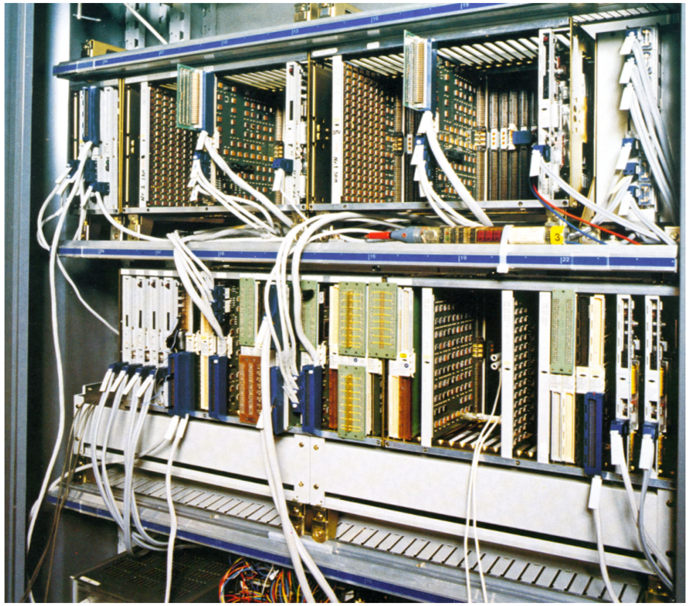
Introduction to Computer Architecture

Erik Hagersten
Uppsala University

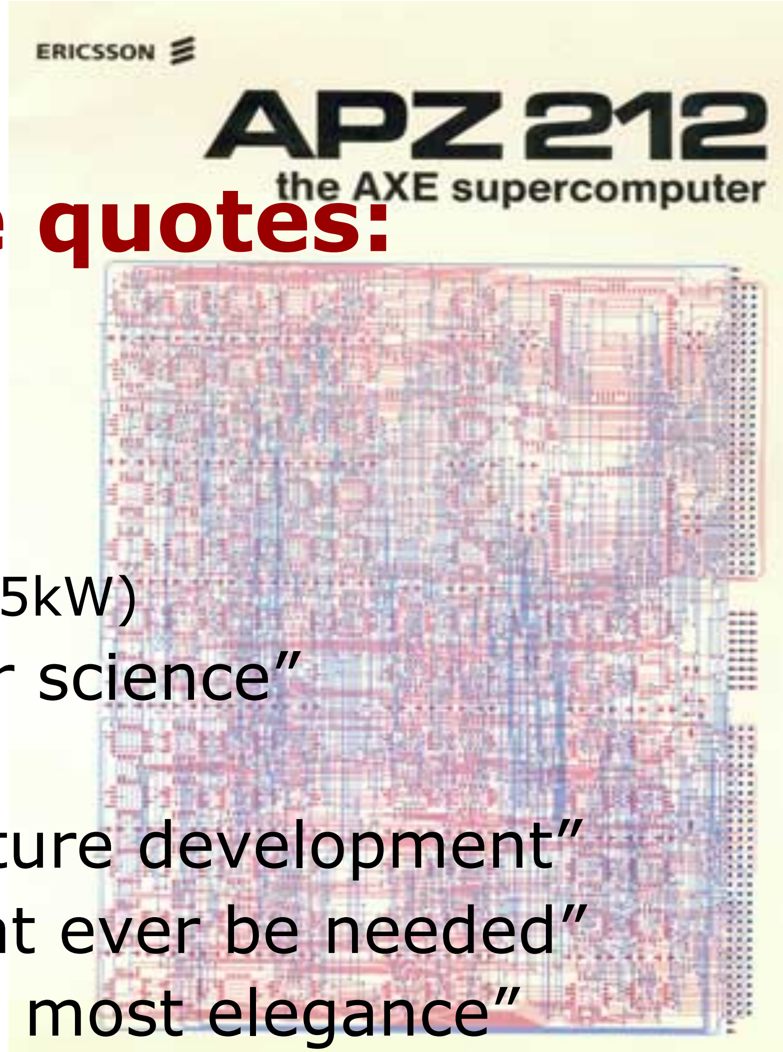
≈35 years ago: APZ 212 @ 5MHz

”the Ericsson supercomputer”



UPPSALA UNIVERSITET

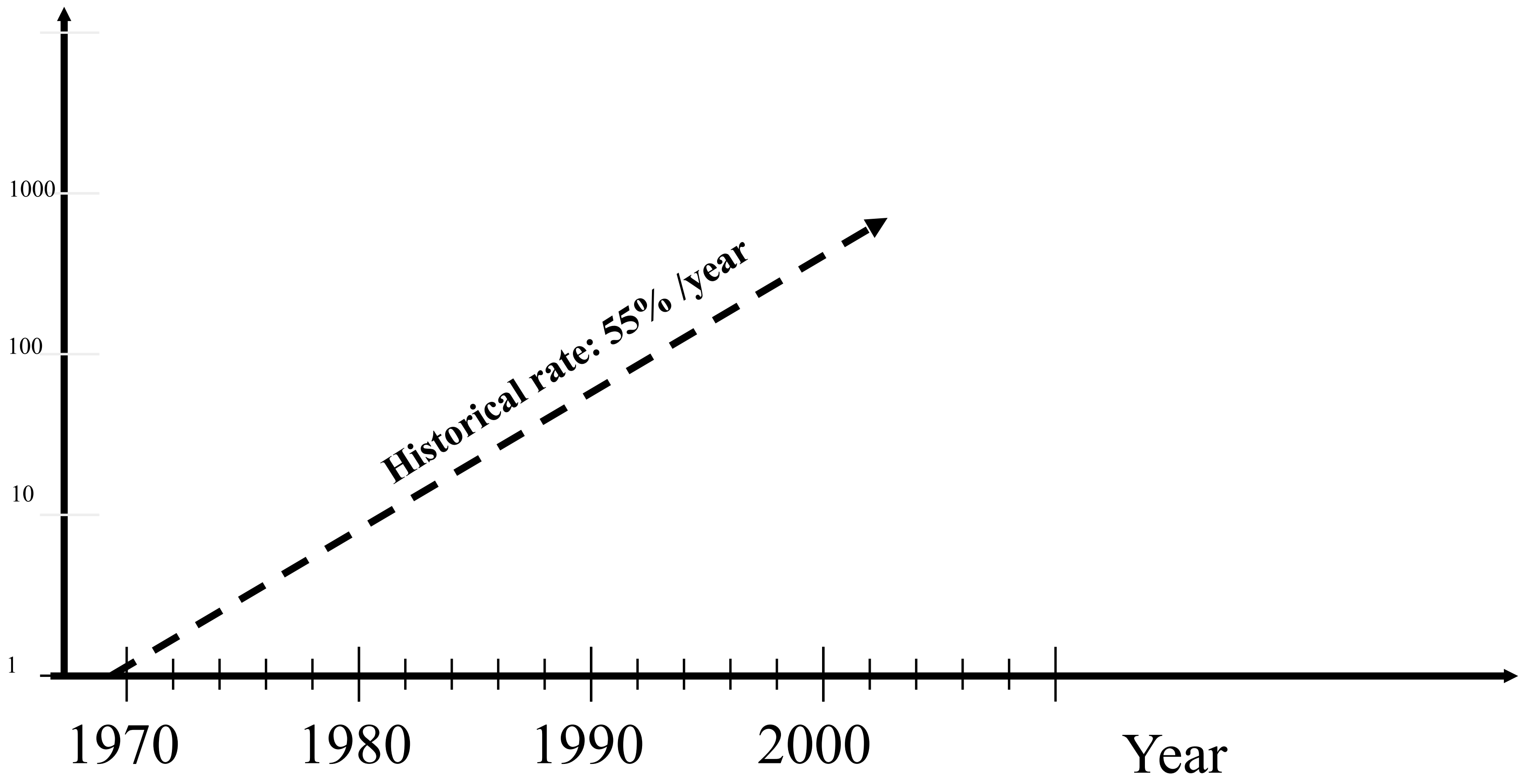
APZ 212
marketing brochure quotes:



- ”Very compact”
  - 6 times the performance
  - 1/6:th the size
  - 1/5 the power consumption (1.5kW)
- ”A breakthrough in computer science”
- ”Why more CPU power?”
- ”All the power needed for future development”
- ”…800,000 BHCA, should that ever be needed”
- ”SPC computer science at its most elegance”
- ”Using 64 kbit memory chips”

CPU Improvements

Relative Performance
[log scale]



