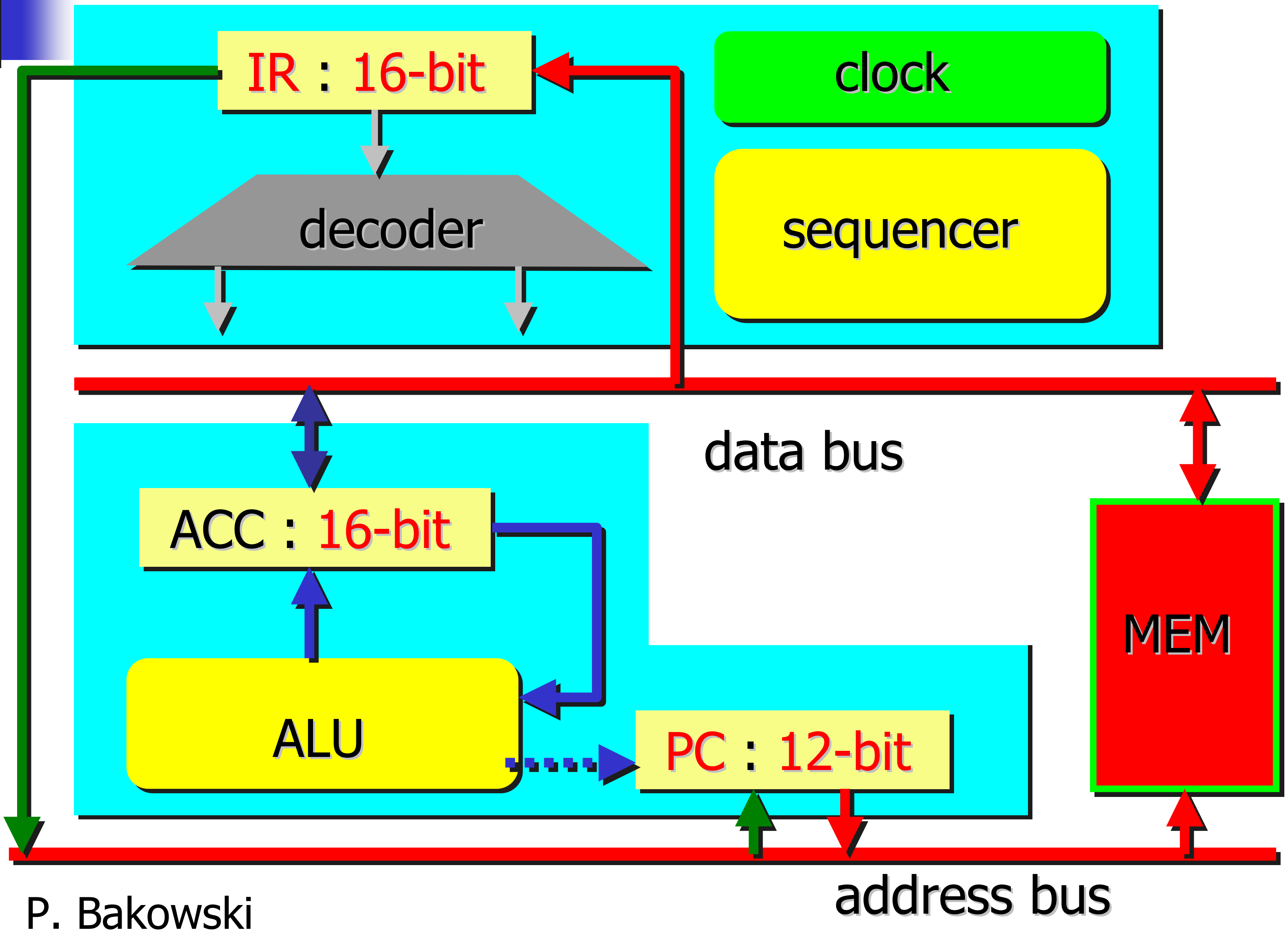
Control Path & Data Path



Control Path & Data Path

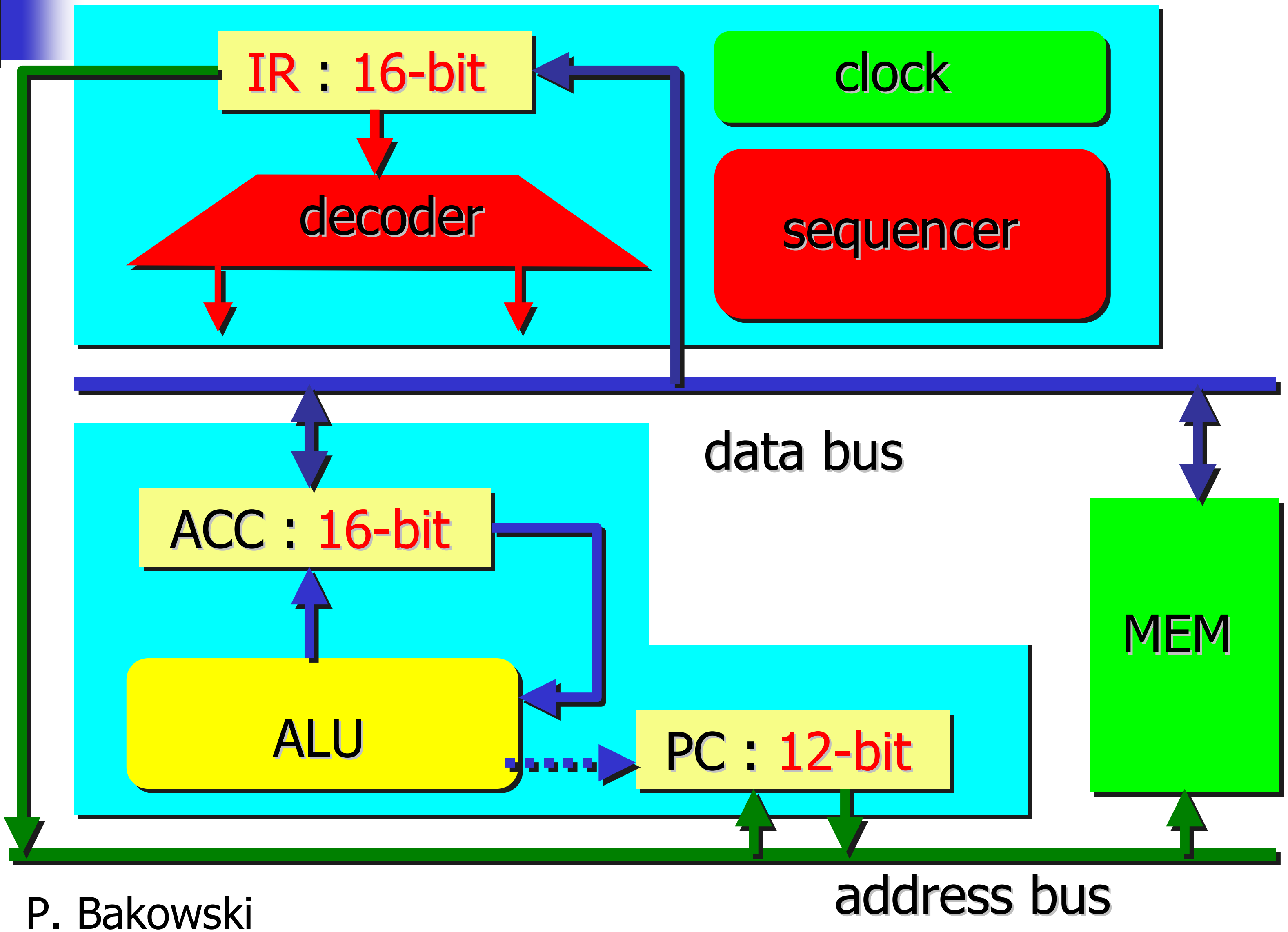


Instruction fetch – IR loaded

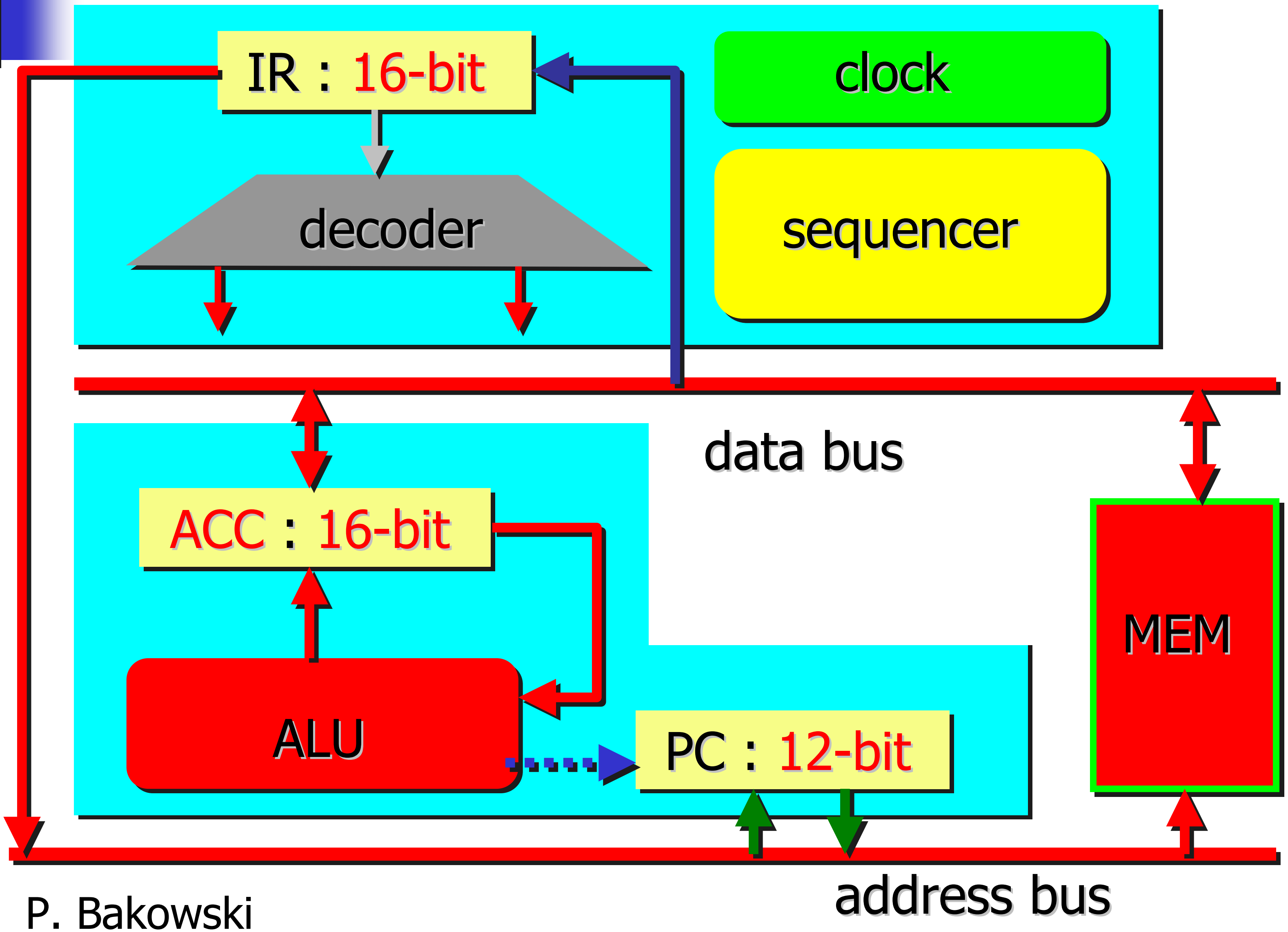


P. Bakowski

Instruction decode

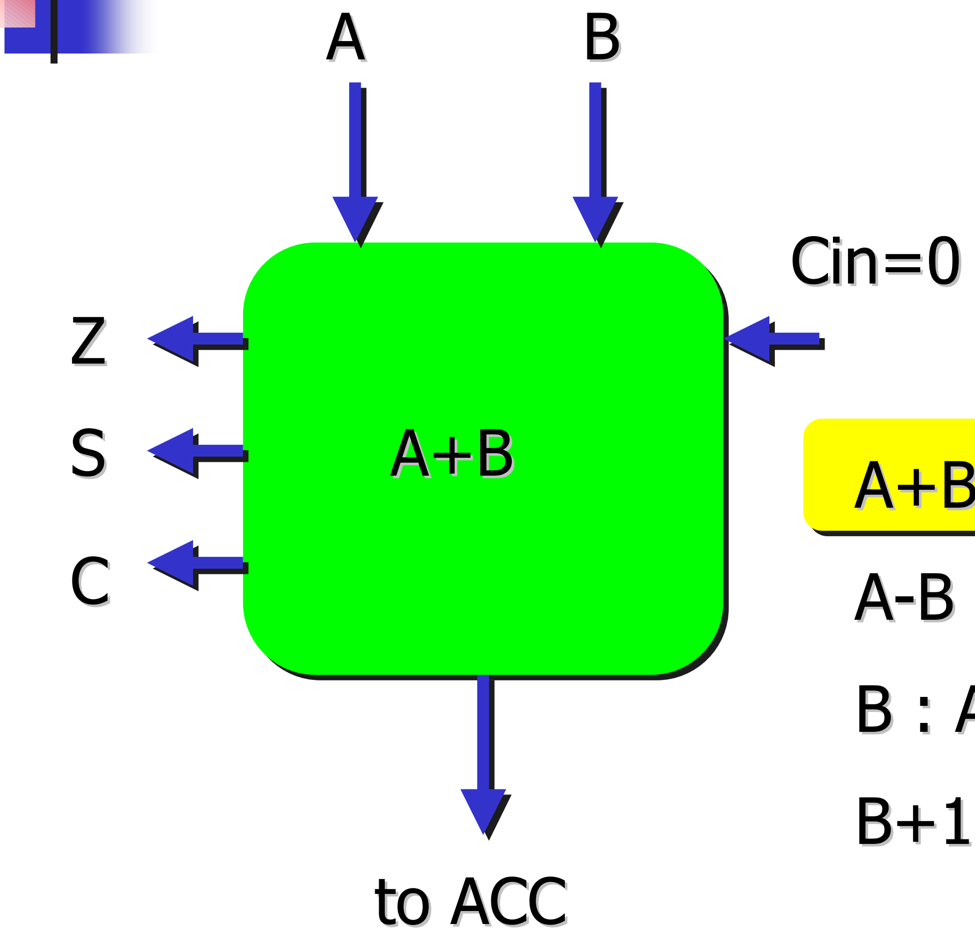


Instruction execute (add)



