# COL216
# Computer Architecture

Processor design -
Pipelining
7th February, 2022

# Outline of this lecture

- Timings of Single cycle, multi-cycle designs
- Increasing performance with pipelining
- Limitations of pipelined approach
  - Hazards
- Handling hazards
  - Removing hazards
  - Reducing effect of hazards

# Single cycle design

# Problems with single cycle design

- Slowest instruction pulls down the clock frequency
- Resource utilization is poor
- There are some instructions which are impossible to be implemented in this manner

# Multi-cycle design

# Features of multi-cycle design

- Actions split into multiple steps
- Registers hold intermediate values
- All instructions need not take same steps
- Clock frequency decided by slowest step
- Resources can be shared across cycles
  - Eliminate adders
  - Use single memory
  - More multiplexing

# Improving the design further

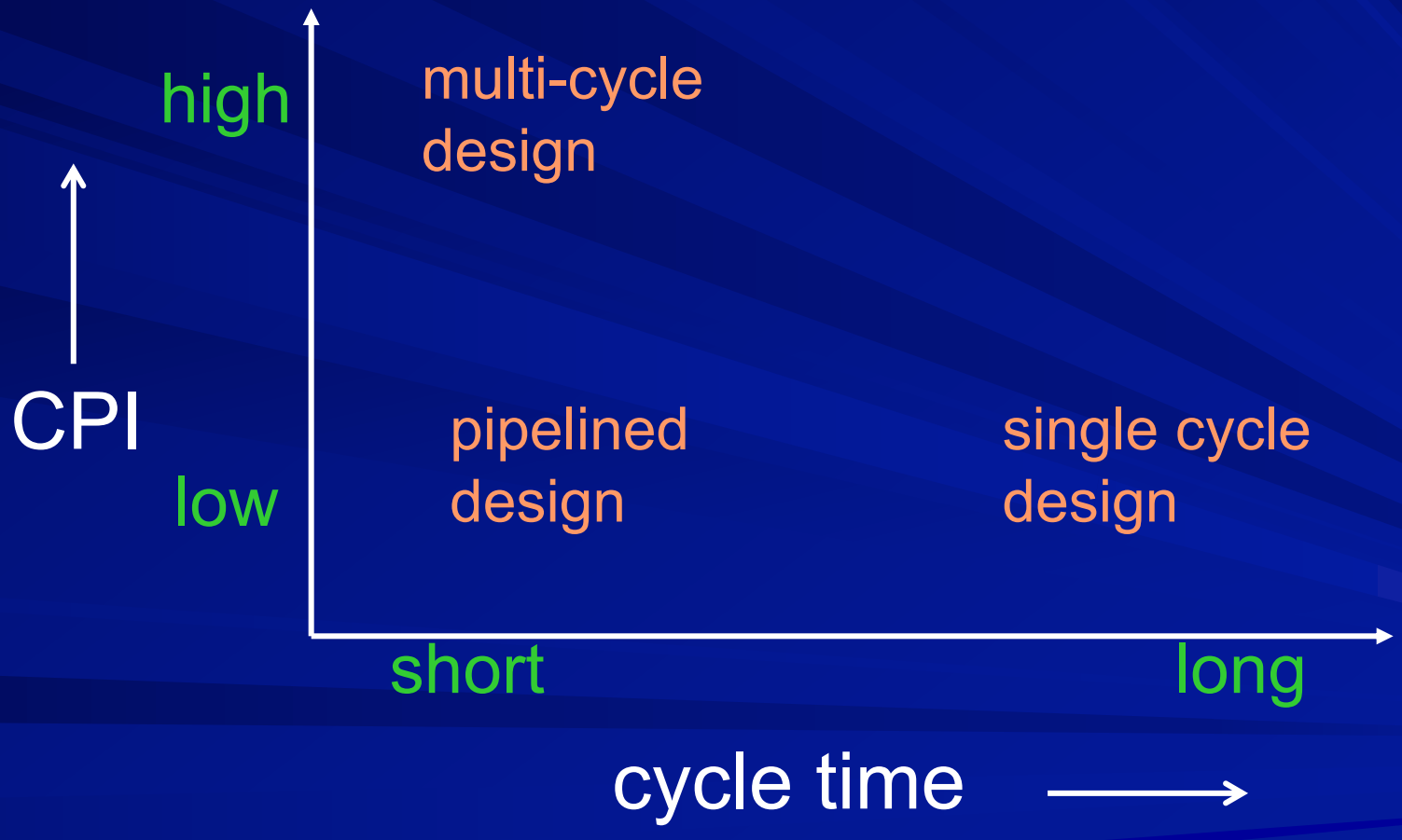
If resource saving is not important,
can the performance be improved further?