How to get efficient architectures…

Very hard today

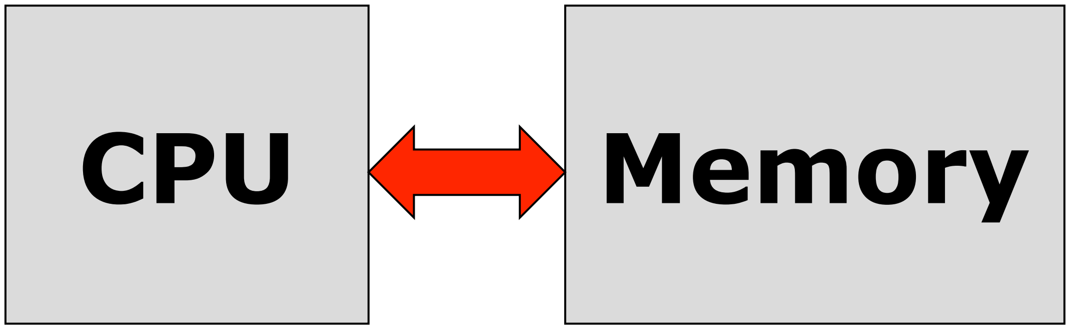
- ~~Increase clock frequency~~
- Create and explore locality:
  - a) Spatial locality
  - b) Temporal locality
- Create and explore parallelism
  - a) Instruction level parallelism (ILP)
  - b) Thread level parallelism (TLP)
  - c) Memory level parallelism (MLP)
- Speculative execution
  - a) Out-of-order execution
  - b) Branch prediction
  - c) Prefetching

Outline of these lectures

1. Processor implementations
2. Caches and memory system
3. Multiprocessors
4. HW optimizations
5. Multicores
6. *SW optimizations: (if there is time…)*

CPU architecture overview

Erik Hagersten
Uppsala University


CPU
Memory

UPPSALA UNIVERSITET

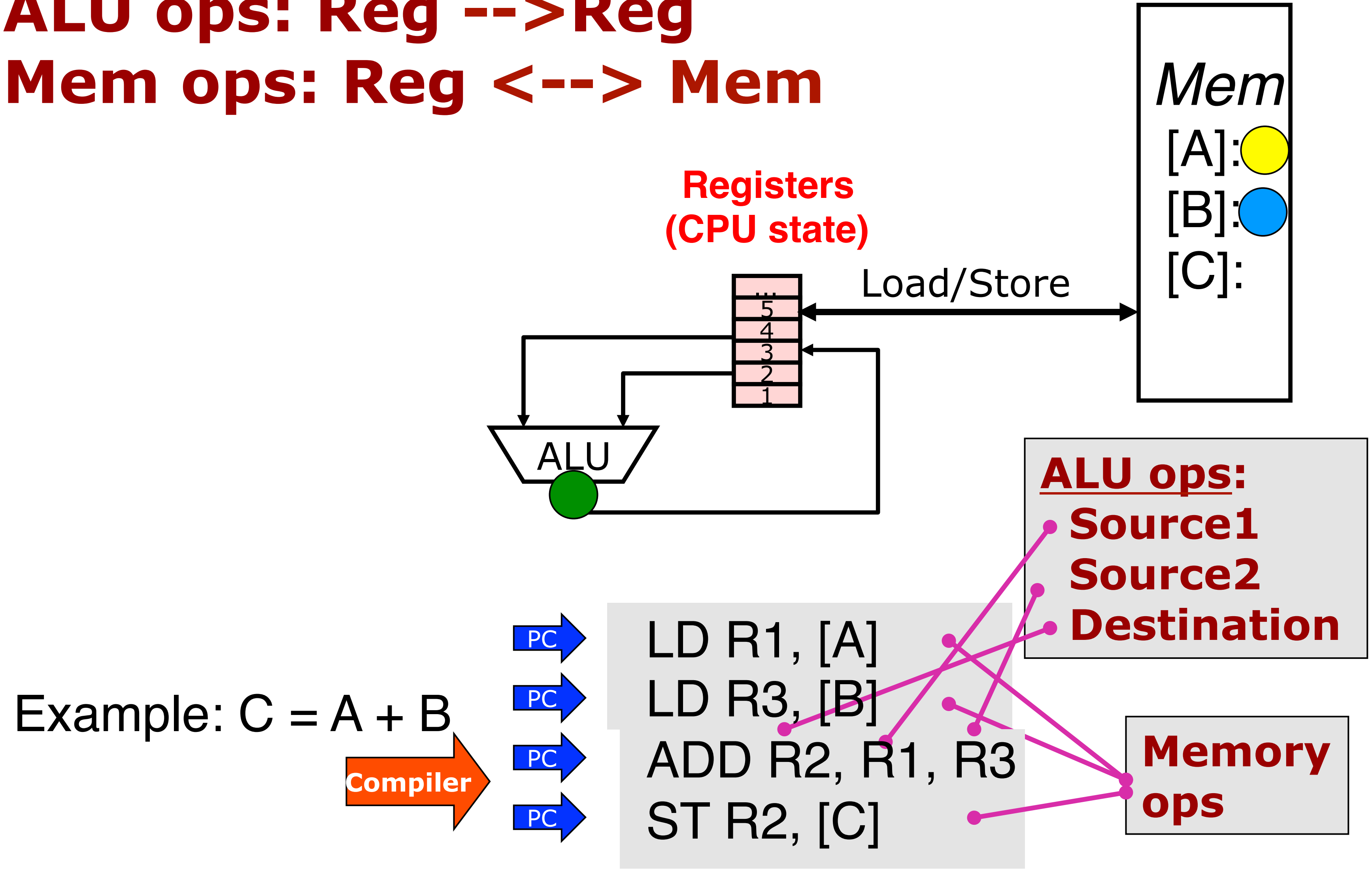
PDC Summer School 2016

How fast is a computer today?

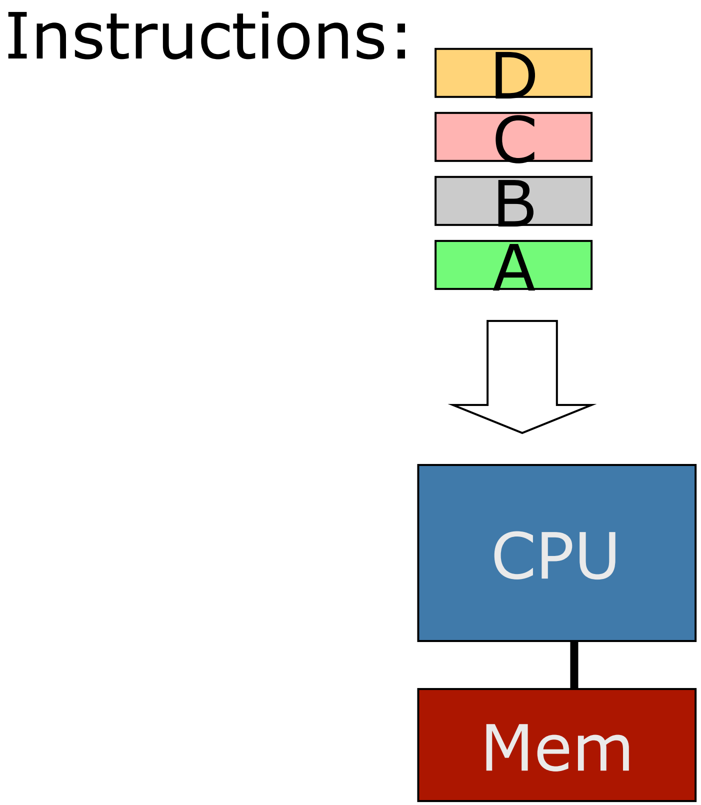
Load/Store architecture (e.g., ”RISC”)