P. Bakowski

ALU design at RTL level



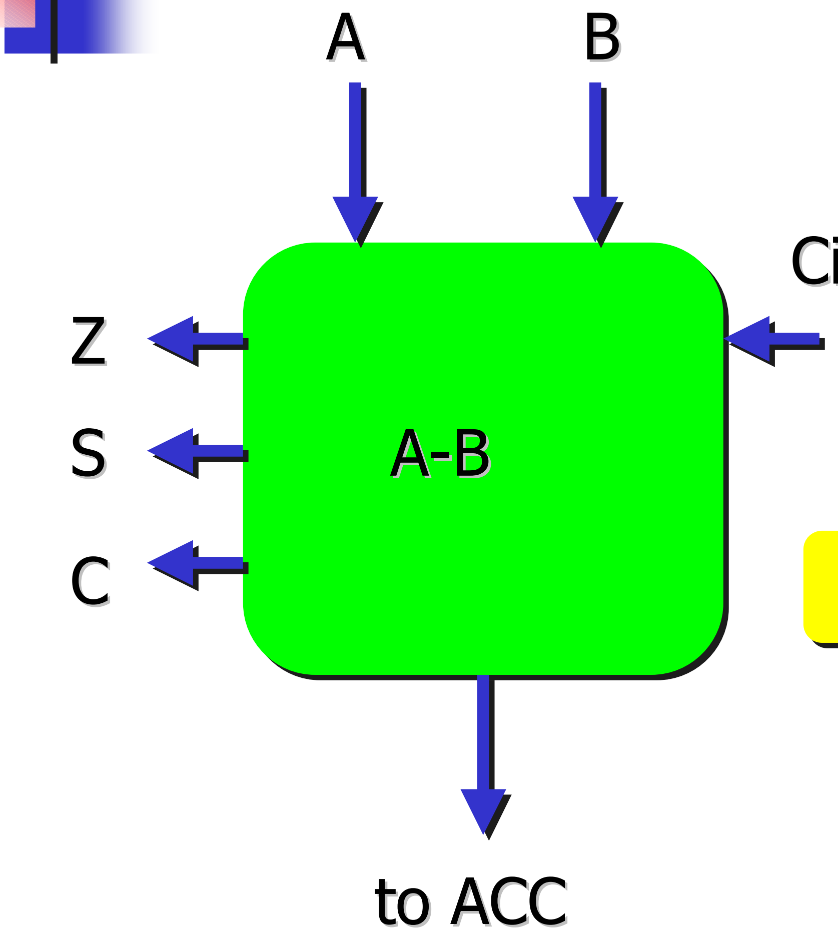
A+B : normal adder output

A-B : A+!B+1   (Cin=1)

B : A=0, Cin=0

B+1 : A=0, Cin=1

ALU design at RTL level



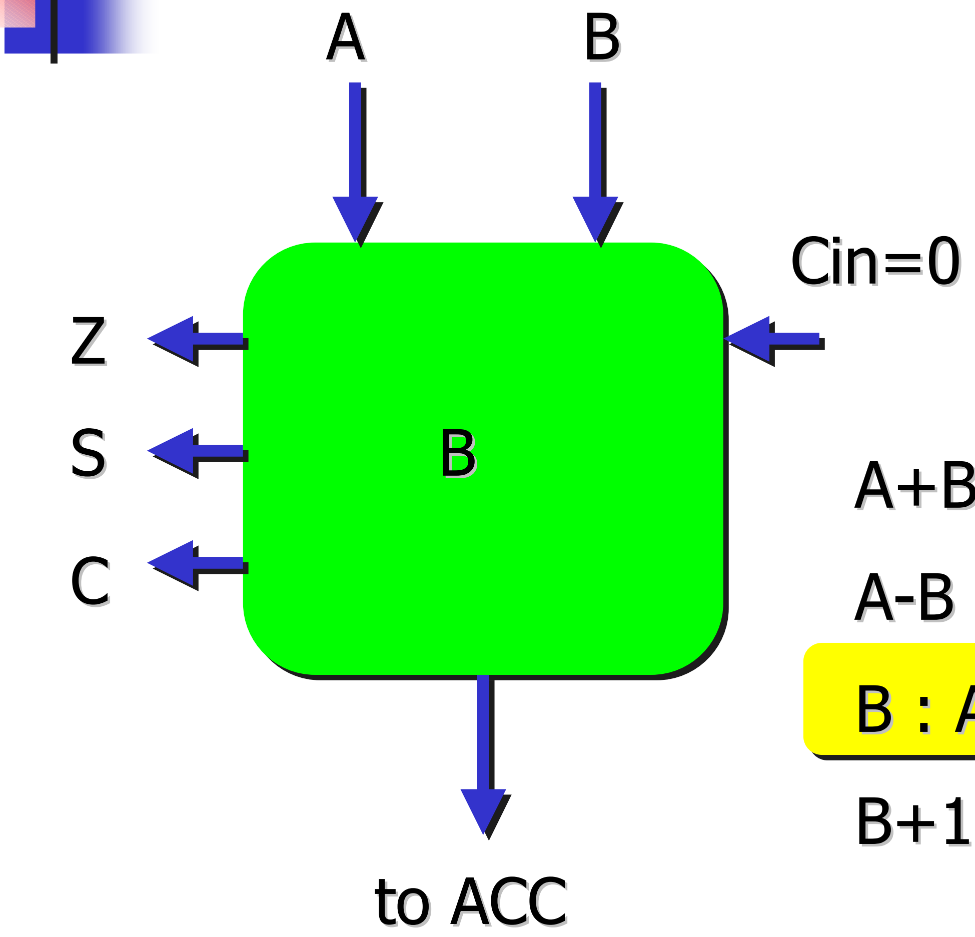
A+B : normal adder output

A-B : A+!B+1 (Cin=1)

B : A=0, Cin=0

B+1 : A=0, Cin=1

ALU design at RTL level



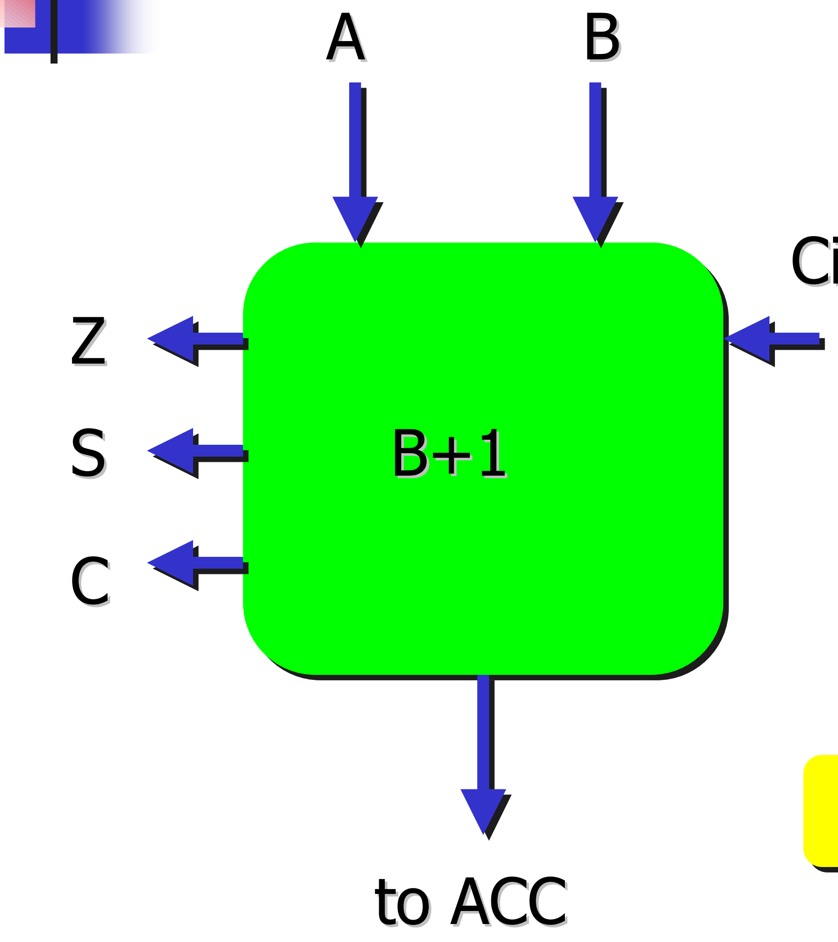
A+B : normal adder output

A-B : A+!B+1   (Cin=1)

B : A=0, Cin=0

B+1 : A=0, Cin=1

ALU design at RTL level



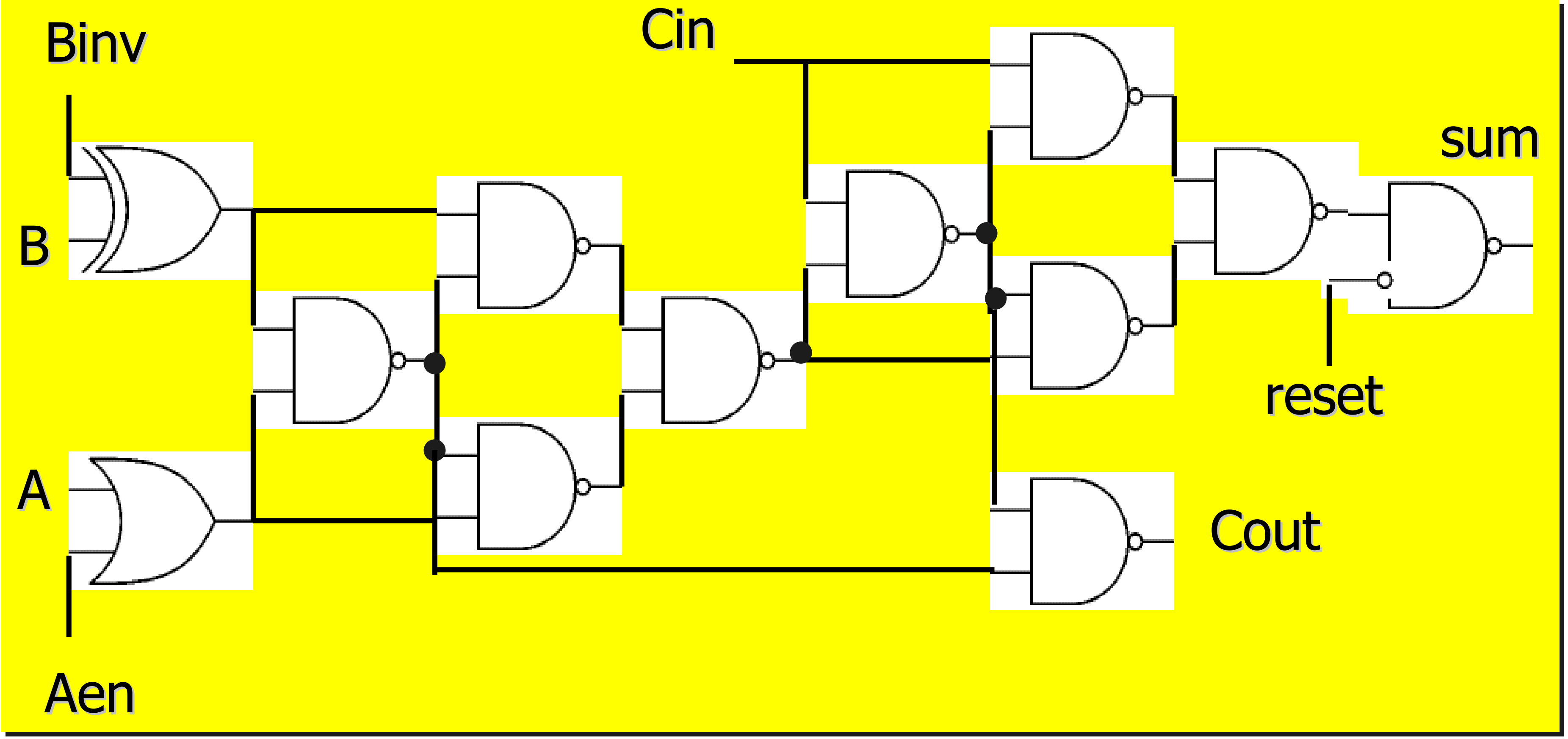
A+B : normal adder output

A-B : A+!B+1 (Cin=1)

B : A=0, Cin=0

B+1 : A=0, Cin=1

ALU design at logic level



one bit slice of ALU

High performance processor

- extending address space: 12 to 24 (32) bits
- adding address modes
- introducing stack for subprogram calls
- introducing register block
- introducing interruptions

High performance processor

- extending address space: 12 to 24 (32) bits
- adding new address modes
- introducing stack for subprogram calls
- introducing register block
- introducing interruptions

High performance processor

- extending address space: 12 to 24 (32) bits
- adding address modes
- introducing stack for subprogram calls
- introducing register block
- introducing interruptions