# Pipelined design

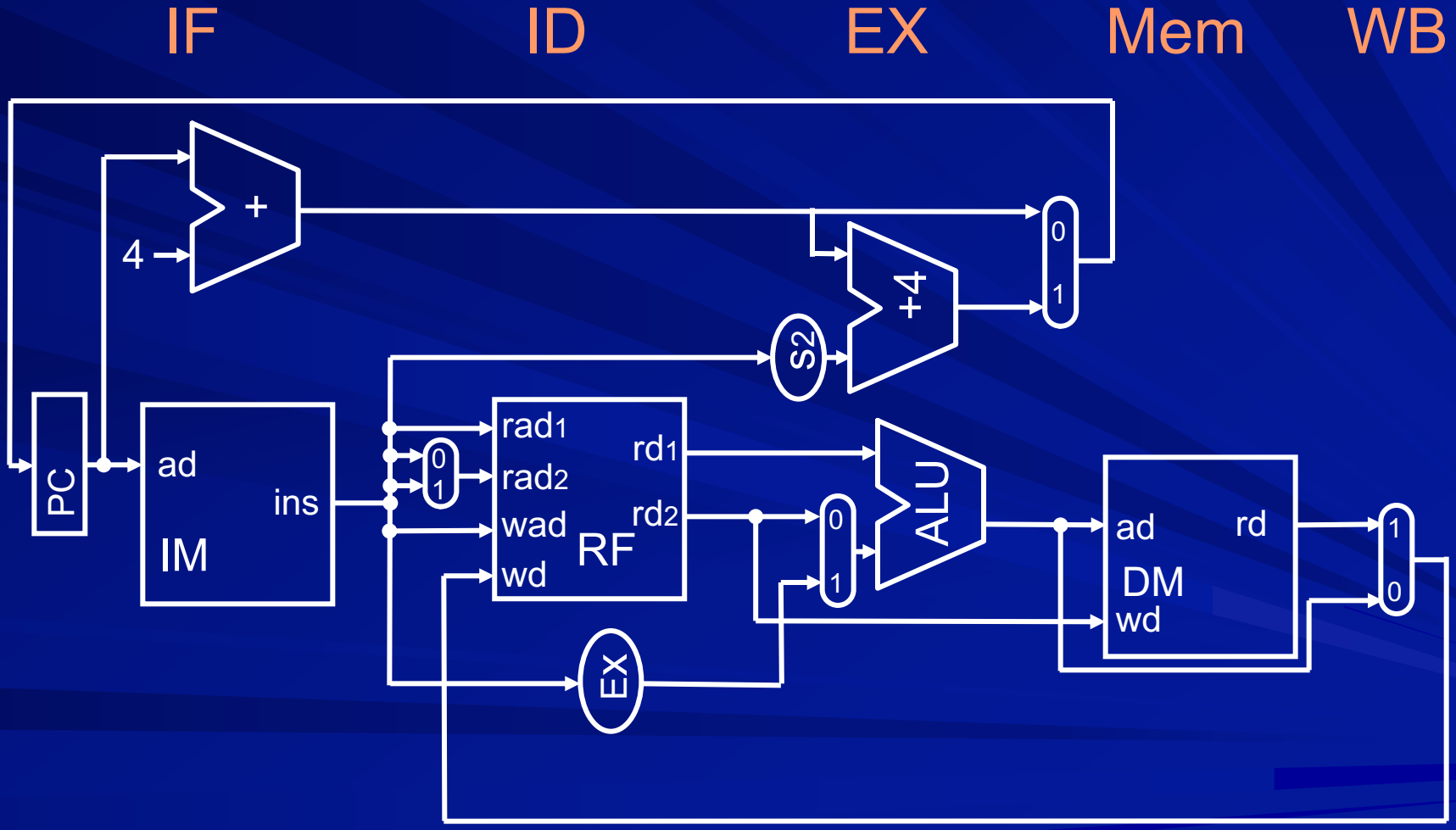
# Resource usage in different designs

- Single cycle design
  - each resource is tied up for the entire duration of the instruction execution
- Multi-cycle design
  - resource utilized in cycle `t` of instruction `I` is available again for cycle `t+1` of instruction `I`
- Pipelined design
  - resource utilized in cycle `t` of instruction `I` is available again for cycle `t` of instruction `I+1`