ALU ops: Reg -->Reg

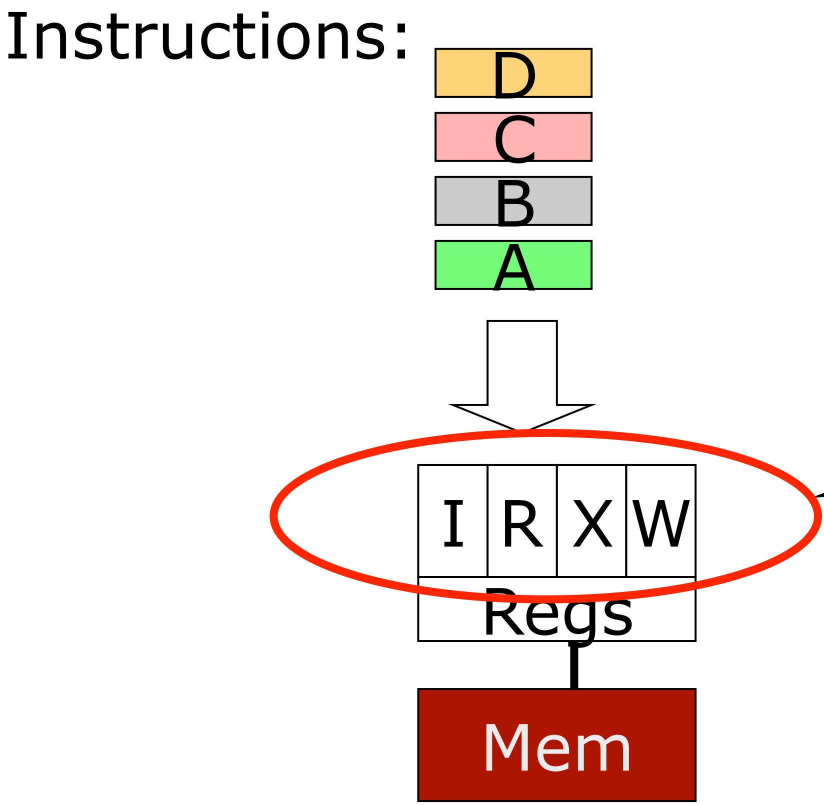
Mem ops: Reg <--> Mem



Lifting the CPU hood (simplified...)

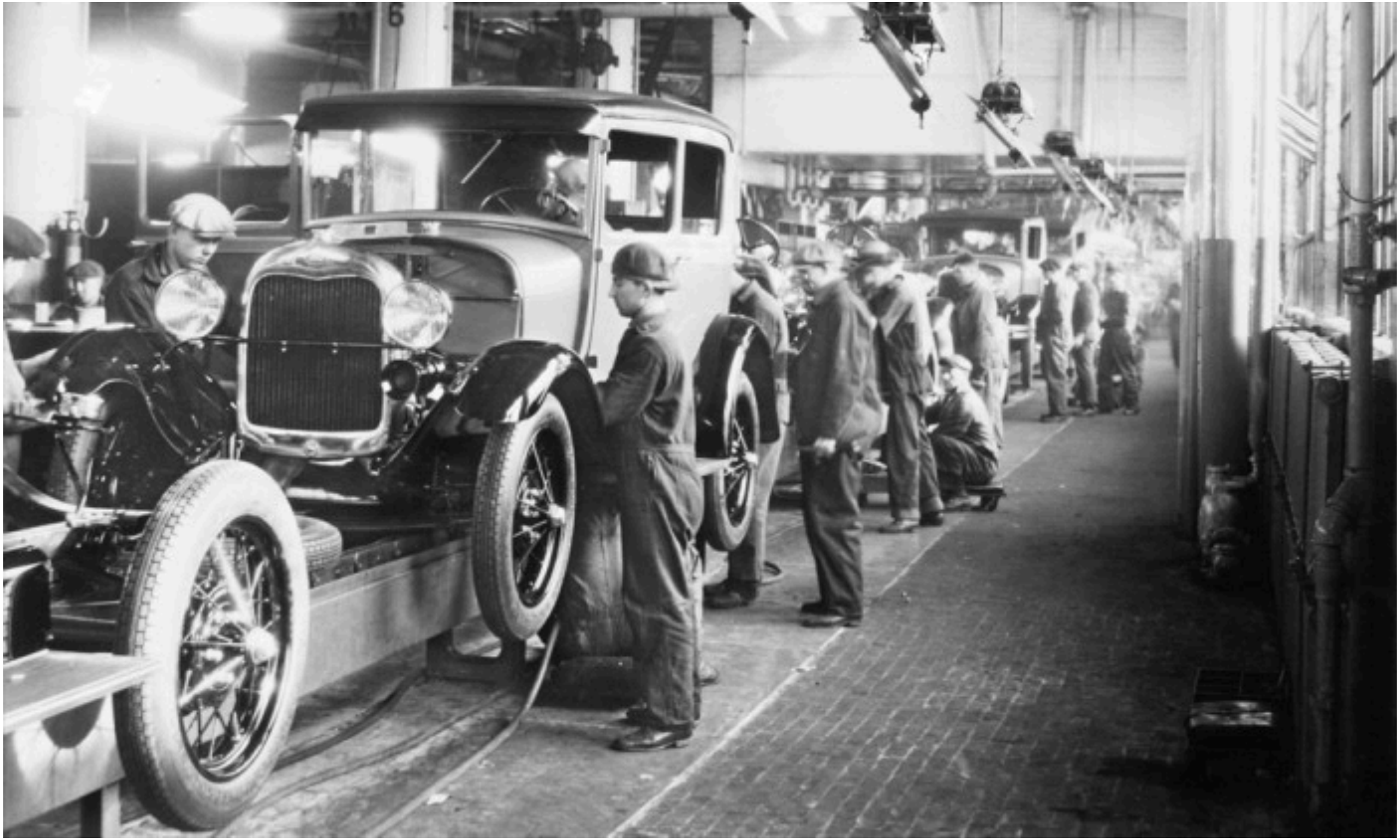


Pipeline



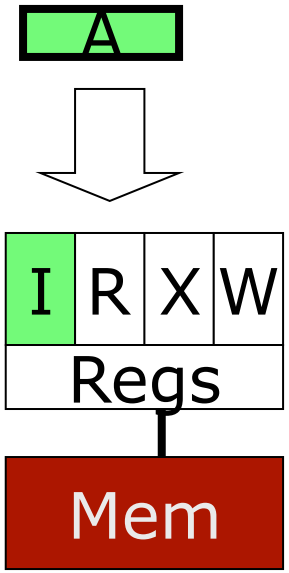
”Pipeline stages” ≈ workers of an assembly line

UPPSALA UNIVERSITET

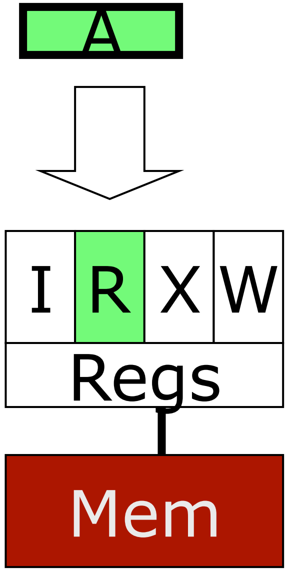


PDC Summer School 2016

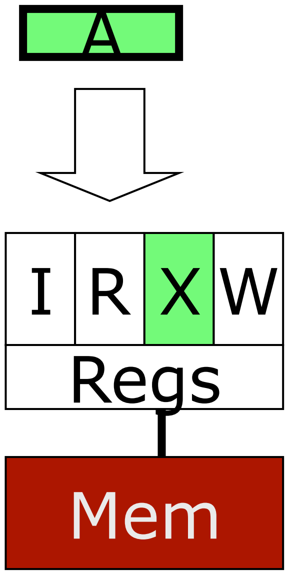
Pipeline



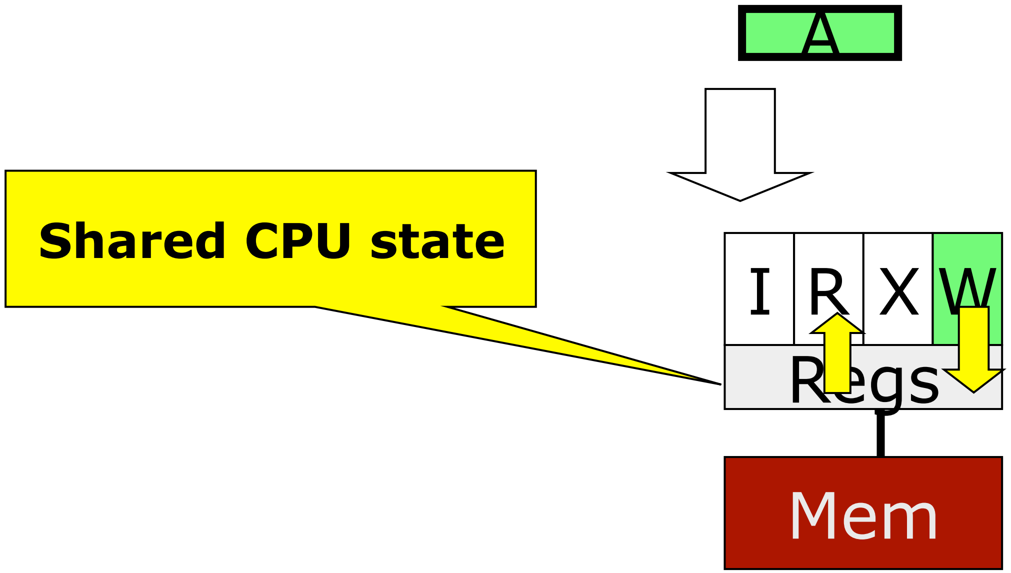
Pipeline



Pipeline



Pipeline:



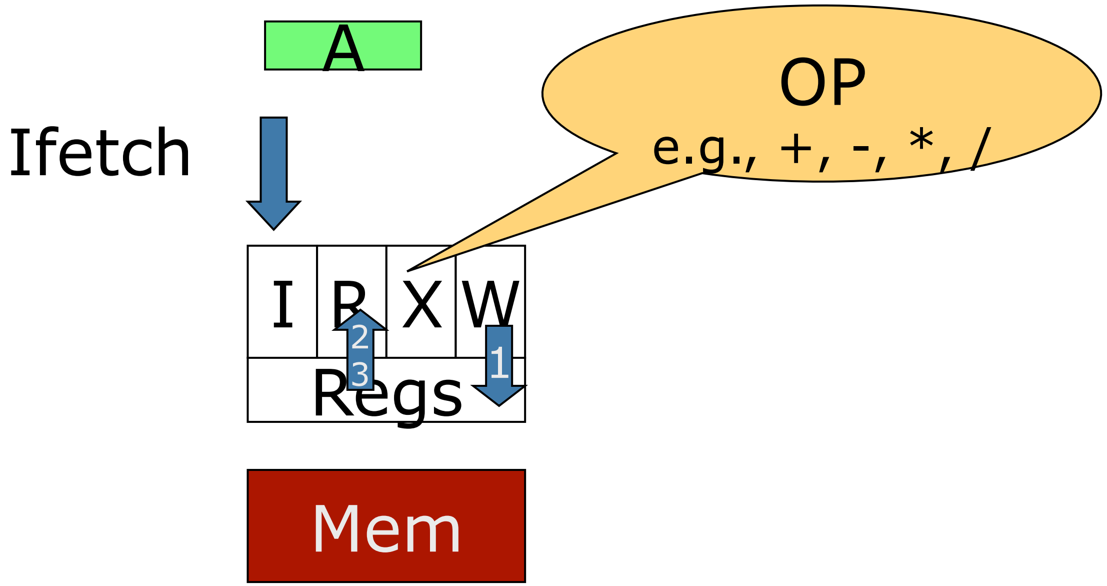
I = Instruction fetch
R = Read register
X = Execute
W= Write register/mem

Register Operations [ALU operation]

UPPSALA UNIVERSITET

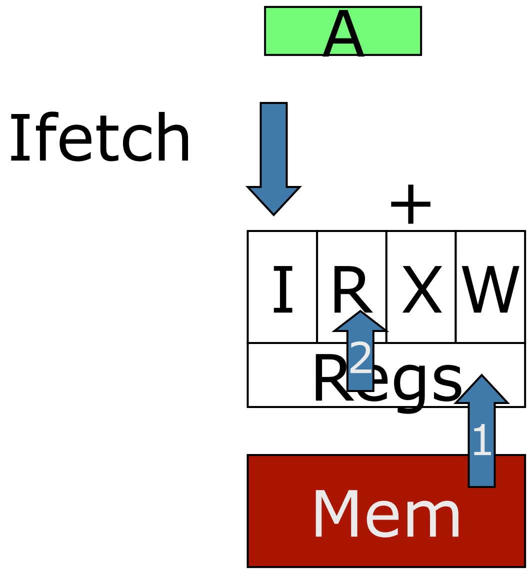
ADD R1, R2, R3

(a.k.a. R1 := R2 + R3)



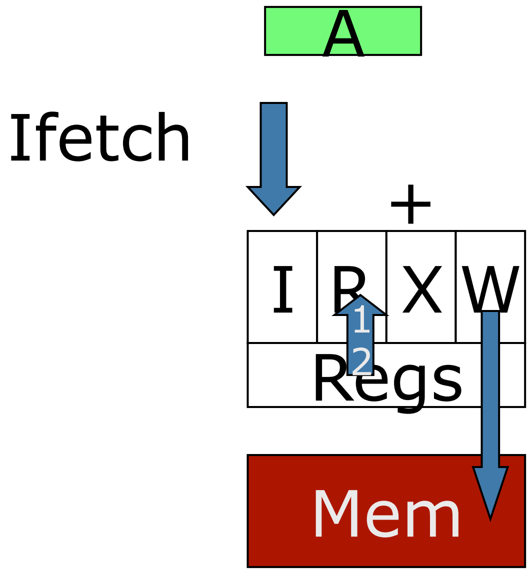
Load Operation:

LD R1, mem[cnst+R2]



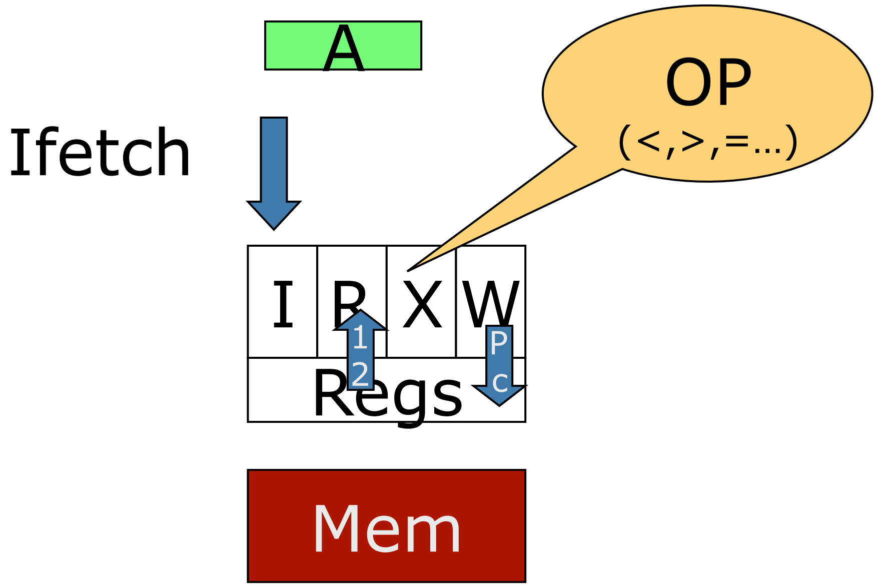
Store Operation:

ST R1, mem[cnst+R2]



Branch Operations:

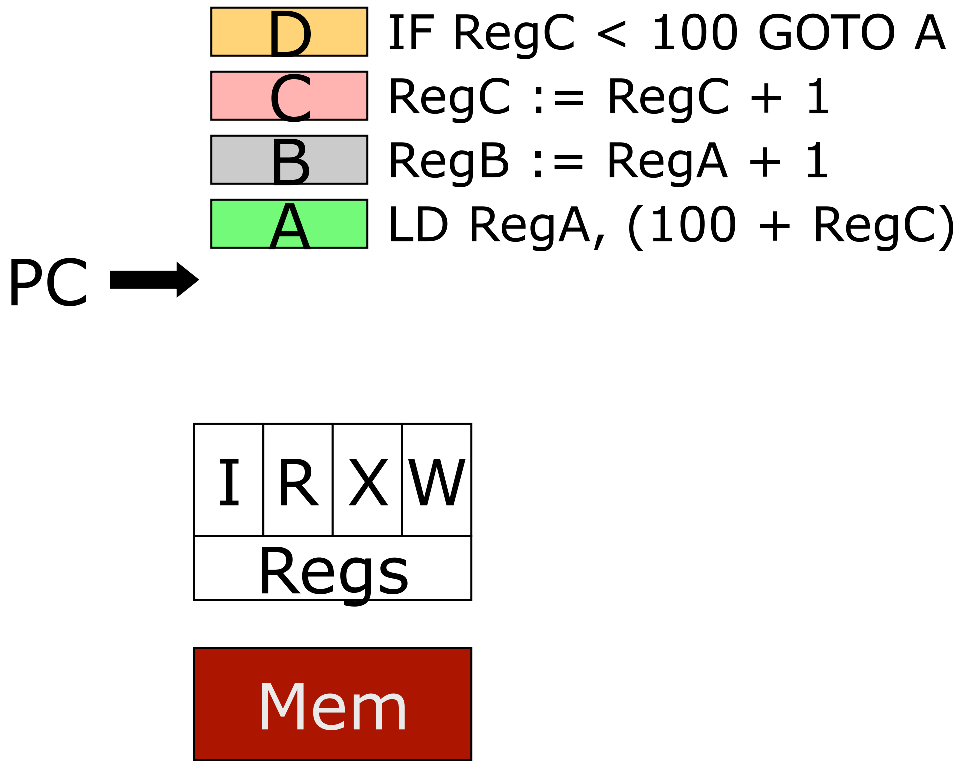
if (R1 < Const) GOTO mem[R2]