High performance processor

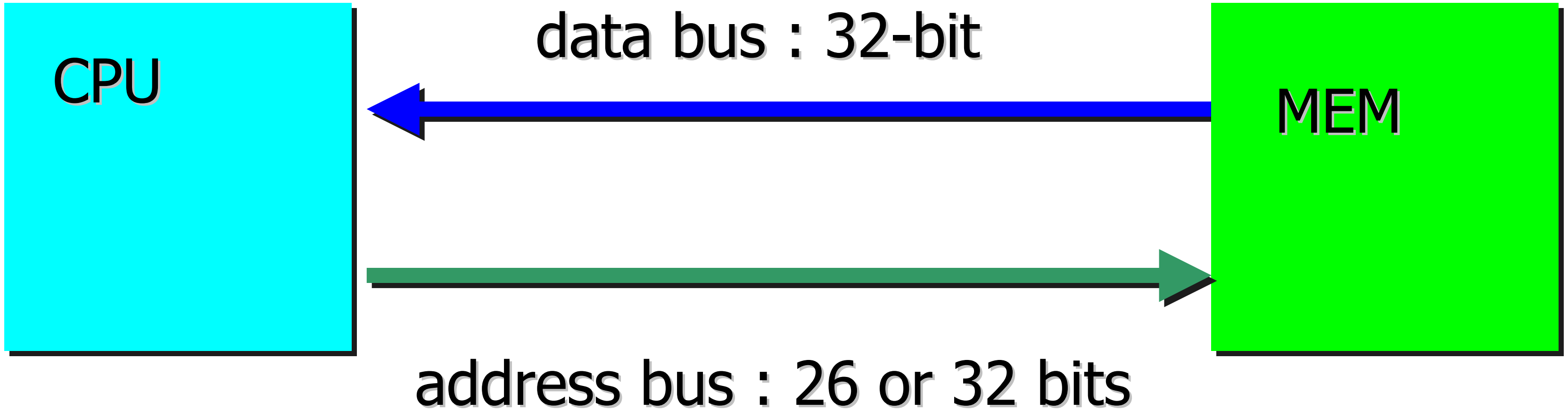
- extending address space: 12 to 24 (32) bits
- adding address modes
- introducing stack for subprogram calls
- introducing register block
- introducing interruptions

High performance processor

- extending address space: 12 to 24 (32) bits
- adding address modes
- introducing stack for subprogram calls
- introducing register block
- introducing interruptions

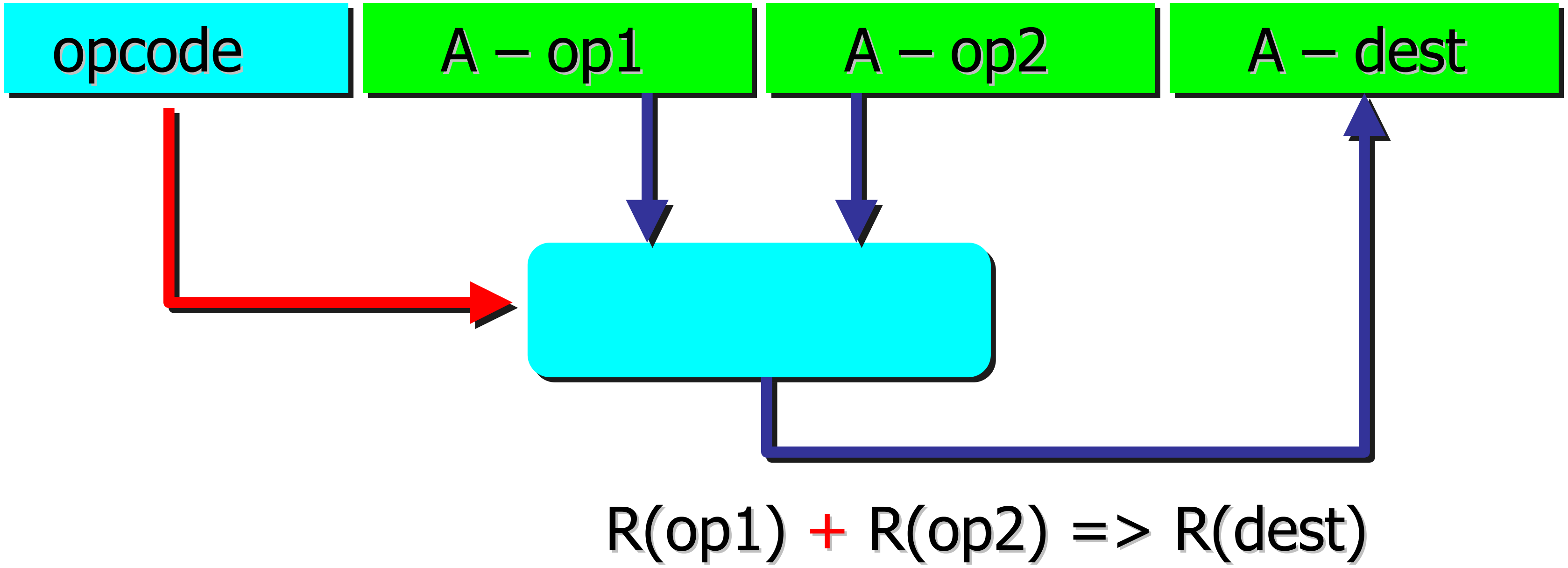
High performance processor

- extending address space: 12 to 24 (32) bits
- adding new address modes



High performance processor

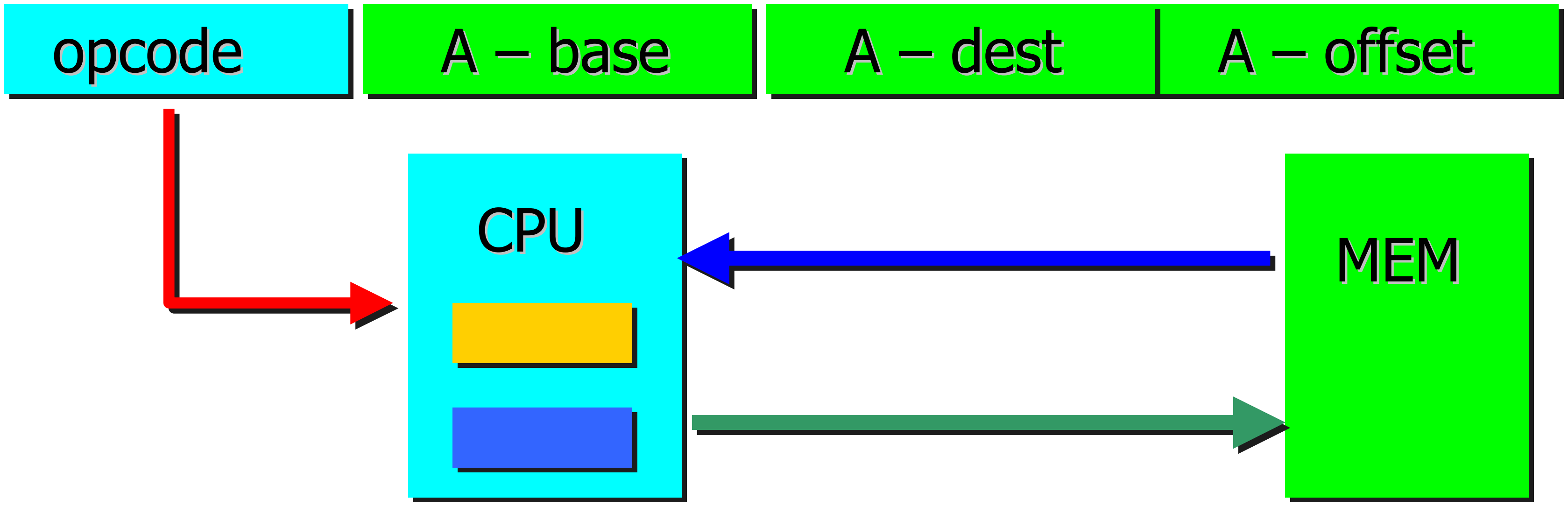
- extending address space: 12 to 24 (32) bits
- adding new address modes



P. Bakowski

High performance processor

- extending address space: 12 to 24 (32) bits
- adding new address modes



MEM[R(base) + offset] => R(dest)

Instruction types

- data movement: load and store
- data processing: logic and arithmetic
- control flow: jump, conditional jump, call, return, ..
- state instructions: execution mode, interruption and memory control

Instruction types

- data movement: load and store
- data processing: logic and arithmetic
- control flow: jump, conditional jump, call, return, ..
- state instructions: execution mode, interruption and memory control

Instruction types

- data movement: load and store
- data processing: logic and arithmetic
- control flow: jump, conditional jump, call, return, ..
- state instructions: execution mode, interruption and memory control

Instruction types

- data movement: load and store
- data processing: logic and arithmetic
- control flow: jump, conditional jump, call, return, ..
- state instructions: execution mode, interruption and memory control, ..

Orthogonal instruction types

- instruction type is a set of similar instructions: e.g. add, subtract, .. with similar addressing schemes

- different instruction types are executed via different architectural blocs

- the use of separate architectural blocs allows for independent execution – concurrent execution

Orthogonal instruction types

- instruction type is a set of similar instructions: e.g. add, subtract, .. with similar addressing schemes
- different instruction types are executed via different architectural blocs
- the use of separate architectural blocs allows for independent execution – concurrent execution

Orthogonal instruction types

- instruction type is a set of similar instructions: e.g. add, subtract, .. with similar addressing schemes

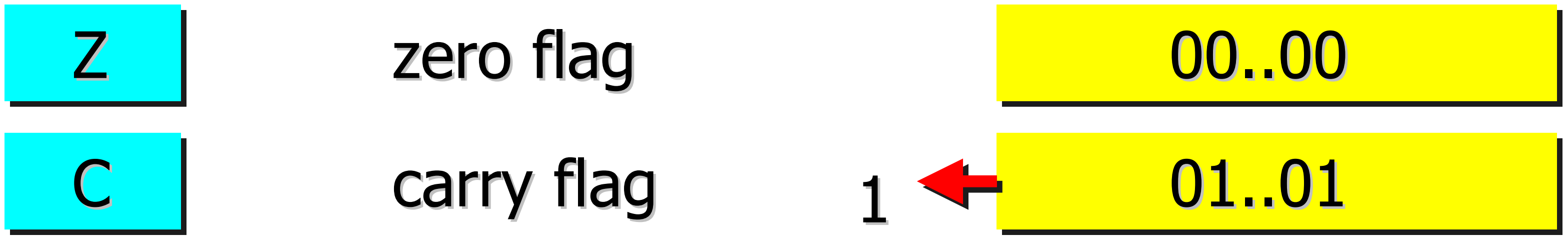
- different instruction types are executed via different architectural blocs

- the use of separate architectural blocs allows for independent execution – concurrent execution