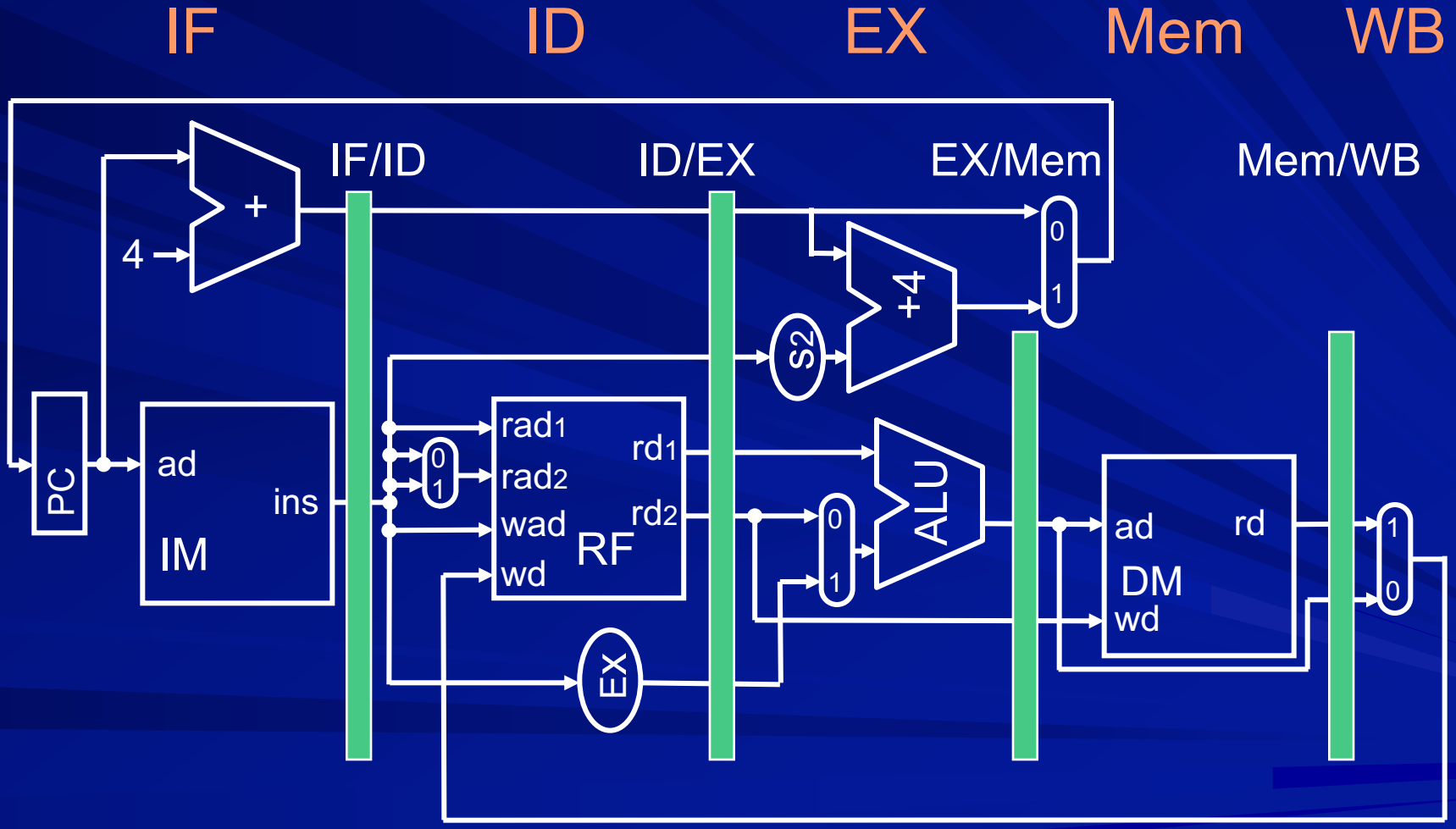
# Performance of different designs



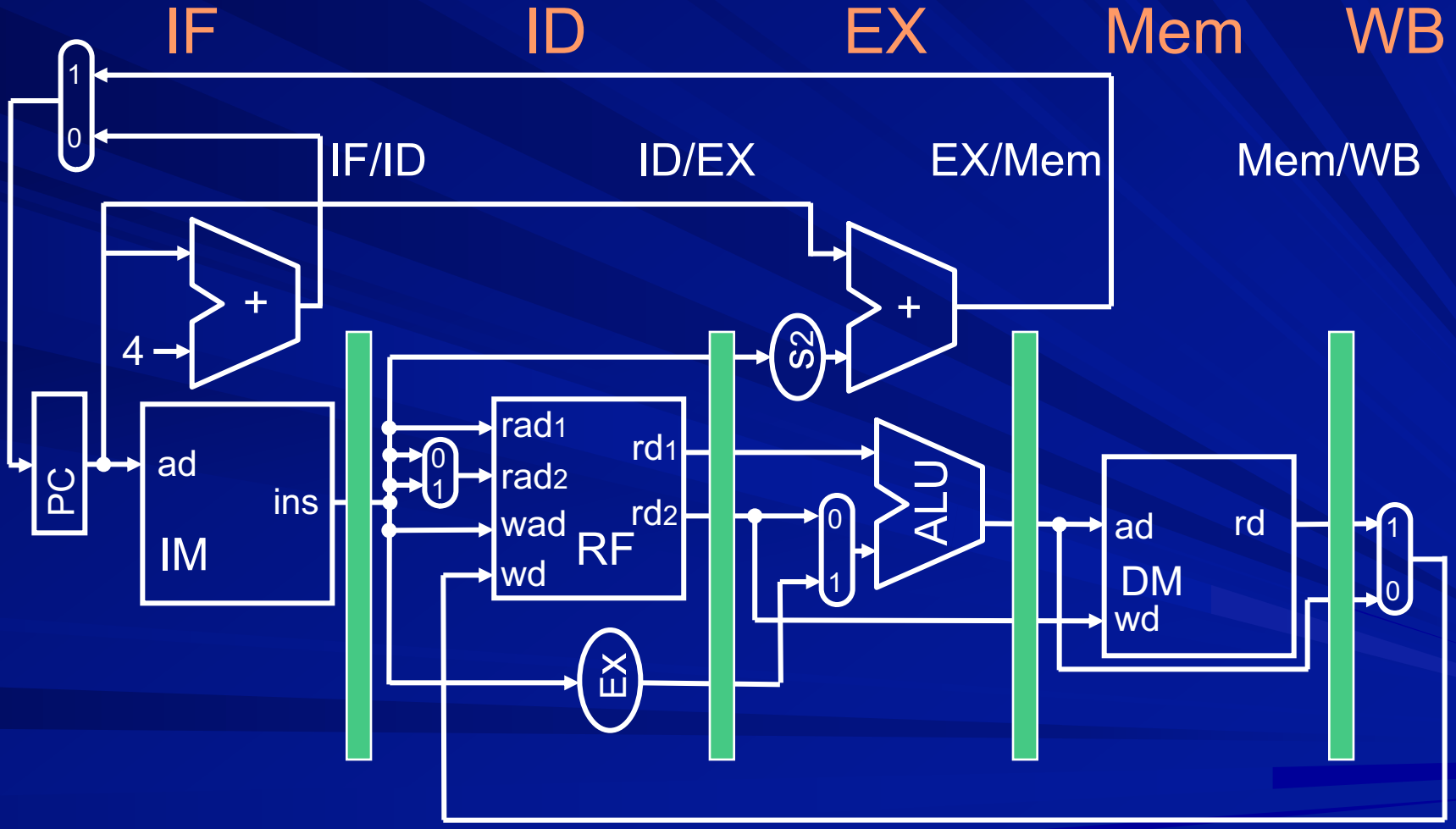
# Single cycle datapath



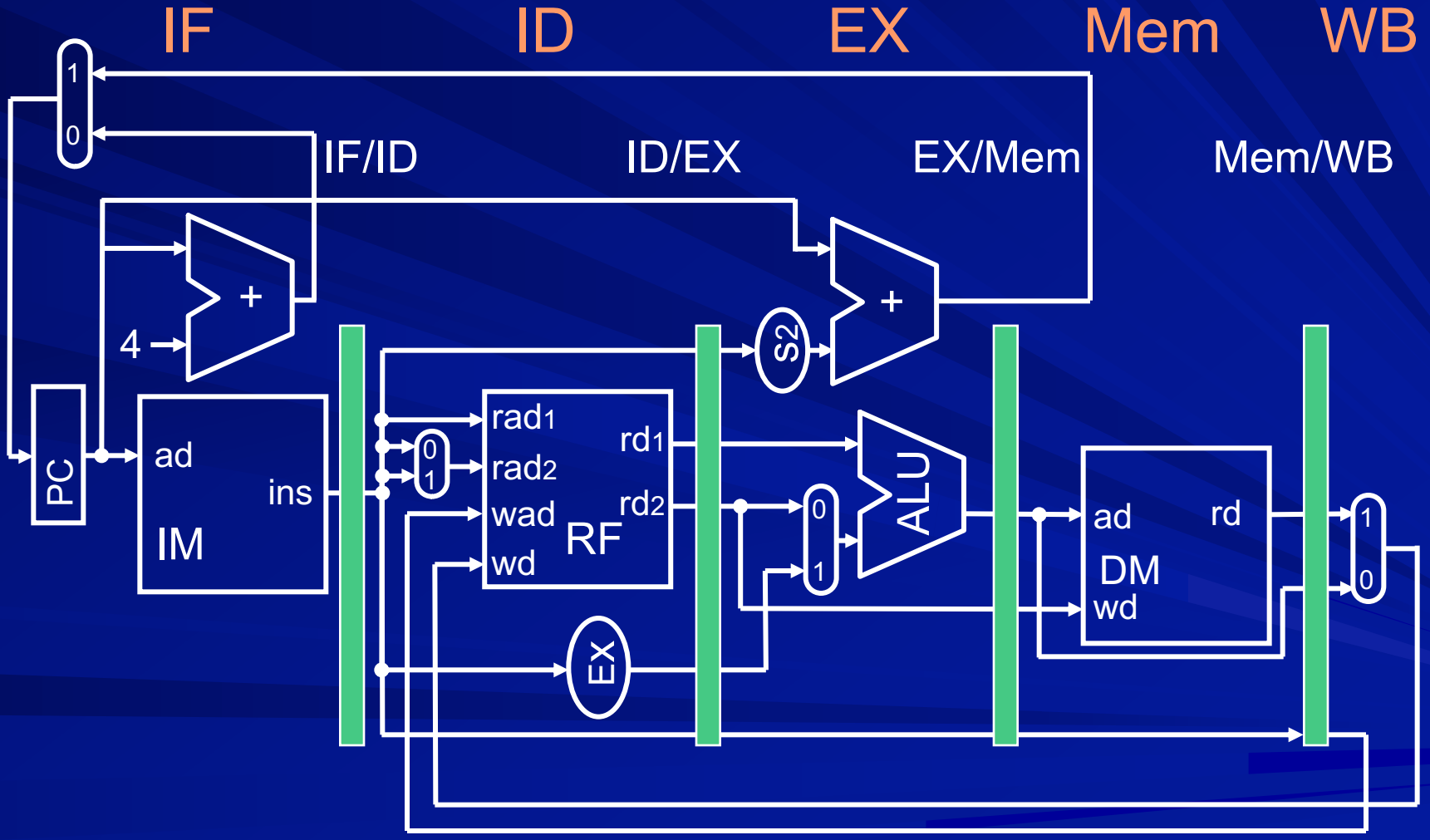
# Pipelined datapath



# Fetch new instruction every cycle

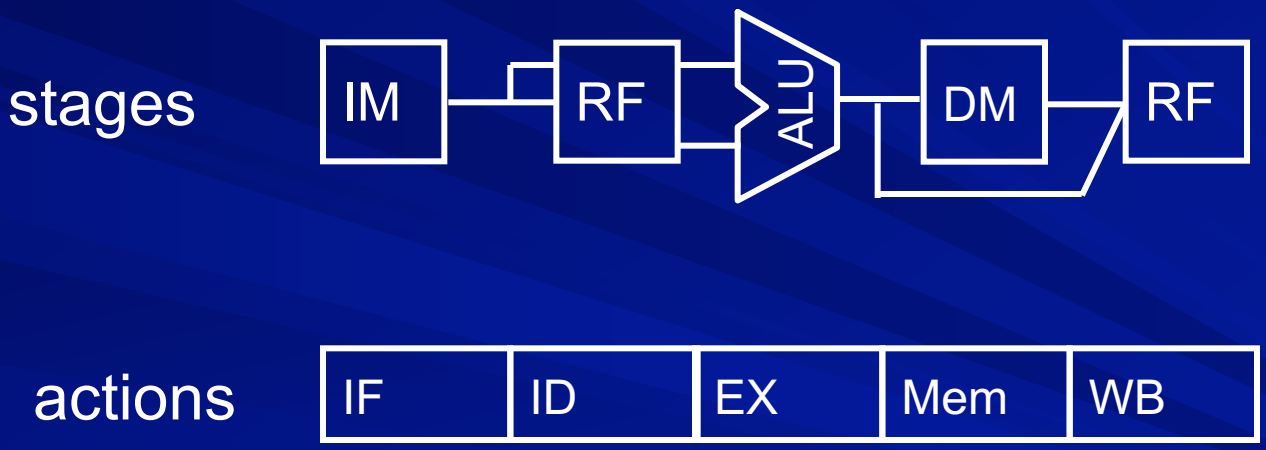


# Correction for WB stage

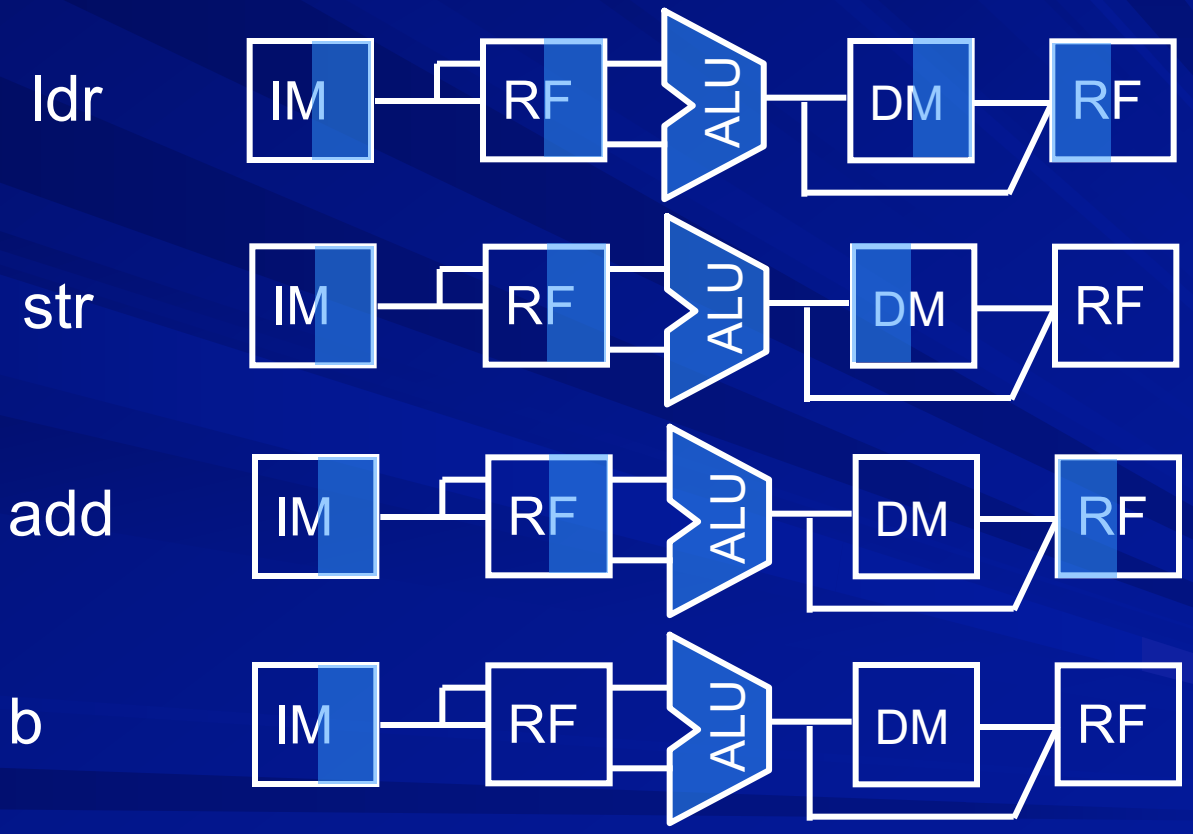