PC = Program Counter.
A special register pointing to the next instruction to execute

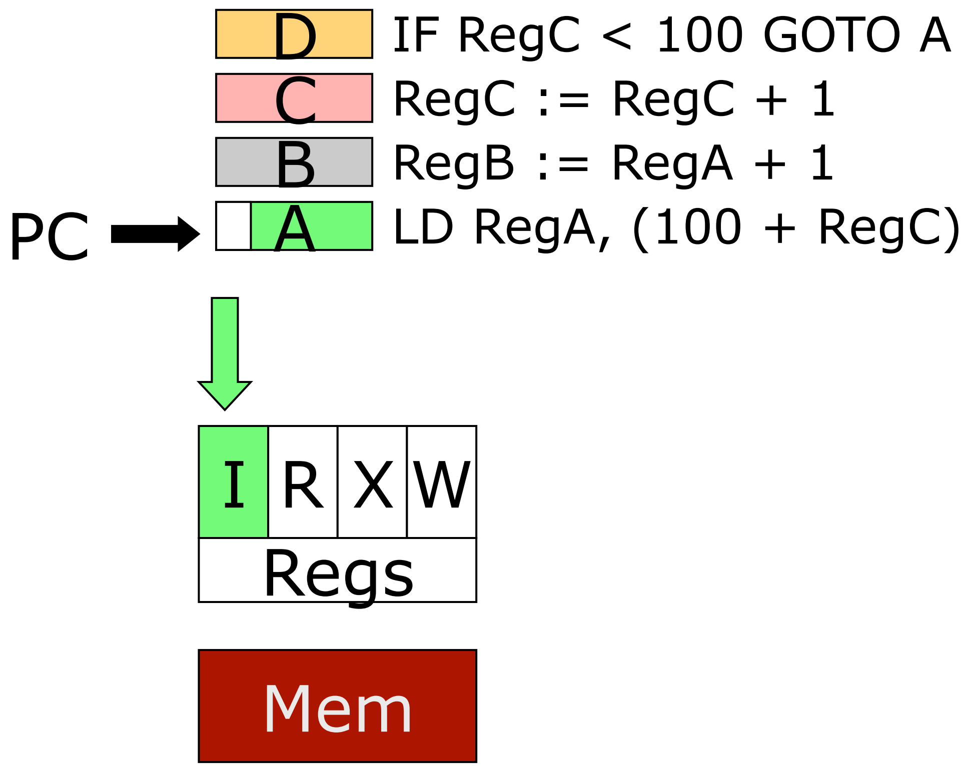
UPPSALA UNIVERSITET

Initially



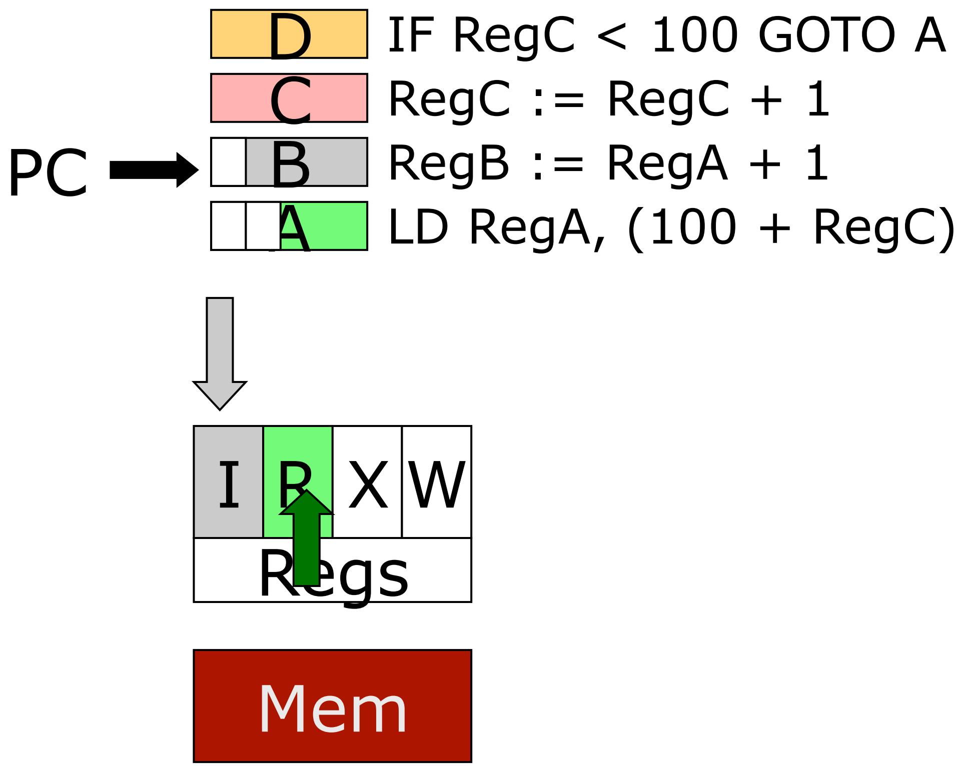
PDC Summer School 2016

Cycle 1

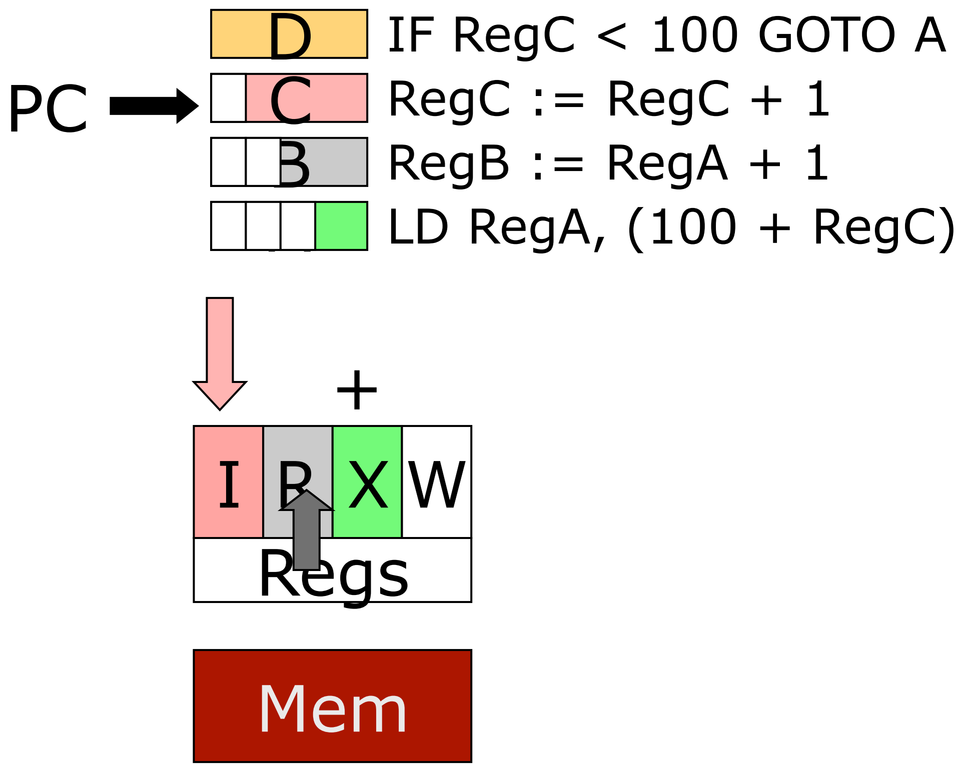


D
IF RegC < 100 GOTO A
C
RegC := RegC + 1
B
RegB := RegA + 1
PC
A
LD RegA, (100 + RegC)
I
R
X
W
Regs
Mem