Condition-state code register

Z	zero flag	00..00

Condition-state code register



Z

zero flag

00..00

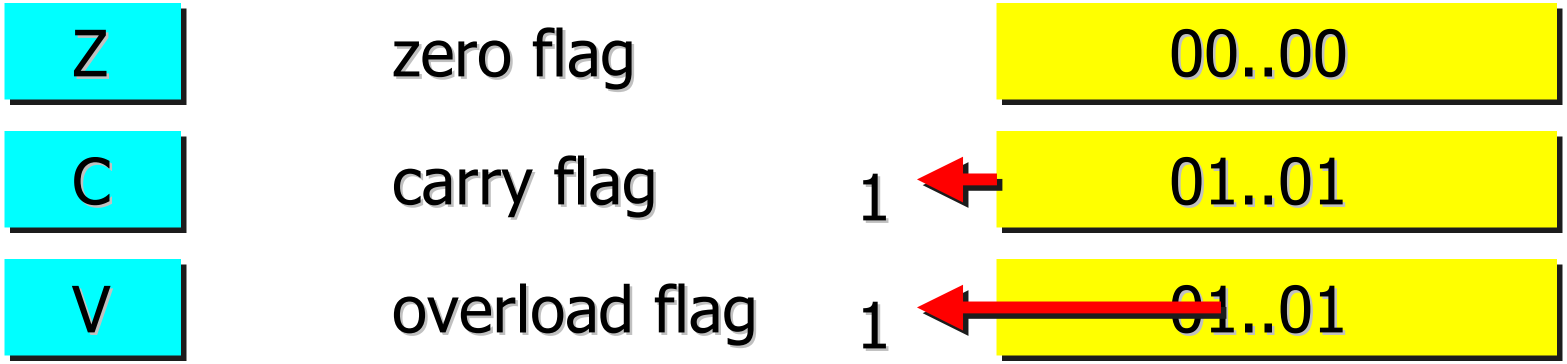
C

carry flag

1

01..01

Condition-state code register

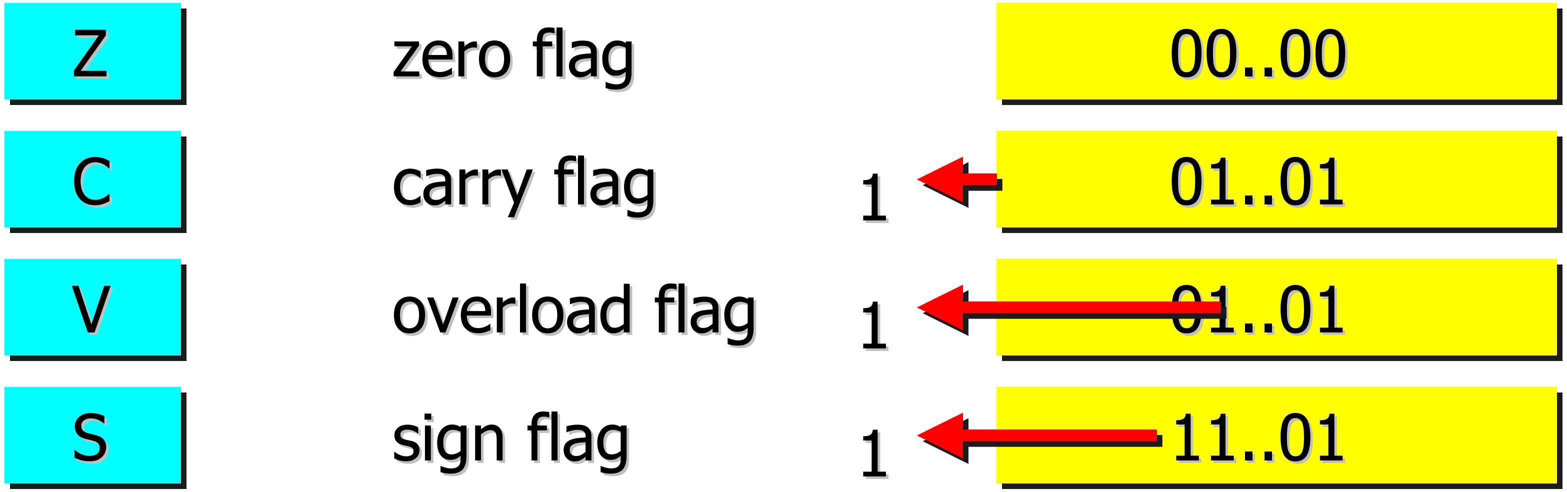


Z zero flag 00..00

C carry flag 1 ← 01..01

V overload flag 1 ← 01..01

Condition-state code register



Z zero flag 00..00

C carry flag 1 01..01

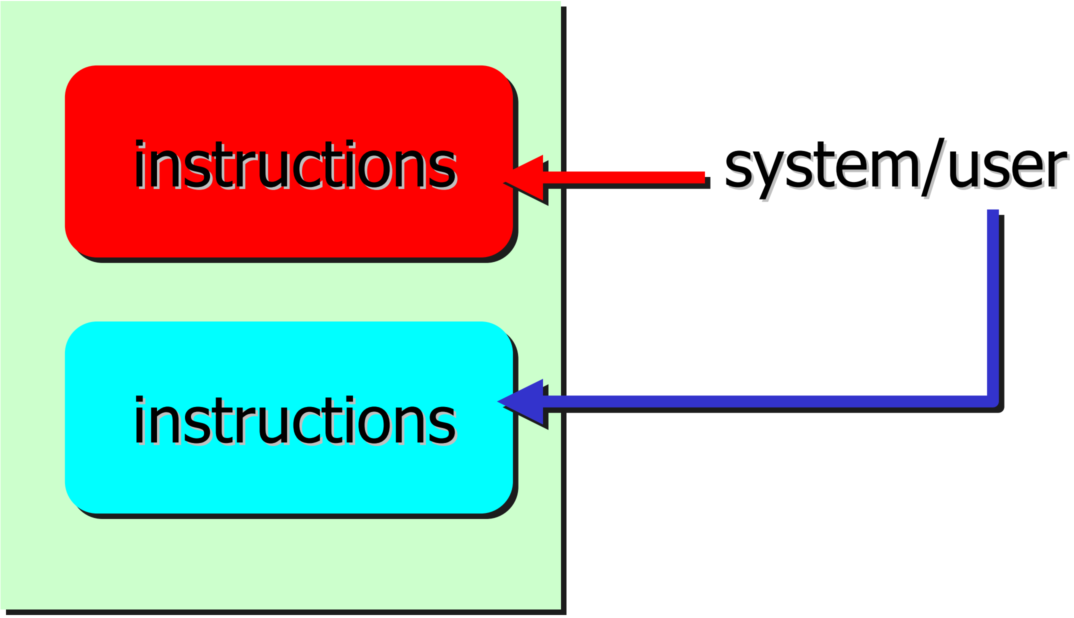
V overload flag 1 01..01

S sign flag 1 11..01

P. Bakowski

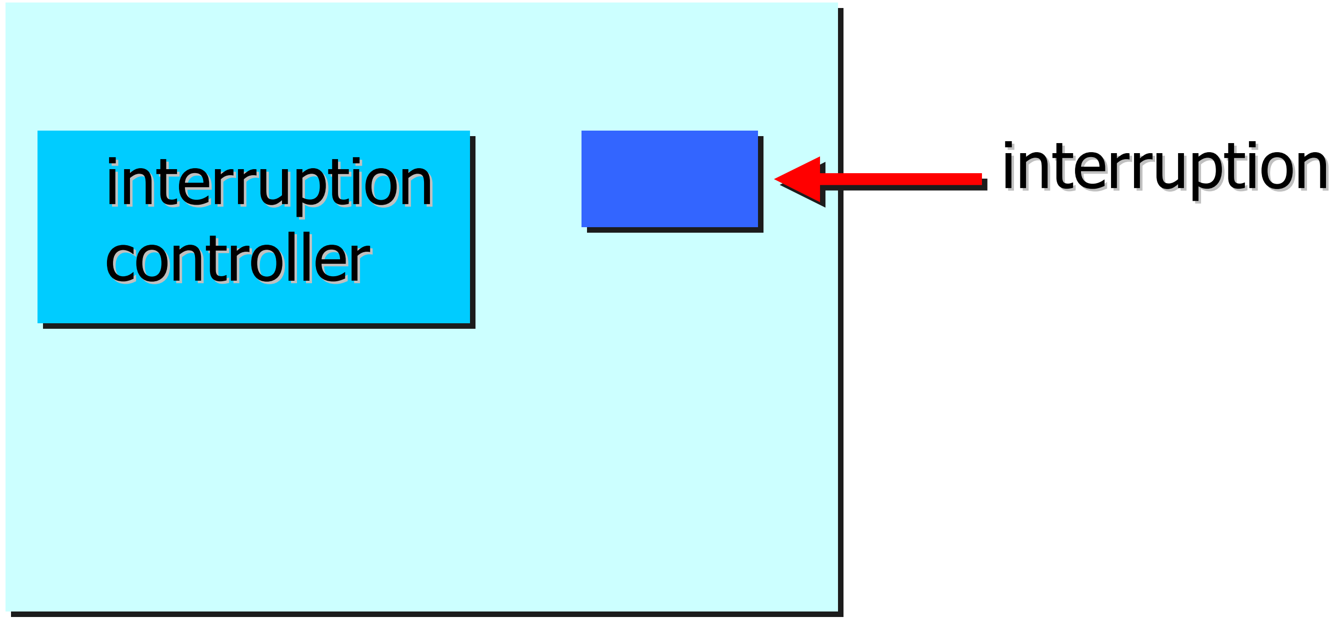
Condition-state code register

M mode flag [0,1]



Condition-state code register

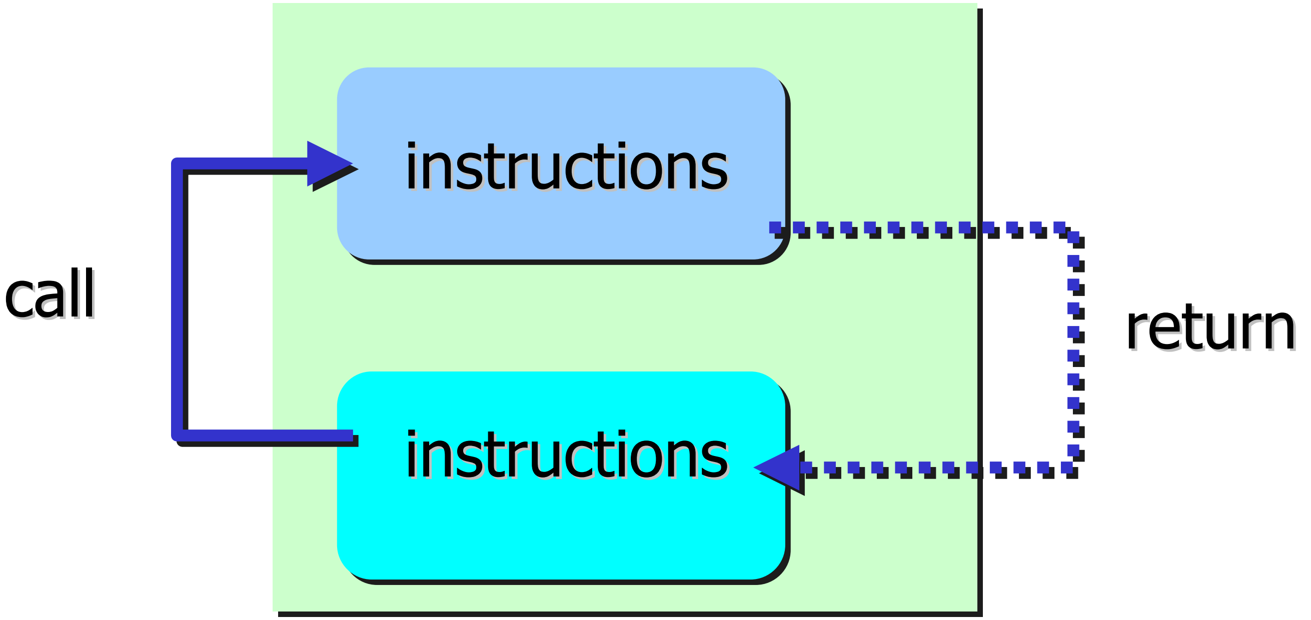
I   interruption flag [0,1]



Subprograms and system calls

M

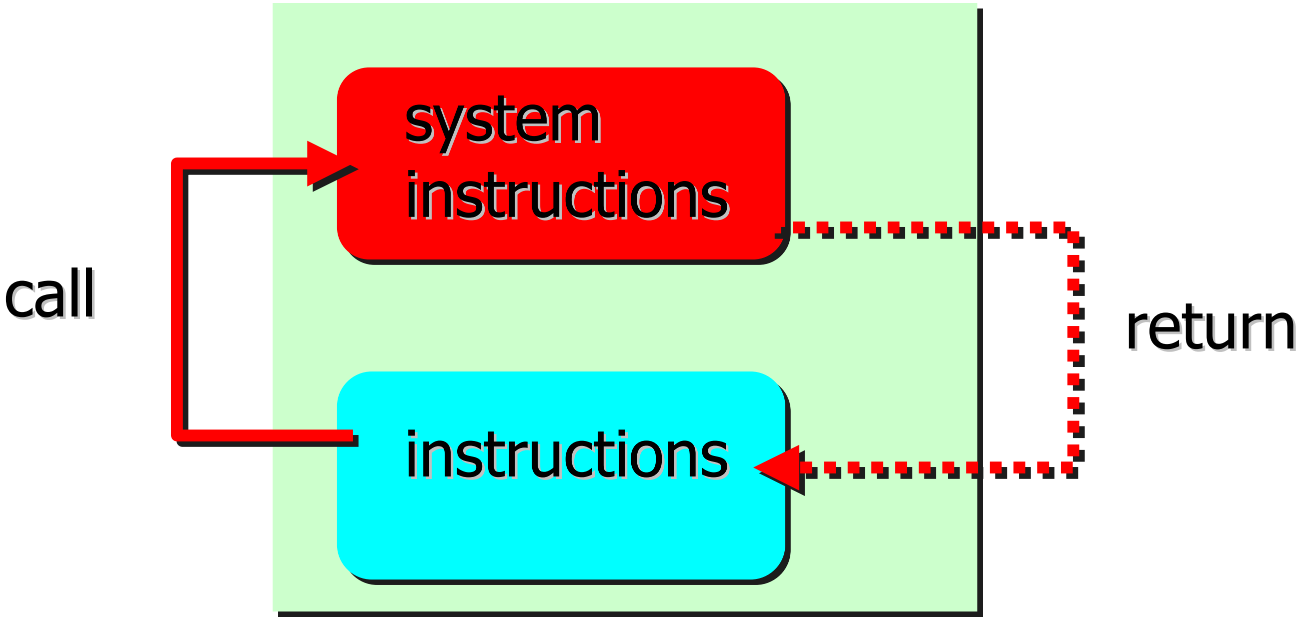
user state/user call



Subprograms and system calls

M

system state/system call



The RISC concept

Reduced Instruction Set Computer

- data movement - 45%
- control flow – 22%
- arithmetic operations – 14%
- comparisons – 13%
- logic operations – 5%
- other – 1%

The RISC concept

Reduced Instruction Set Computer

- data movement   - 45%
- control flow – 22%
- arithmetic operations – 14%
- comparisons – 13%
- logic operations – 5%
- other – 1%

The RISC concept

Reduced Instruction Set Computer

- data movement - 45%
- control flow – 22%
- arithmetic operations – 14%
- comparisons – 13%
- logic operations – 5%
- other – 1%

The RISC concept

Reduced Instruction Set Computer

- data movement - 45%
- control flow – 22%
- arithmetic operations – 14%
- comparisons – 13%
- logic operations – 5%
- other – 1%

The RISC concept

Reduced Instruction Set Computer

- data movement - 45%
- control flow – 22%
- arithmetic operations – 14%
- comparisons – 13%
- logic operations – 5%
- other – 1%

The RISC concept

Reduced Instruction Set Computer

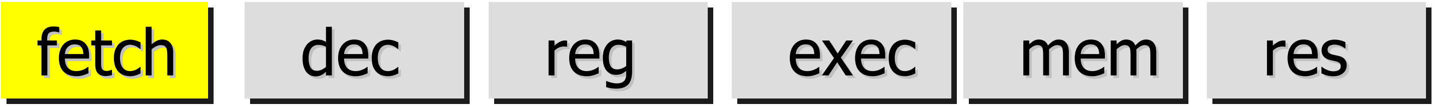
- data movement - 45%
- control flow – 22%
- arithmetic operations – 14%
- comparisons – 13%
- logic operations – 5%
- other – 1%

P. Bakowski

The RISC concept - pipelines

Instruction elaboration stages:

- instruction fetch

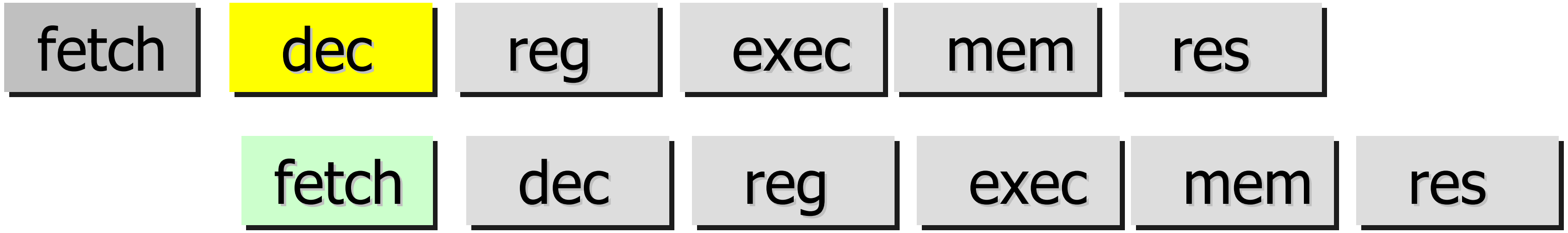


fetch dec reg exec mem res

The RISC concept - pipelines

Instruction elaboration stages:

- instruction fetch
- decode

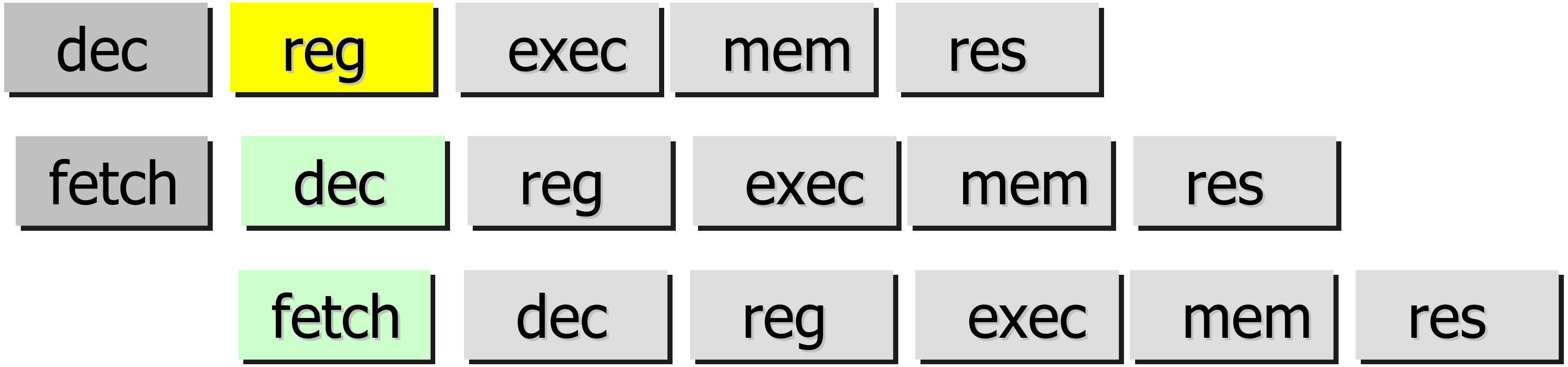


fetch dec reg exec mem res

fetch dec reg exec mem res

The RISC concept - pipelines

- instruction fetch
- decode
- read operands



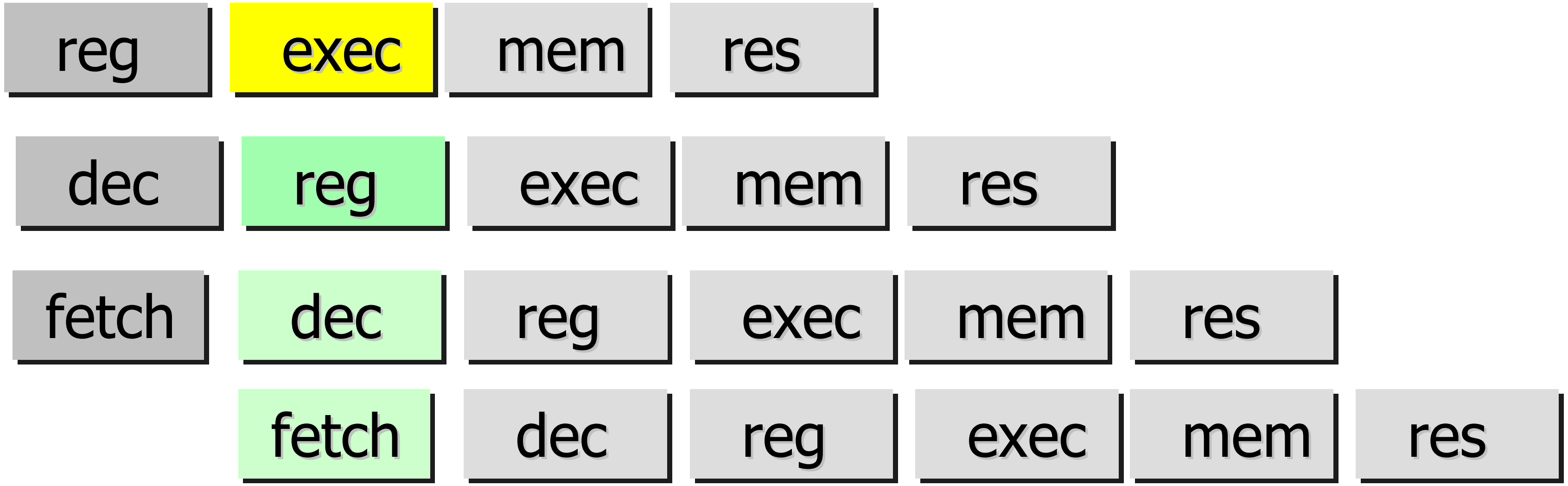
dec reg exec mem res

fetch dec reg exec mem res

fetch dec reg exec mem res

The RISC concept - pipelines

- execute/ calculate memory address

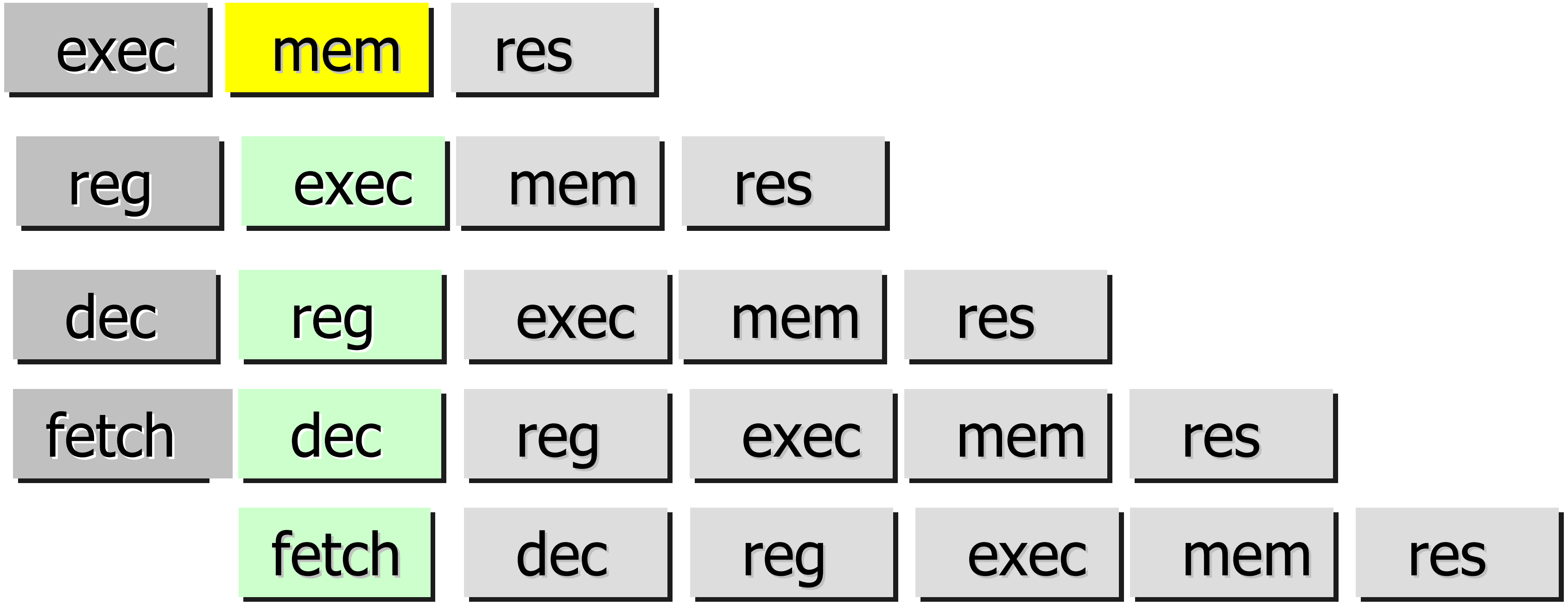


reg	exec	mem	res			
dec	reg	exec	mem	res		
fetch	dec	reg	exec	mem	res	
	fetch	dec	reg	exec	mem	res

P. Bakowski

The RISC concept - pipelines

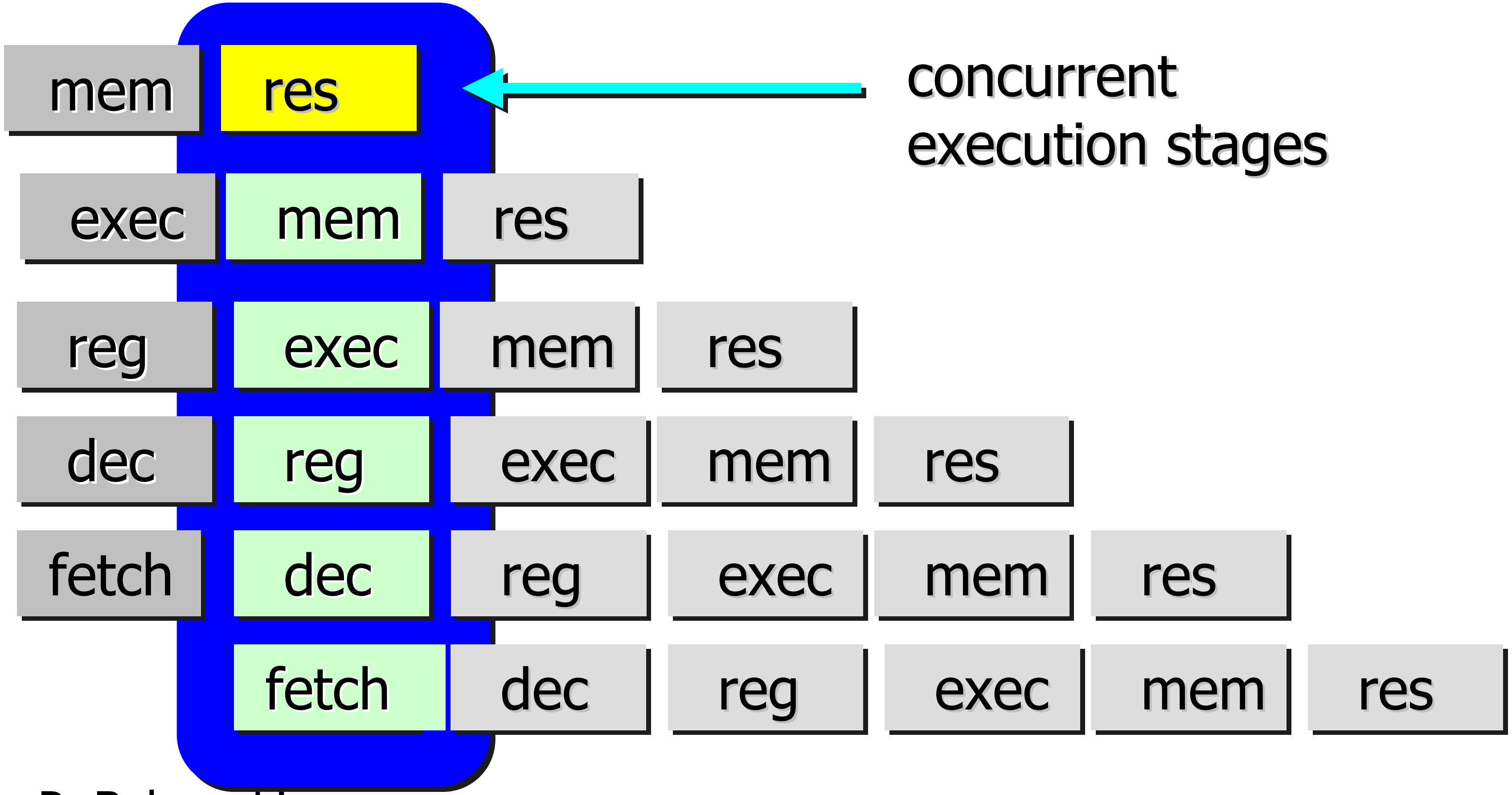
- read-memory memory



exec	mem	res				
reg	exec	mem	res			
dec	reg	exec	mem	res		
fetch	dec	reg	exec	mem	res	
	fetch	dec	reg	exec	mem	res

The RISC concept - pipelines

- write the result

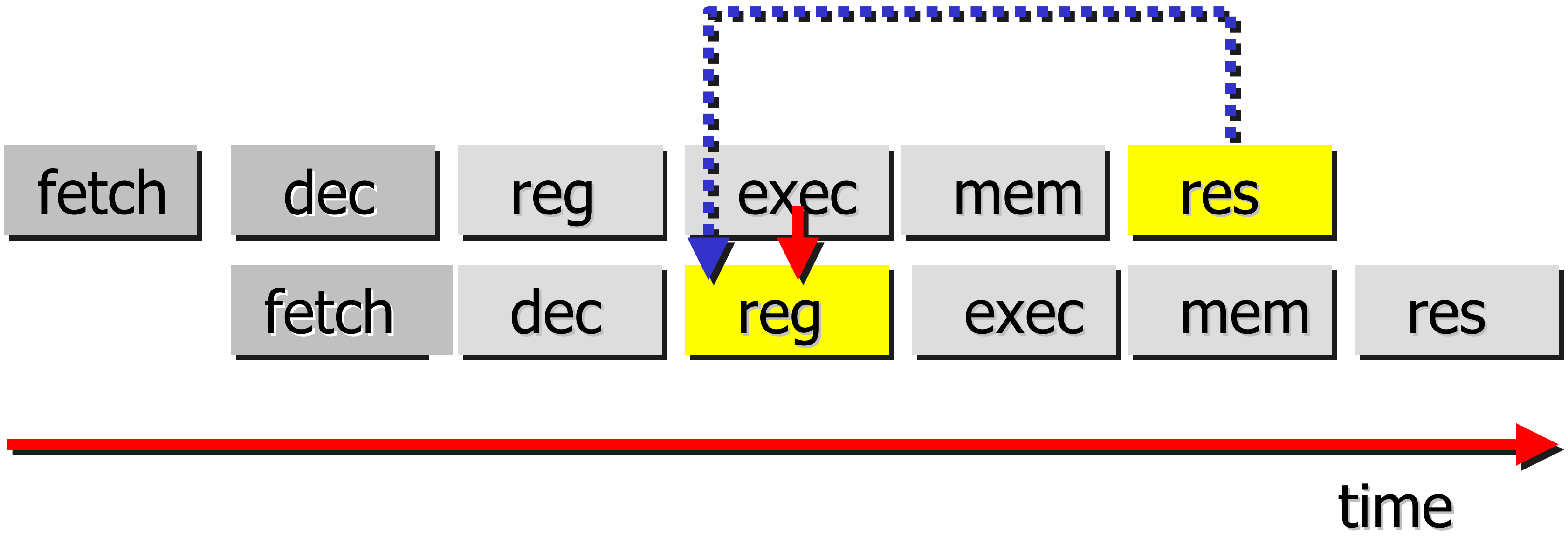


concurrent
execution stages

mem	res					
exec	mem	res				
reg	exec	mem	res			
dec	reg	exec	mem	res		
fetch	dec	reg	exec	mem	res	
	fetch	dec	reg	exec	mem	res