# Graphical representation

5 stage pipeline



# Usage of stages by instructions



# Representing pipelined execution

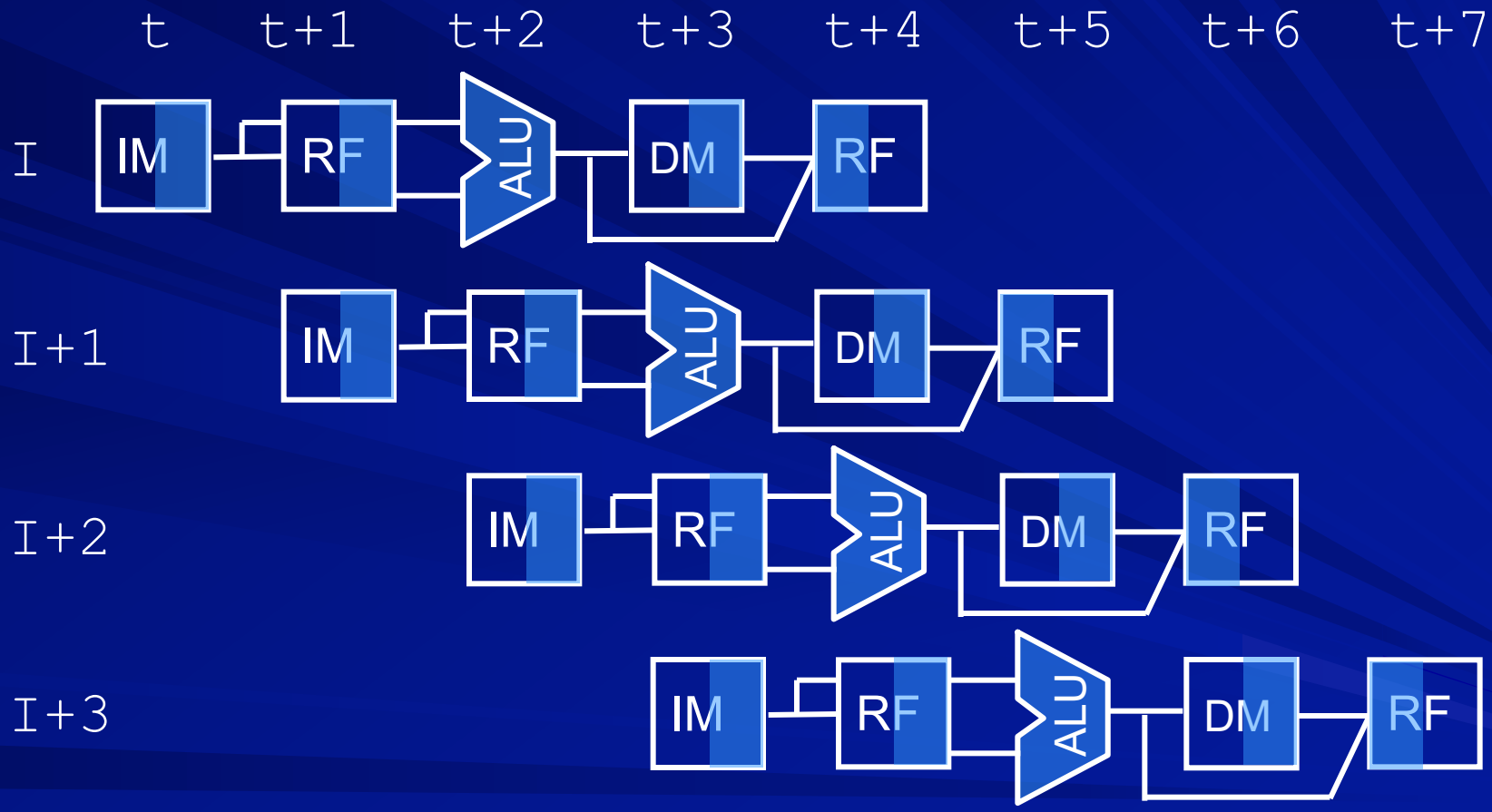
## Representation I

- Horizontal axis: time
- Vertical axis: instructions

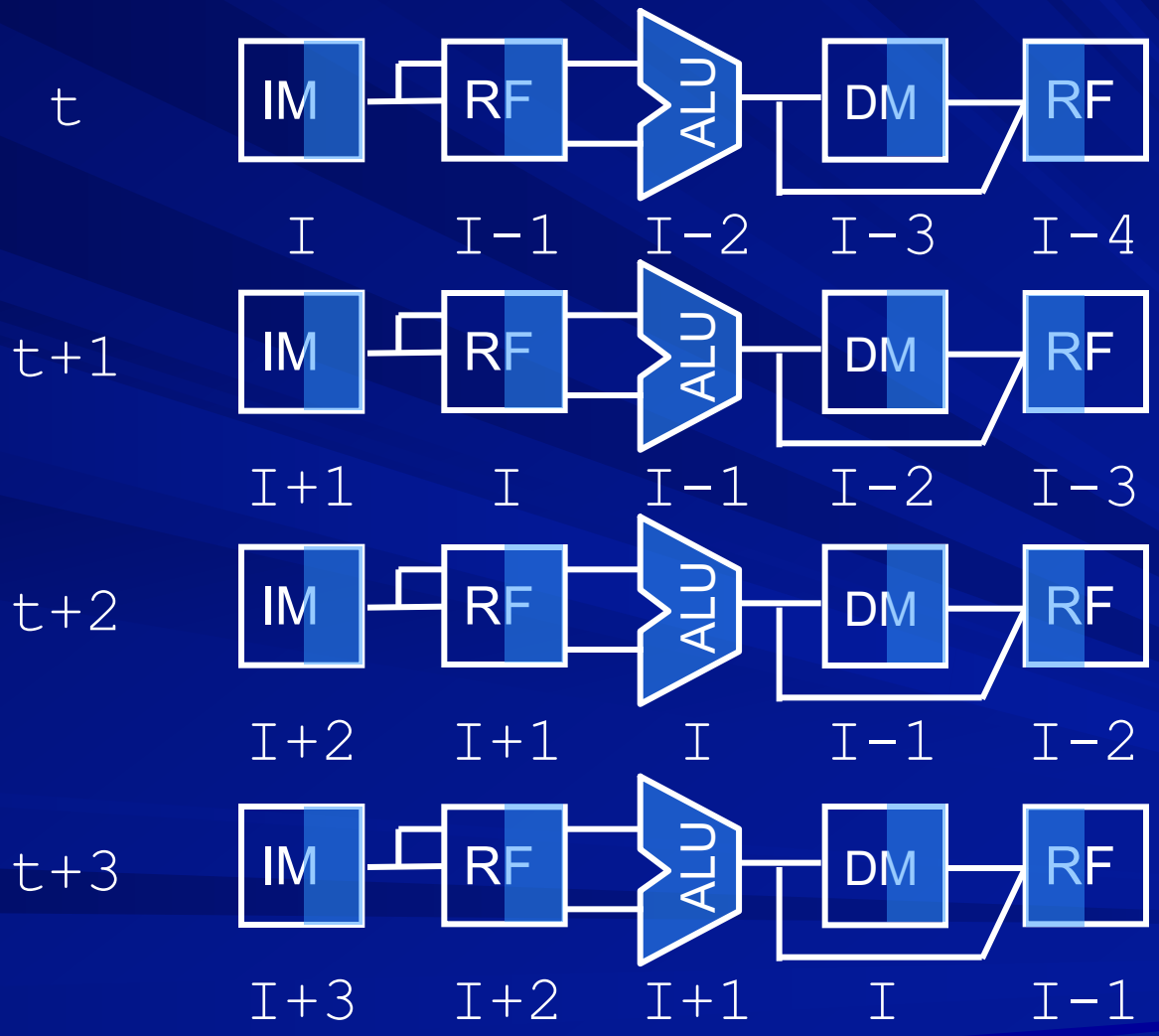
## Representation II

- Horizontal axis: pipeline stages
- Vertical axis: time

# Representation I



# Representation II



# Hurdles in instruction pipelining

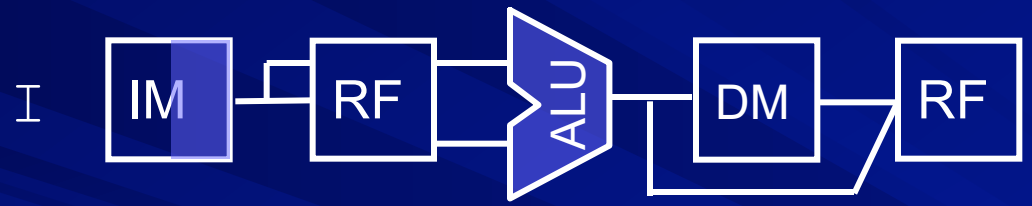
- Structural hazards
  - Resource conflicts - two instruction require same resource in the same cycle
- Data hazards
  - Data dependencies - one instruction needs data which is yet to be produced by another instruction
- Control Hazards
  - Decision about next instruction needs more cycles