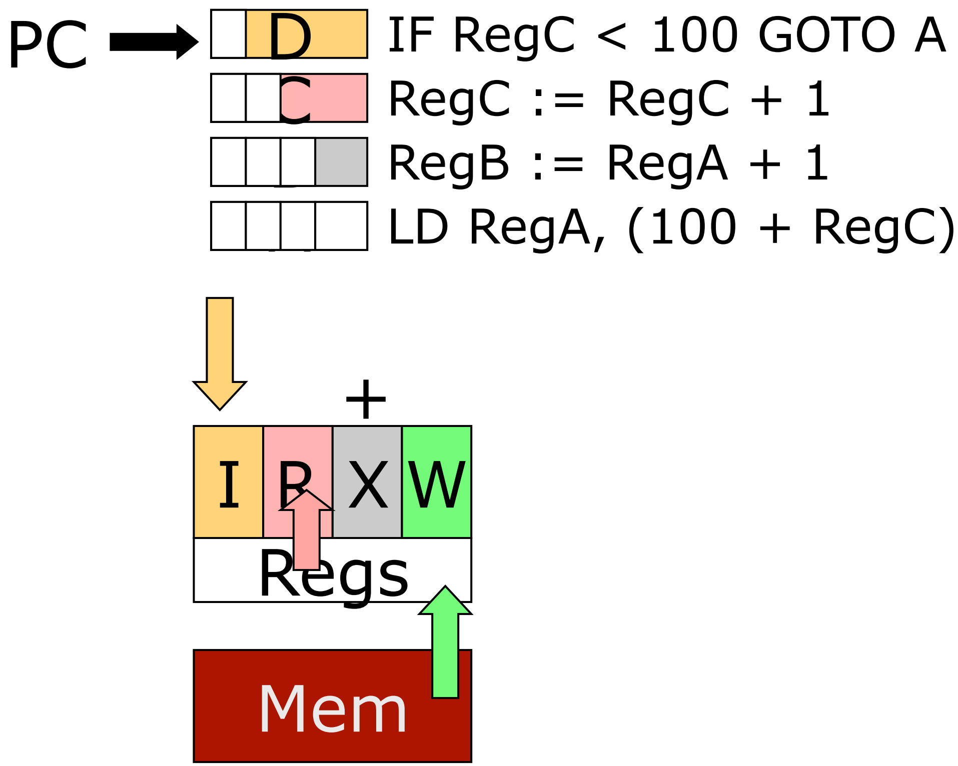
Cycle 2


D
IF RegC < 100 GOTO A
C
RegC := RegC + 1
PC
B
RegB := RegA + 1
A
LD RegA, (100 + RegC)
I
R
X
W
Regs
Mem

Cycle 3


D
IF RegC < 100 GOTO A
PC
C
RegC := RegC + 1
B
RegB := RegA + 1
LD RegA, (100 + RegC)
+
I
R
X
W
Regs
Mem

Cycle 4



PC
D
IF RegC < 100 GOTO A
C
RegC := RegC + 1
RegB := RegA + 1
LD RegA, (100 + RegC)
+
I
R
X
W
Regs
Mem

UPPSALA UNIVERSITET

Cycle 5

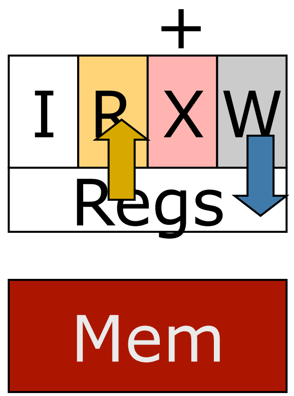
PC

IF RegC < 100 GOTO A

RegC := RegC + 1

RegB := RegA + 1

LD RegA, (100 + RegC)

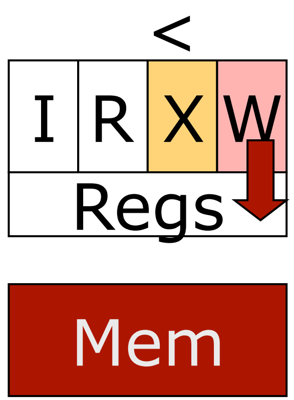


PDC Summer School 2016

Cycle 6

PC

IF RegC < 100 GOTO A
RegC := RegC + 1
RegB := RegA + 1
LD RegA, (100 + RegC)



Cycle 7

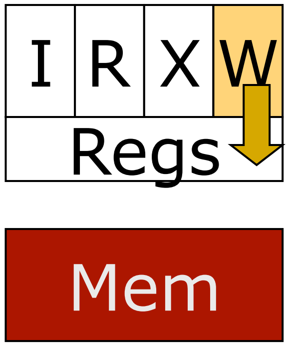
PC →

IF RegC < 100 GOTO A

RegC := RegC + 1

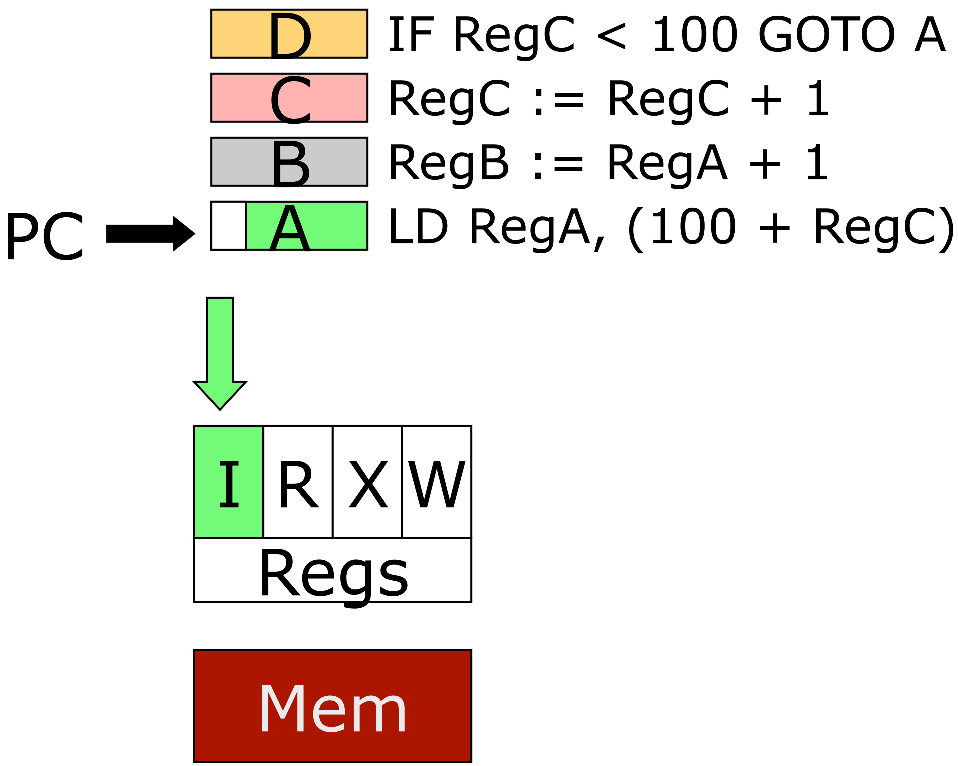
RegB := RegA + 1

A: LD RegA, (100 + RegC)



Branch: Write "A" to PC

Cycle 8

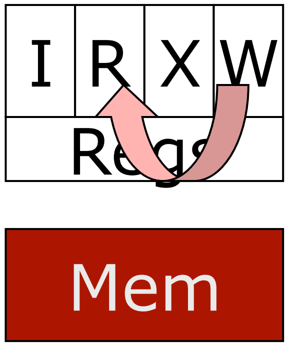


First example of "Instruction Level Parallelism" (ILP) -- working on several instructions simultaneously. Improves "throughput" of instructions, not latency

UPPSALA UNIVERSITET

Data dependency ☹ Previous execution example wrong!