Pipeline hazards

- read after write - bypass
- jump instructions – sequence
- memory waits – stalls



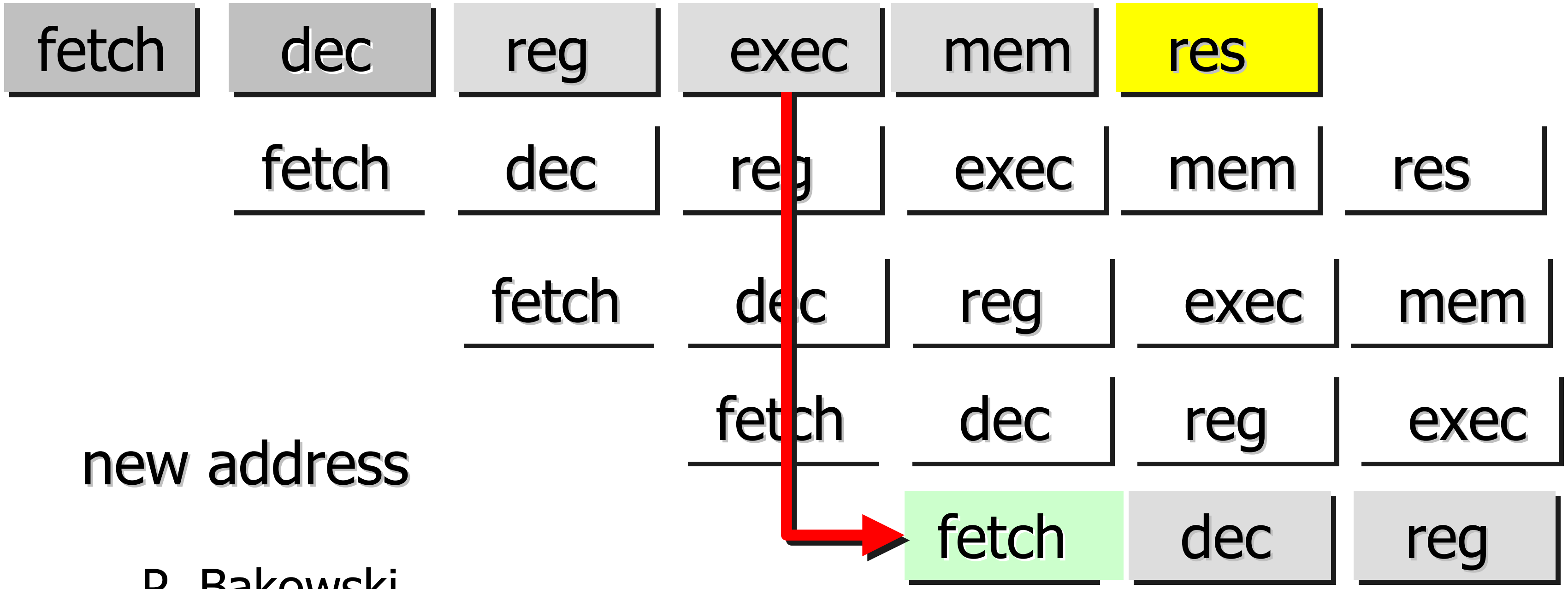
fetch dec reg exec mem res

fetch dec reg exec mem res

time

Pipeline hazards

- read after write - bypass
- jump instructions – sequence
- memory waits – stalls



fetch dec reg exec mem res

fetch dec reg exec mem res

fetch dec reg exec mem

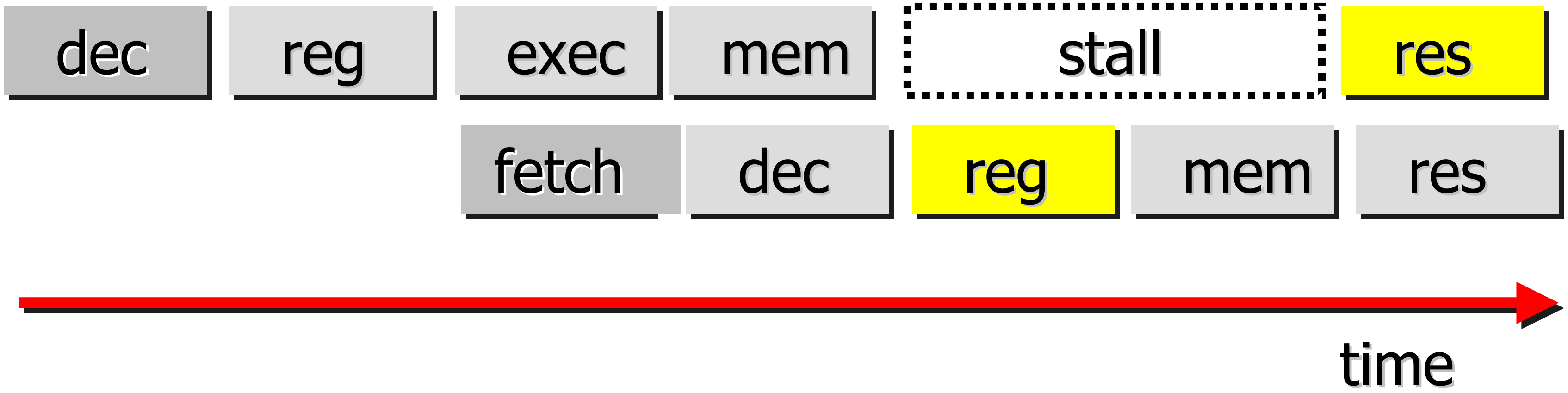
fetch dec reg exec

new address

fetch dec reg

Pipeline hazards

- read after write - bypass
- jump instructions – sequence
- memory waits – stalls



dec reg exec mem stall res

fetch dec reg mem res

time

Risc architecture (basics)

- a fixed 32-bit instruction/word size
- load-store architecture where calculation instructions operate only on registers
- large register bank of 32 32-bit registers

time

Risc architecture (basics)

- a fixed 32-bit instruction/word size
- load-store architecture where calculation instructions operate only on registers
- large register bank of 32 32-bit registers

time

Risc architecture (basics)

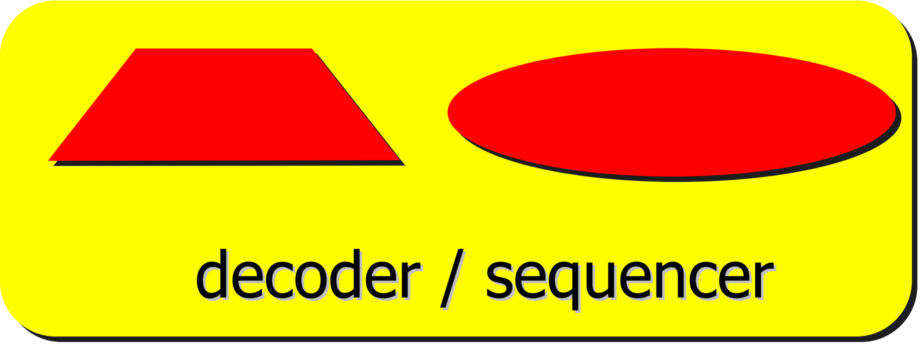
- a fixed 32-bit instruction/word size
- load-store architecture where calculation instructions operate only on registers
- large register bank of 32 32-bit registers

time

Risc organization (basics)

- hard-wired instruction decode logic
- pipelined execution
- single-cycle execution (throughput)

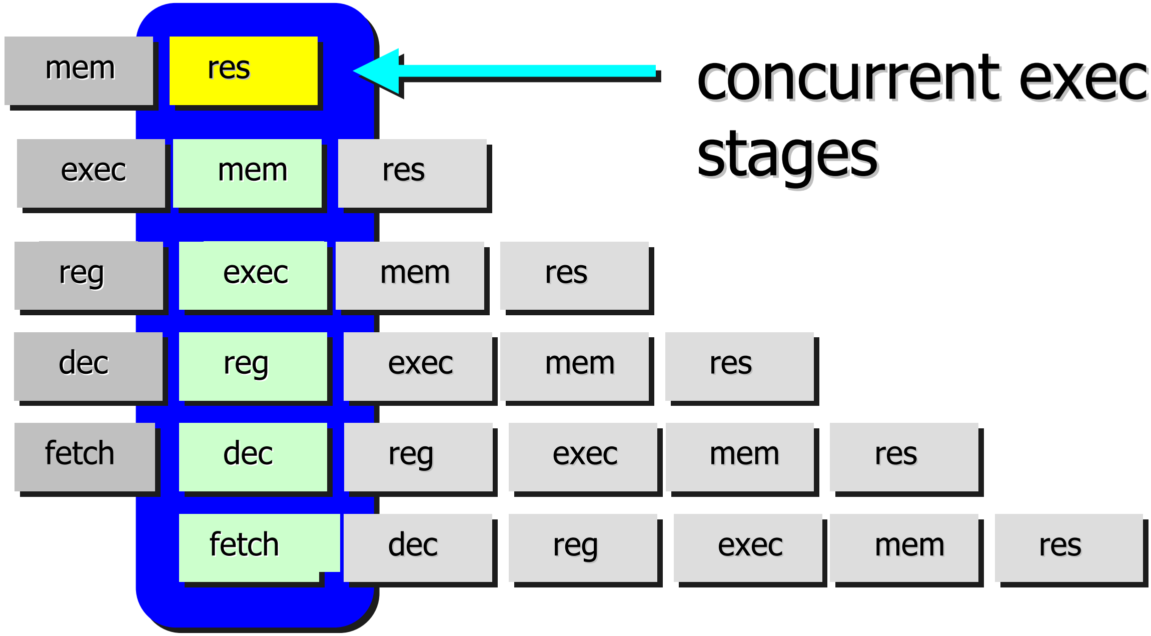
opcode	A – base	A – dest	A – offset



decoder / sequencer

Risc organization (basics)

- hard-wired instruction decode logic
- pipelined execution
- single-cycle execution (throughput)



mem res

exec mem res

reg exec mem res

dec reg exec mem res

fetch dec reg exec mem res

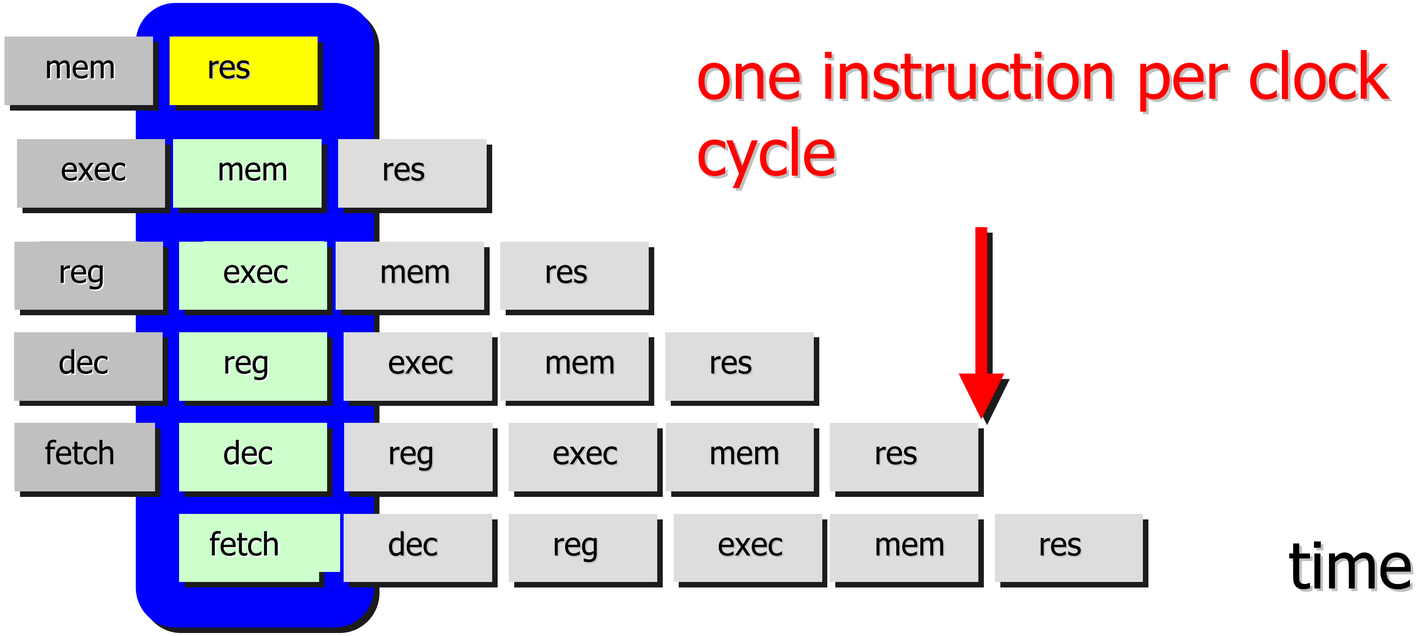
fetch dec reg exec mem res

concurrent execution stages

time

Risc organization (basics)

- hard-wired instruction decode logic
- pipelined execution
- single-cycle execution (throughput)