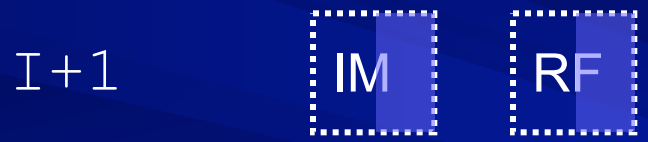
# Structural hazards

- No structural hazards in the present design
  - separate instruction and data memories
  - adders for PC increment and offset addition to PC separate from main ALU
  - each instruction uses ALU at most in one cycle
  - one instruction can read from RF while other can write into it in the same cycle

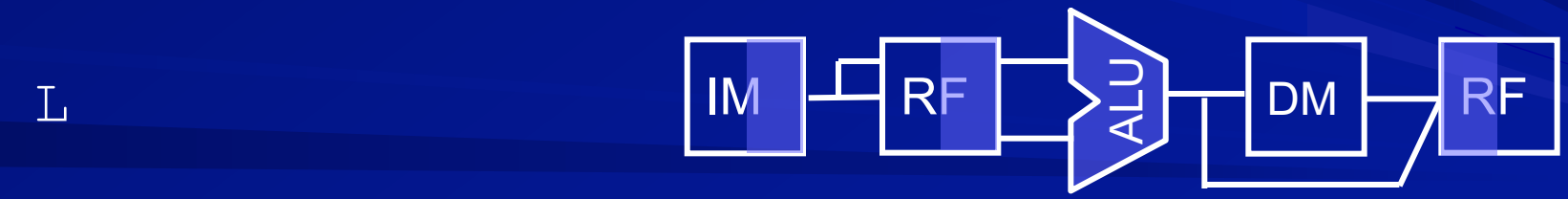
# Stalls due to control hazards

```
I: beq L
   ...
L: add ...
```