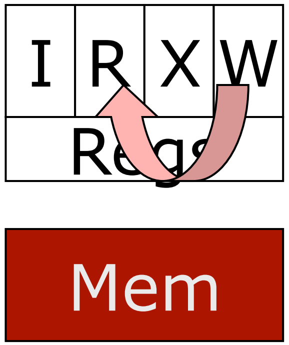
D IF RegC < 100 GOTO A
C RegC := RegC + 1
B RegB := RegA + 1
A LD RegA, (100 + RegC)



PDC Summer School 2016

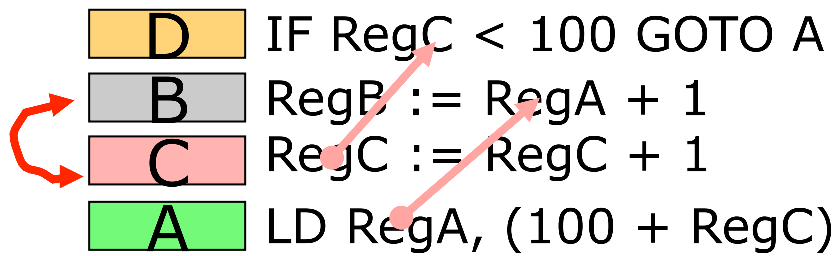
Data dependency fix 1: pipeline stalls

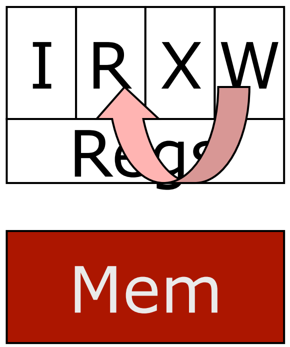
D IF RegC < 100 GOTO A
"Stall"
C RegC := RegC + 1
B RegB := RegA + 1
"Stall"
A LD RegA, (100 + RegC)



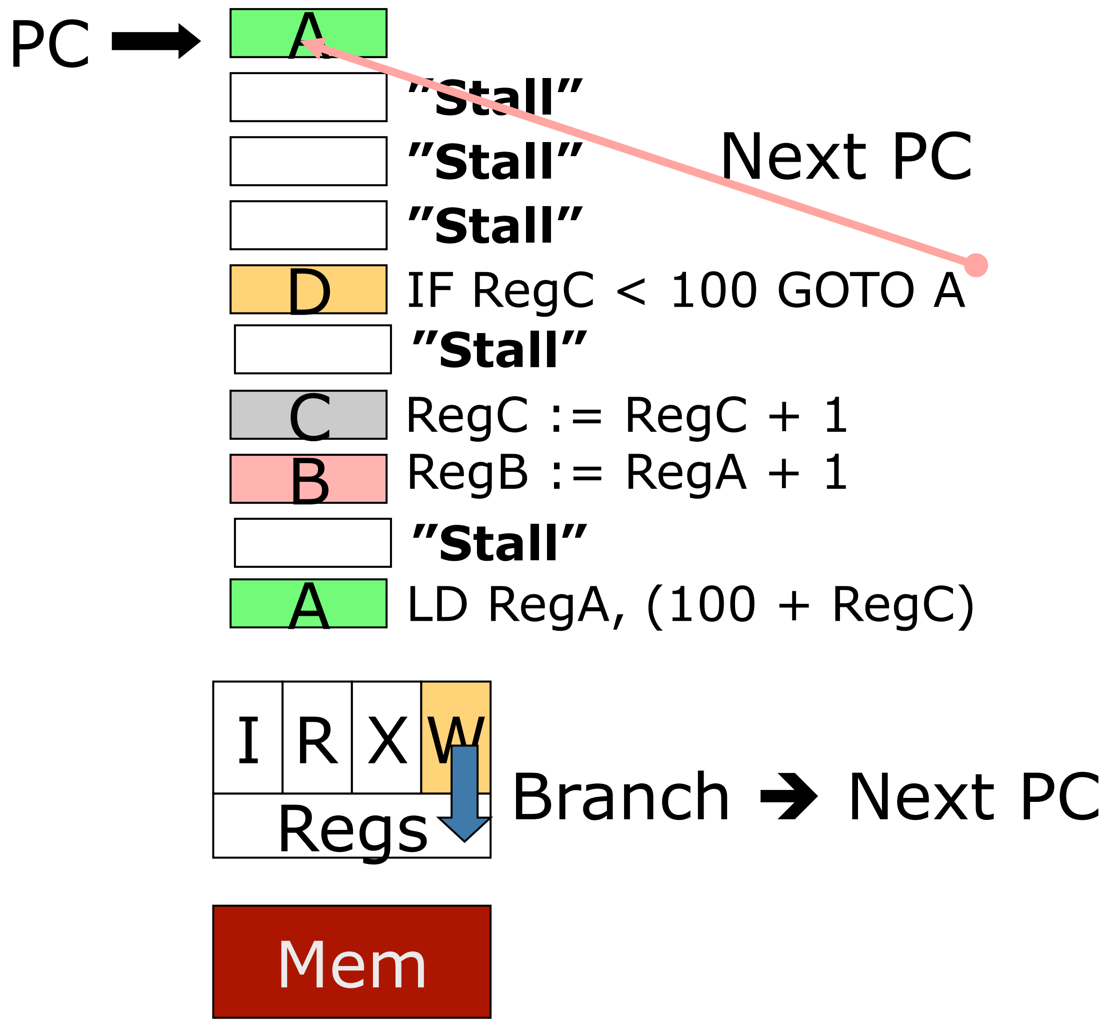
UPPSALA UNIVERSITET

Data dependency fix 2: Compiler instruction scheduling





Branch delays ☹



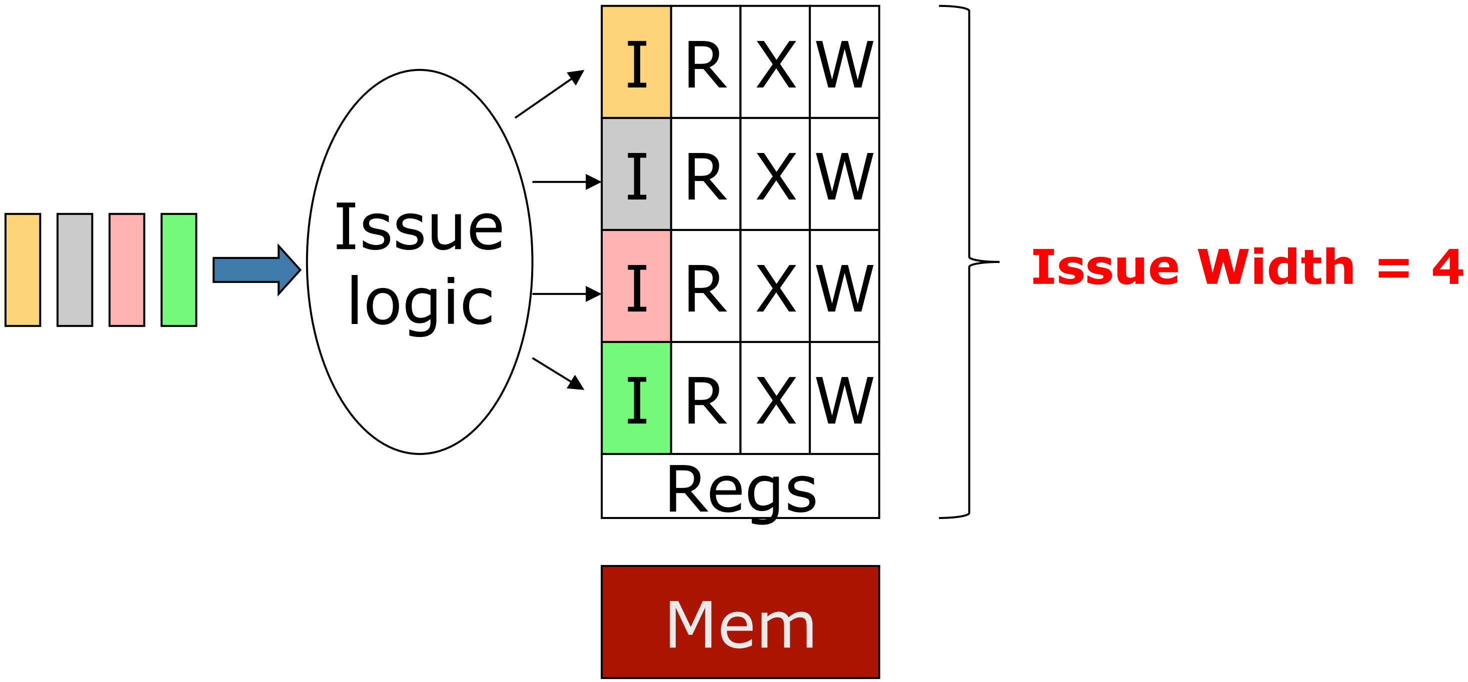
9 cycles per iteration of 4 instructions ☹
Need longer "basic blocks" with independent instr.

Pipeline Challenges

- Balance the pipeline stages
- Setup and hold time overhead
- Minimize pipeline stalls
- Predict and perform speculative work
- Undo speculative work

It is actually a lot worse!