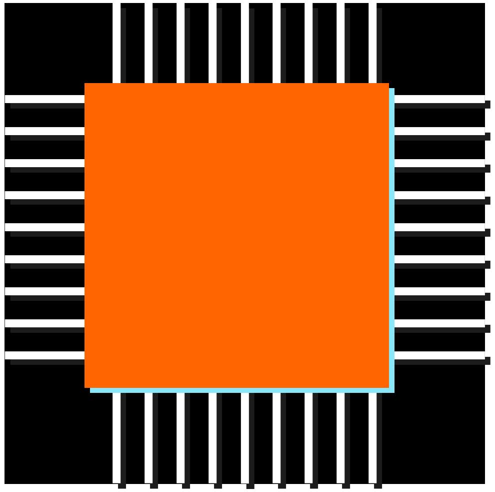
mem	res					
exec	mem	res				
reg	exec	mem	res			
dec	reg	exec	mem	res		
fetch	dec	reg	exec	mem	res	
	fetch	dec	reg	exec	mem	res

one instruction per clock cycle

time

Risc advantages

- small die size
- short development time
- high performance



regular structure

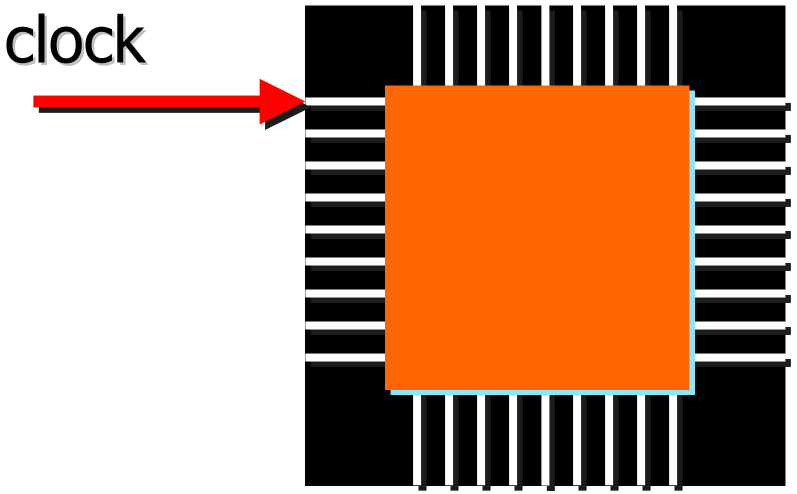
Risc advantages

- small die size
- short development time
- high performance

simple structure

Risc advantages

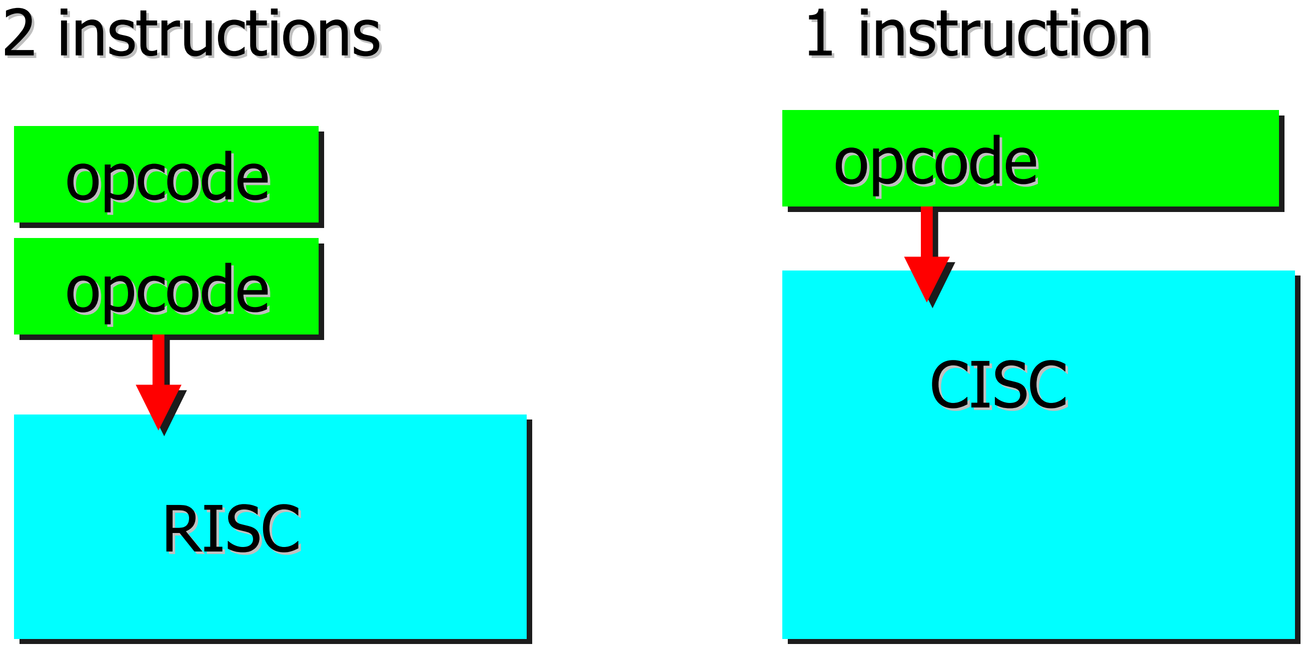
- small die size
- short development time
- high performance



fast clock –
simple pipeline
stages

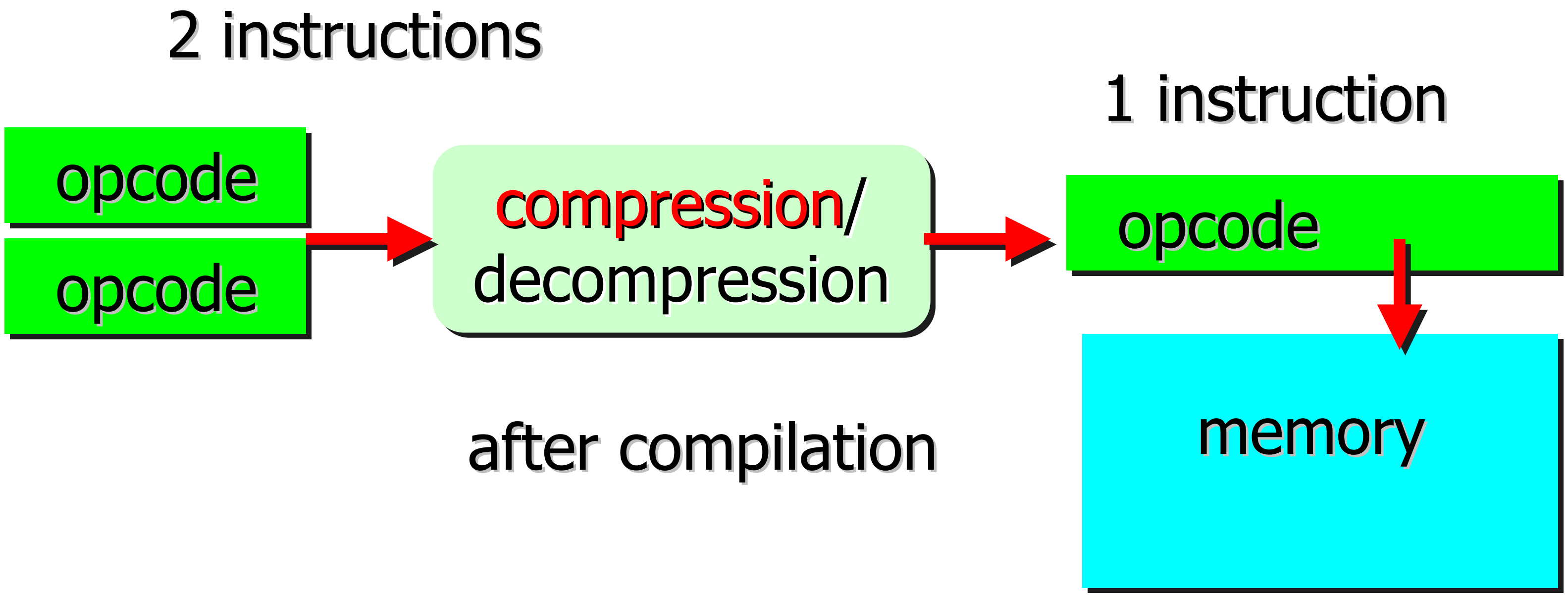
Risc drawback

Main drawback of RISC architecture is lower instruction code density than in CISC architectures



Risc drawback

The solution to this problem is code compression/decompresion mechanism (ARM)



2 instructions

opcode

opcode

compression/ decompression

after compilation

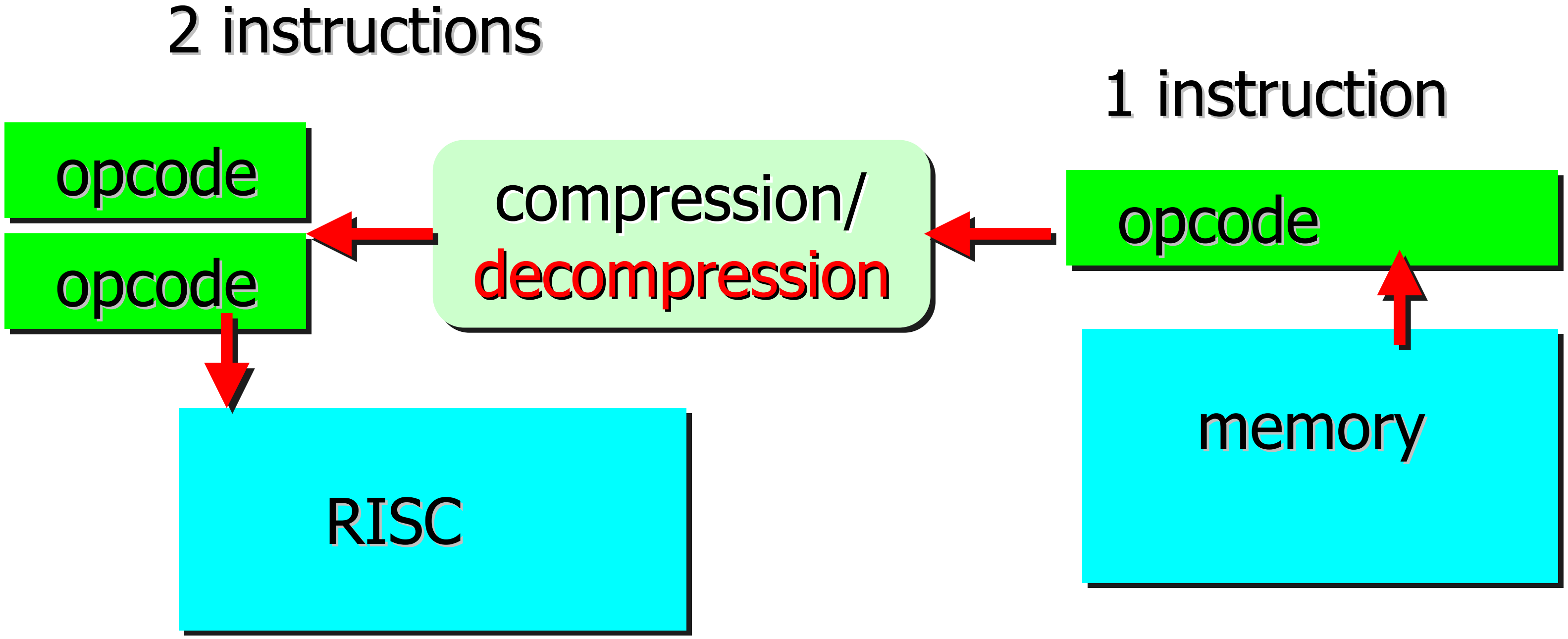
1 instruction

opcode

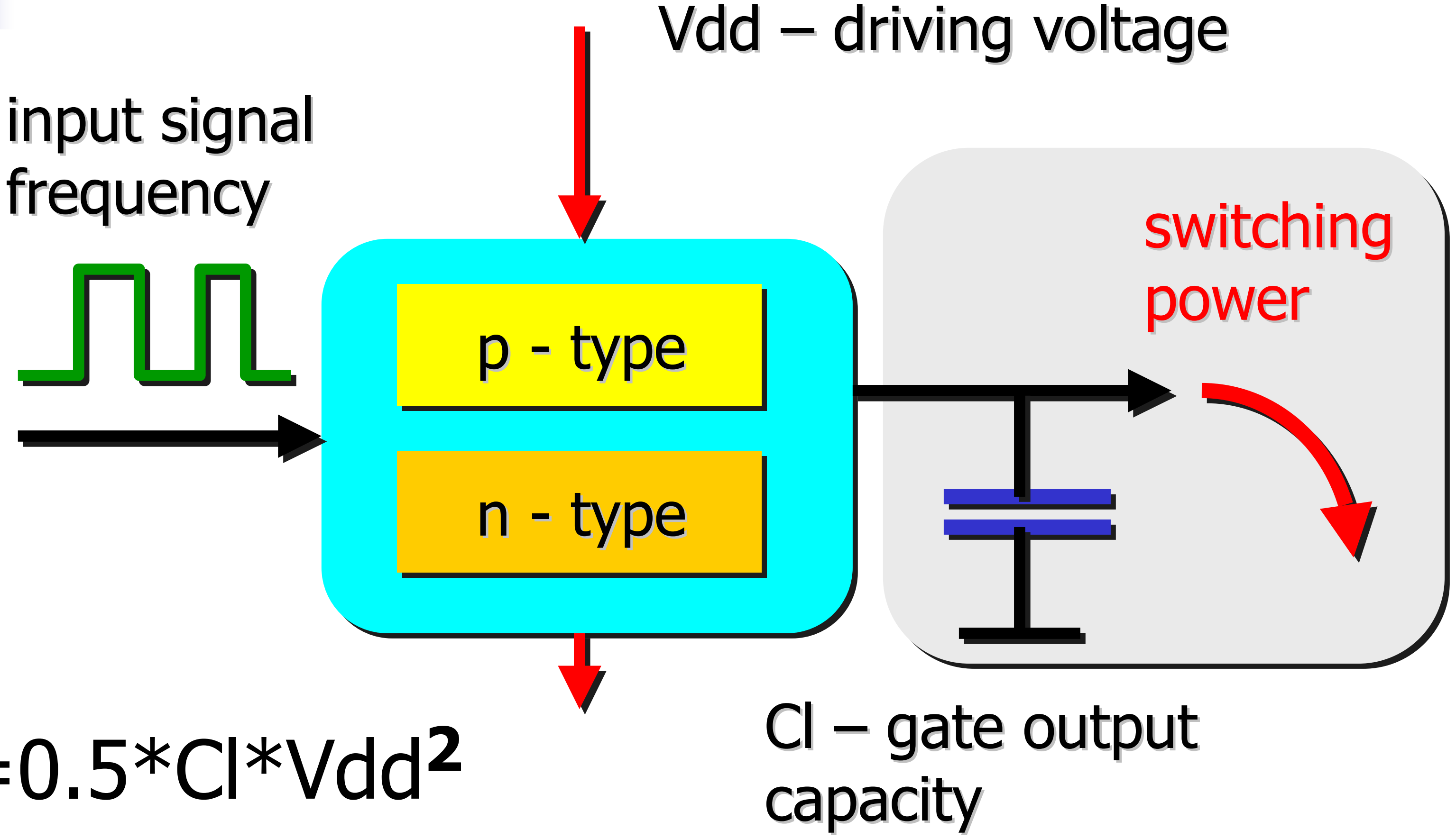
memory

Risc drawback

The solution to this problem is code compression/decompresion mechanism (ARM)