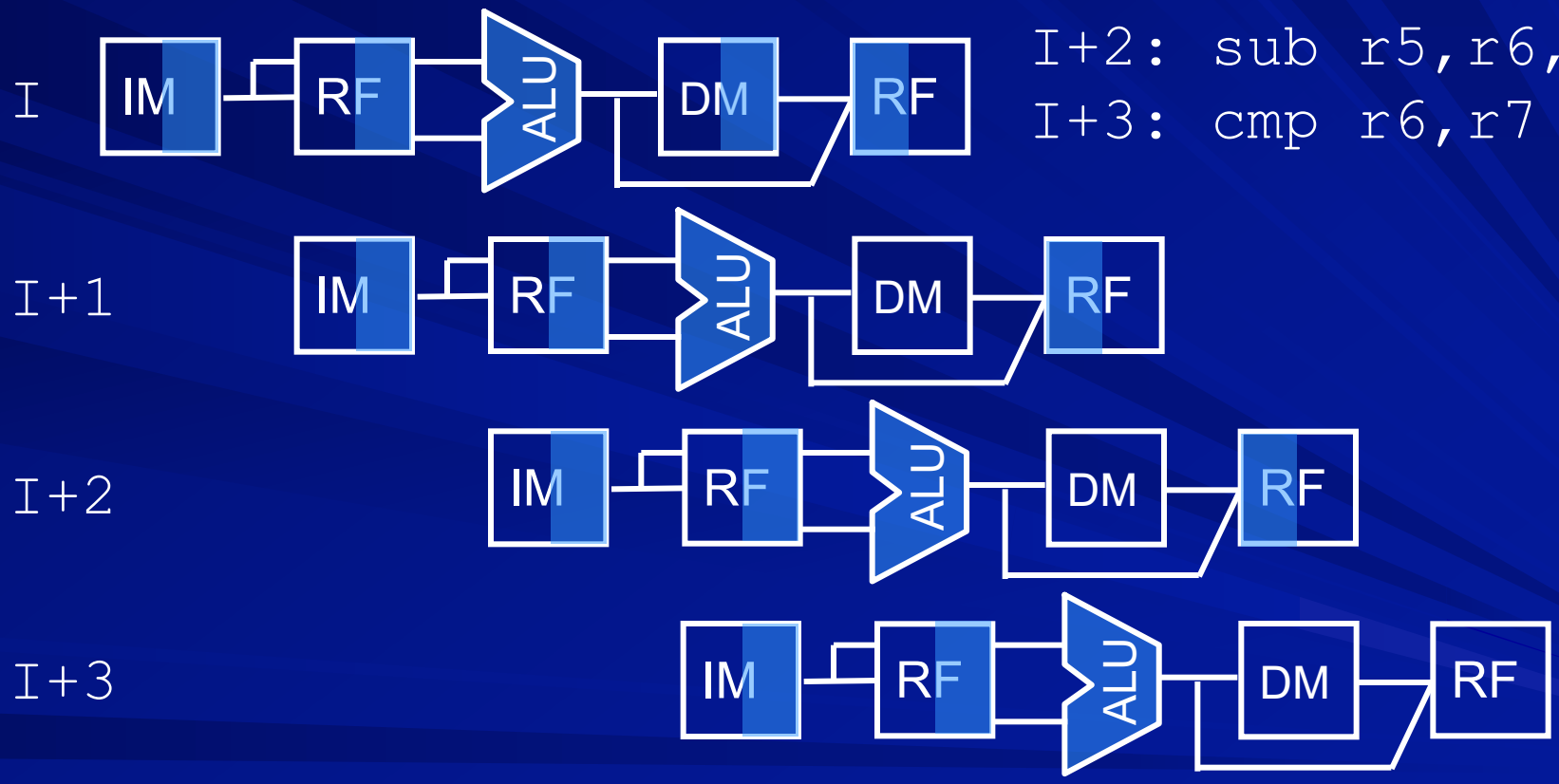







# Data hazards

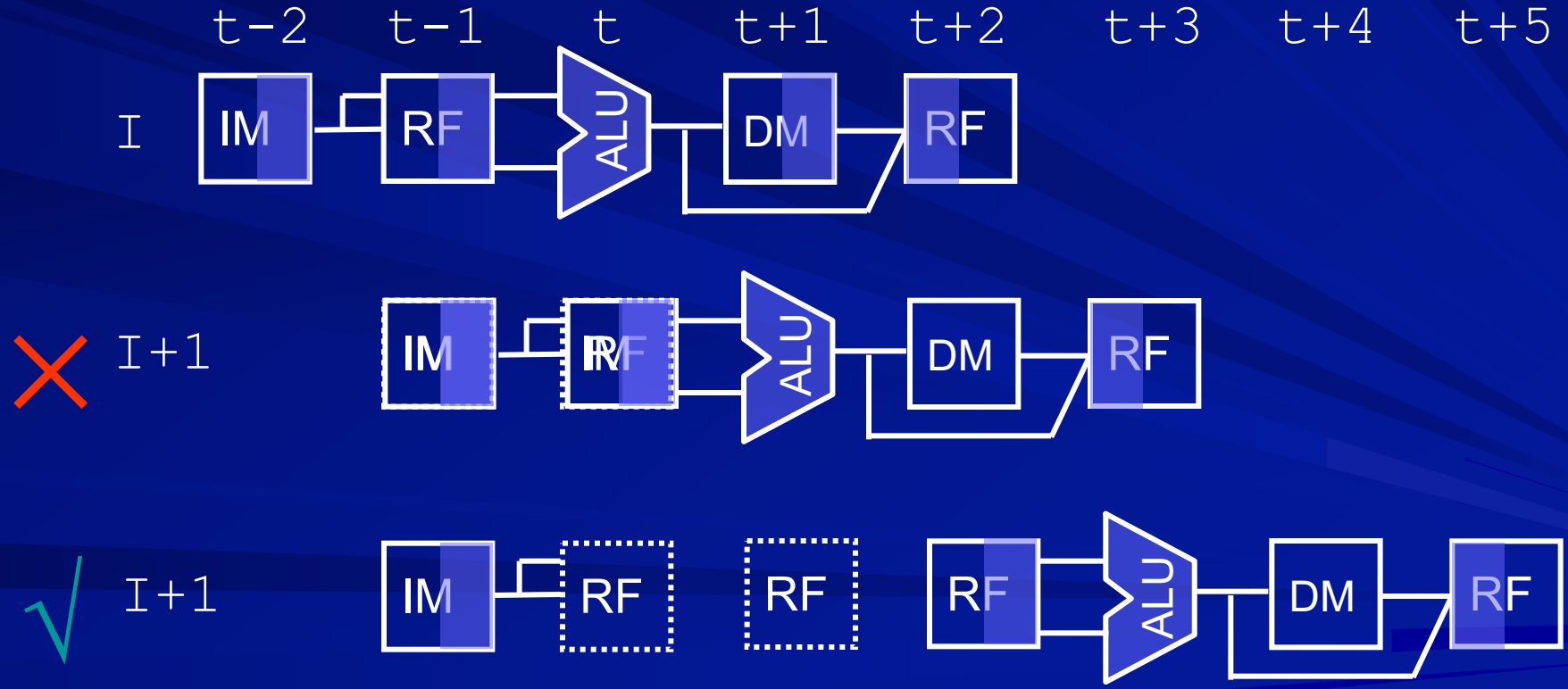
```
I:    ldr r6,...
I+1:  add r4,r6,..
I+2:  sub r5,r6,..
I+3:  cmp r6,r7
```



# Stalls due to data hazards

## instruction view

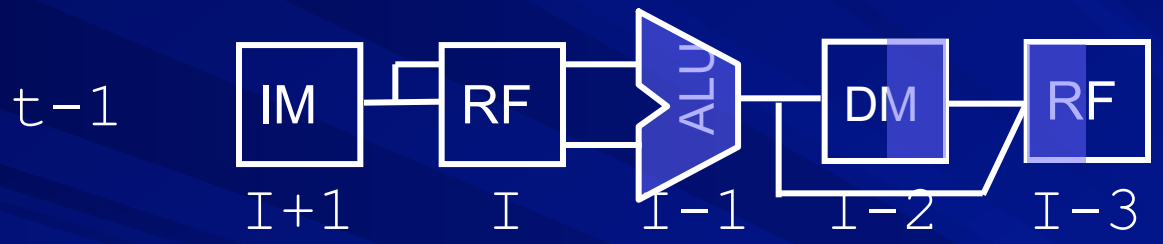
```
I:    ldr r6,...
I+1: add r4,r6,..
```



# Stalls due to data hazards

## stage-wise view

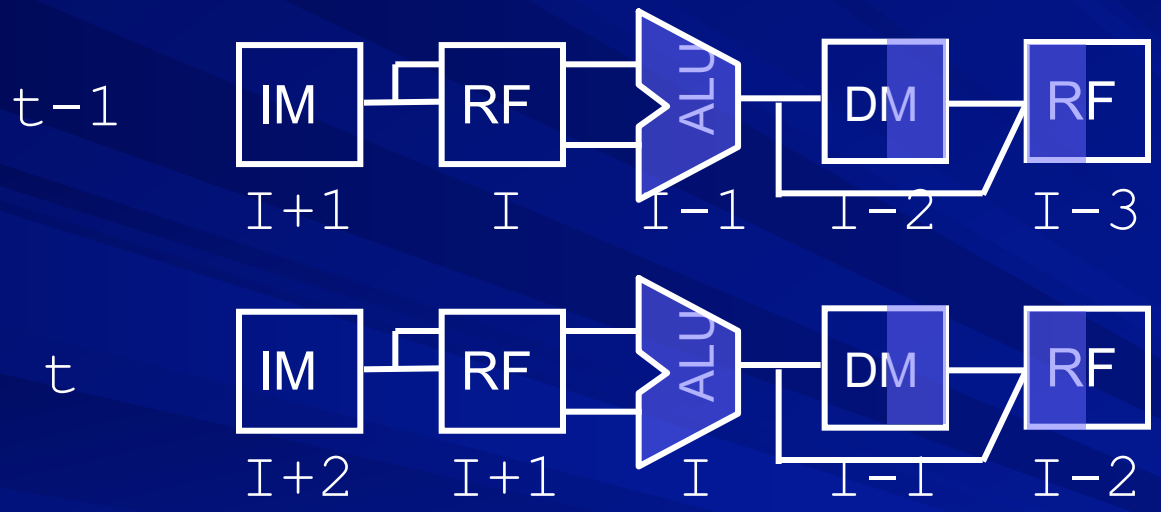
```
I:   ldr r6,...
I+1: add r4,r6,..
```