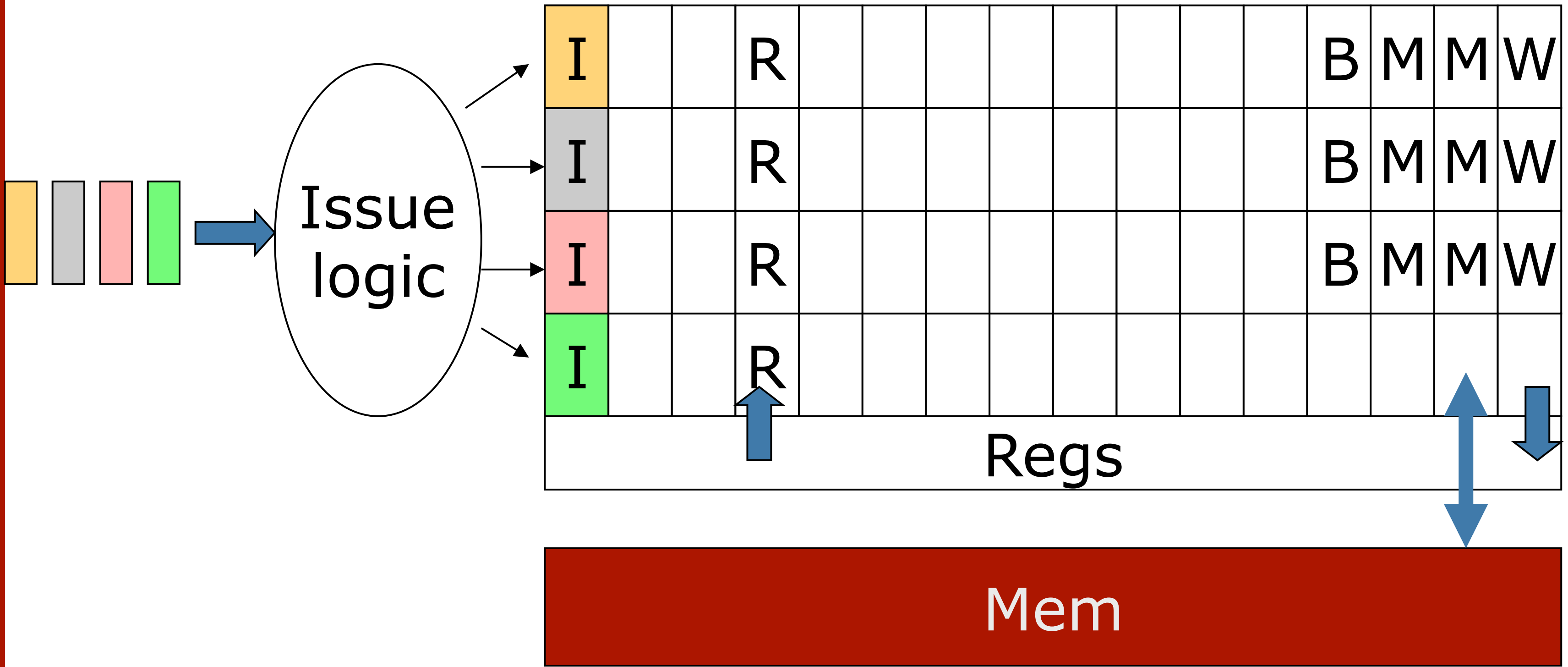
Modern CPUs:”superscalars” with ≈4 parallel pipelines



+More ILP → Higher throughput
- More complicated architecture
- Branch delay more expensive (more instr. wasted)
- Harder to find ”enough” independent instr. (need 8 instr. between write and use)

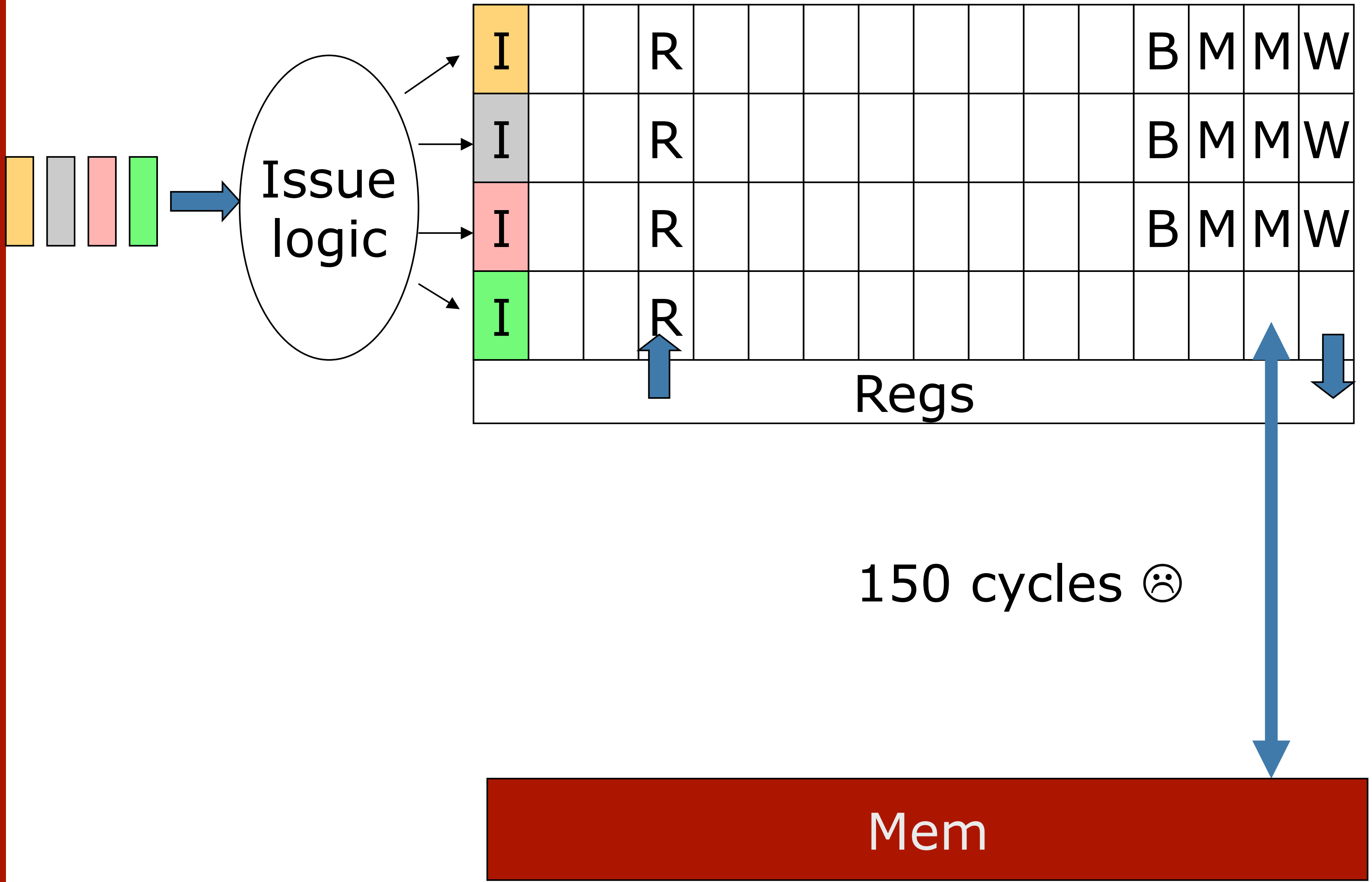
UPPSALA UNIVERSITET

It is actually a lot worse!
Modern CPUs: ~10-20 stages/pipe



+Explores even more ILP
+Shorter cycletime (higher frequency)
- Branch delay even more expensive
- Even harder to find ”enough” independent instr.

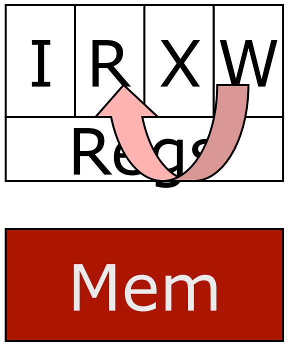
It is actually a lot worse!
DRAM access: ~150 CPU cycles



UPPSALA UNIVERSITET

Pipeline delays gets worse

D	IF RegC < 100 GOTO A
x1	**"Stall"**
C	RegC := RegC + 1
B	RegB := RegA + 1
x151	**"Stall"**
A	LD RegA, (100 + RegC)



Fix: Use a cache