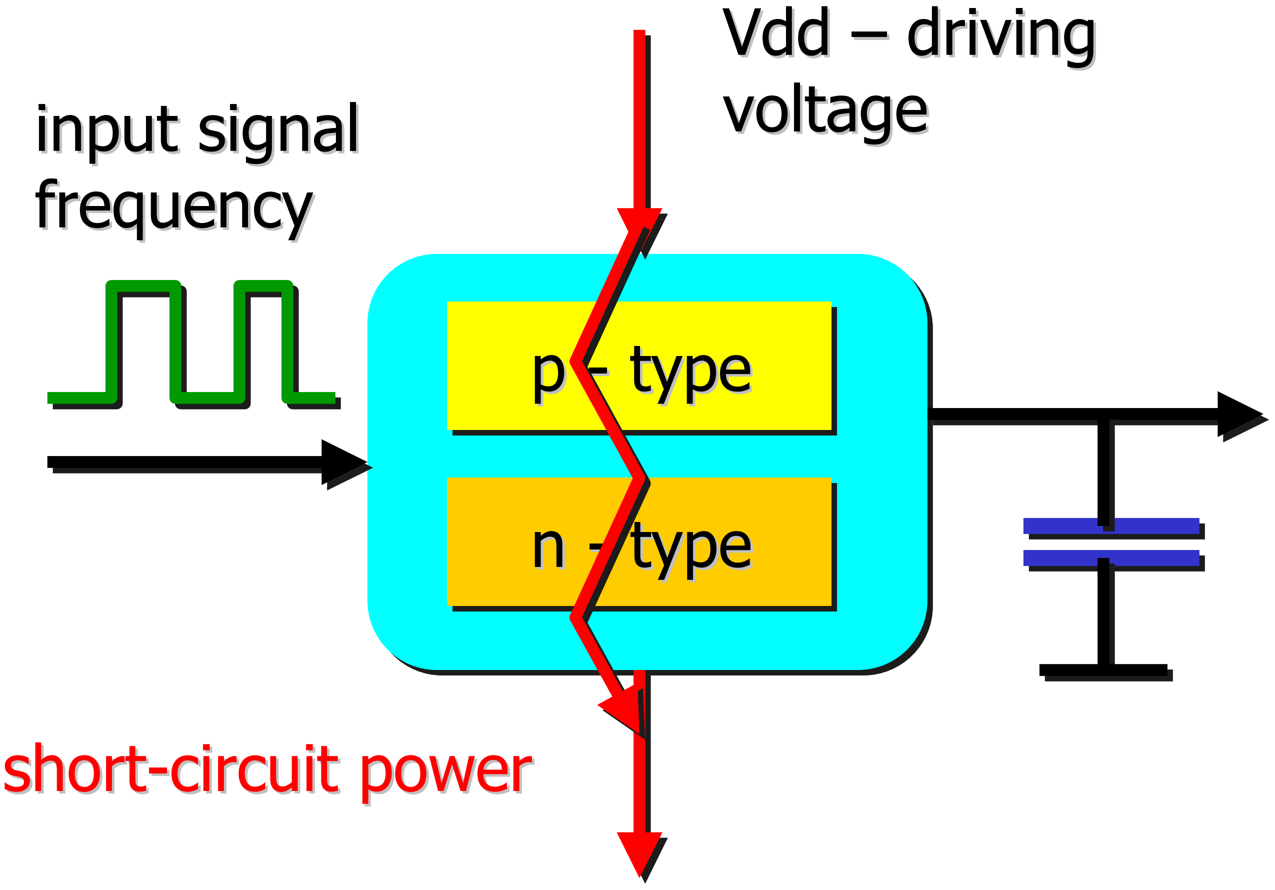
Low power consumption



E_t=0.5*Cl*Vdd^2

switching power par transition

Low power consumption



input signal frequency

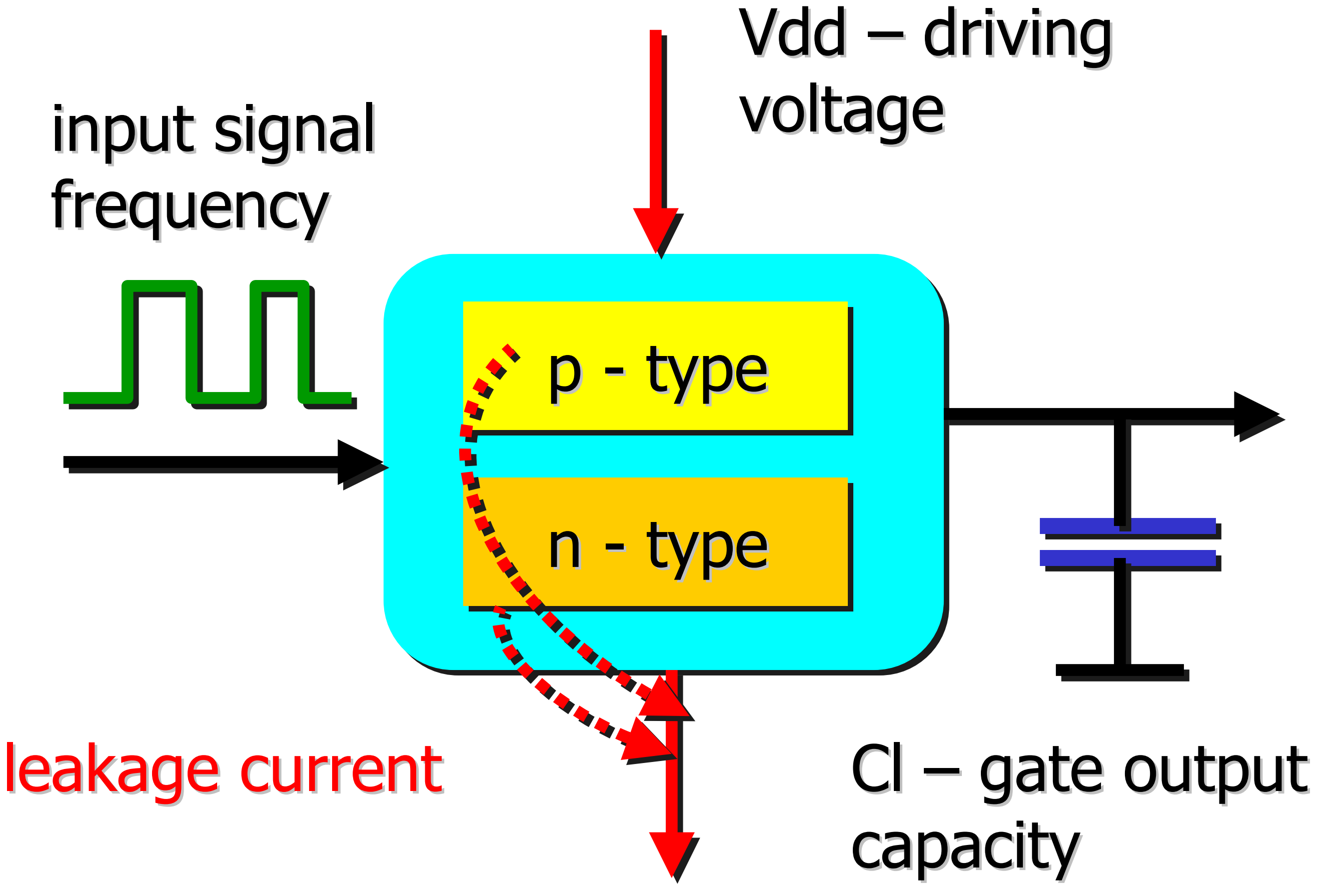
Vdd – driving voltage

p - type

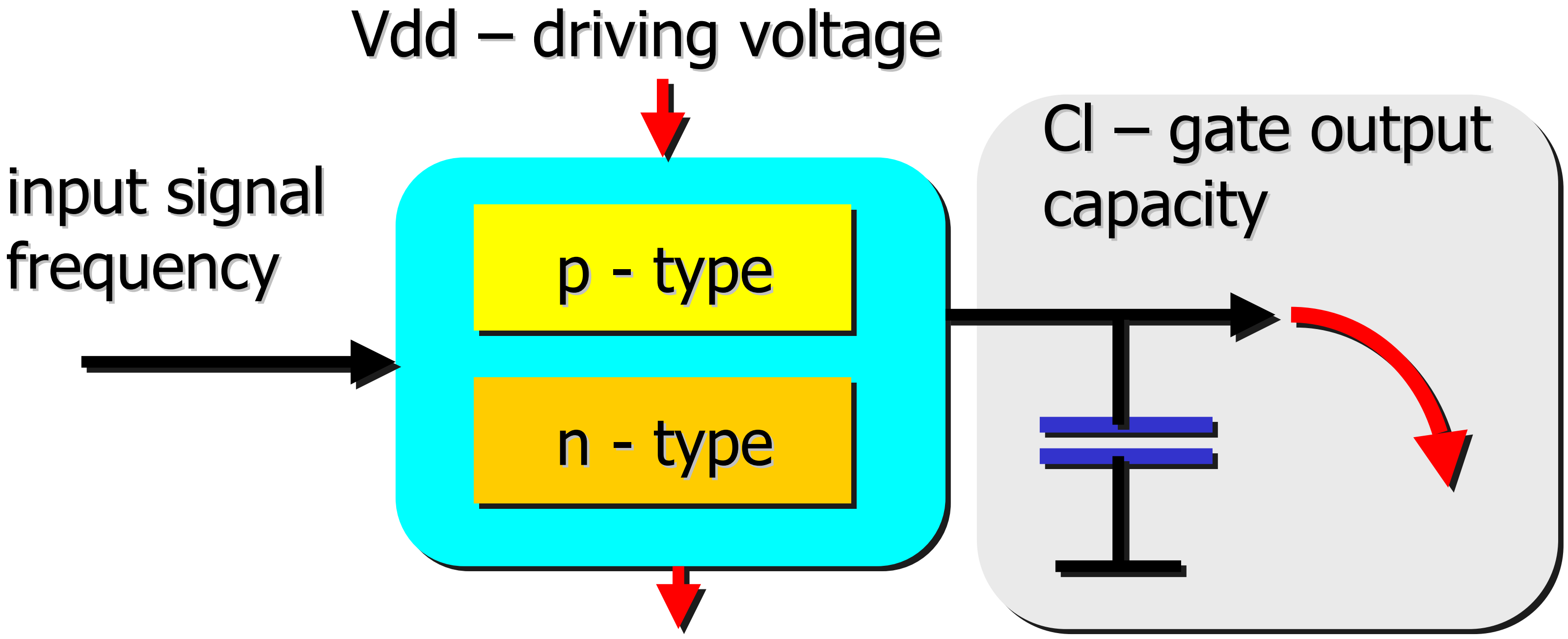
n - type

short-circuit power

Low power consumption



Total dynamic power consumption



$$P_c = 0.5 * f * Vdd^2 * \Sigma A_g * Cl$$

A_g - gate activity factor

Low power consumption

$$P_c = 0.5 * f * Vdd^2 * \Sigma A_g * Cl$$

- minimize supply voltage : technology
- minimize circuit activity
- minimize number of gates
- minimize clock frequency

Low power consumption

$$P_c = 0.5 * f * Vdd^2 * \Sigma A_g * Cl$$

- minimize supply voltage
- minimize circuit activity : utilization
- minimize number of gates
- minimize clock frequency

Low power consumption

$$P_c = 0.5 * f * Vdd^2 * \Sigma A_g * Cl$$

- minimize supply voltage
- minimize circuit activity
- minimize number of gates : design
- minimize clock frequency

Low power consumption

$$P_c = 0.5 * f * Vdd^2 * \Sigma A_g * Cl$$

- minimize supply voltage
- minimize circuit activity
- minimize number of gates
- minimize clock frequency : problem !

Summary

- a simple 16-bit processor model
- instruction elaboration phases
- instruction types
- control path and data path
- high performance 32-bit processor
- RISC concept – advantages and drawbacks
- low power consumption features

Summary

- a simple 16-bit processor model
- instruction elaboration phases: fetch, decode, execute, write-back
- instruction types: arithmetic, load/store, control
- control path and data path
- high performance 32-bit processor
- RISC concept – advantages and drawbacks
- low power consumption features

Summary

- a simple 16-bit processor model
- instruction elaboration phases
- instruction types: arithmetic, load/store, control
- control path and data path
- high performance 32-bit processor
- RISC concept – advantages and drawbacks
- low power consumption features

Summary

- a simple 16-bit processor model
- instruction elaboration phases
- instruction types: arithmetic, load/store, control
- control path and data path
- high performance 32-bit processor
- RISC concept – advantages and drawbacks
- low power consumption features

Summary

- a simple 16-bit processor model
- instruction elaboration phases
- instruction types: arithmetic, load/store, control
- control path and data path
- high performance 32-bit processor
- RISC concept – advantages and drawbacks
- low power consumption features

Summary

- a simple 16-bit processor model
- instruction elaboration phases
- instruction types: arithmetic, load/store, control
- control path and data path
- high performance 32-bit processor
- RISC concept – advantages and drawbacks
- low power consumption features

Summary

- a simple 16-bit processor model
- instruction elaboration phases
- instruction types: arithmetic, load/store, control
- control path and data path
- high performance 32-bit processor
- RISC concept – advantages and drawbacks
- low power consumption features