# Stalls due to data hazards

## stage-wise view

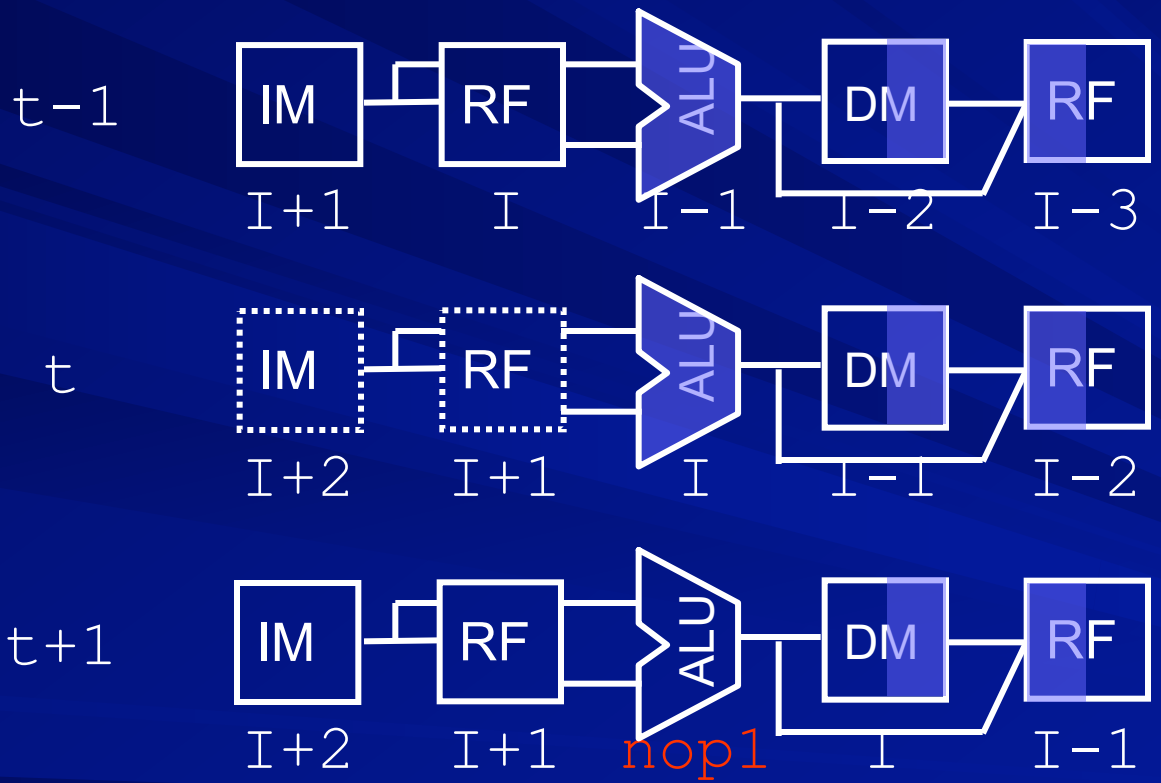
```
I:   ldr r6,...
I+1: add r4,r6,..
```



# Stalls due to data hazards

## stage-wise view

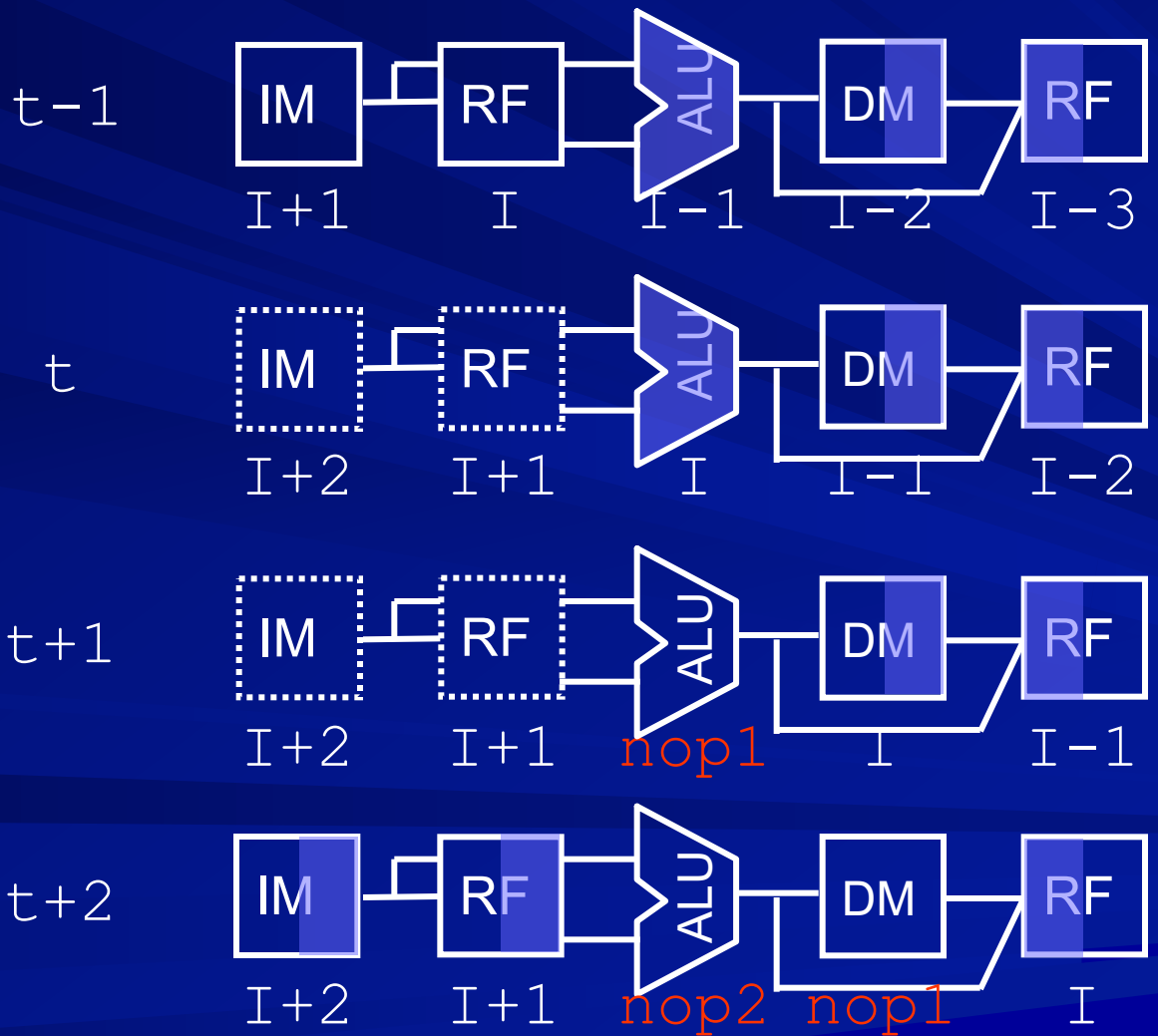
```
I:   ldr r6,...
I+1: add r4,r6,..
```



# Stalls due to data hazards

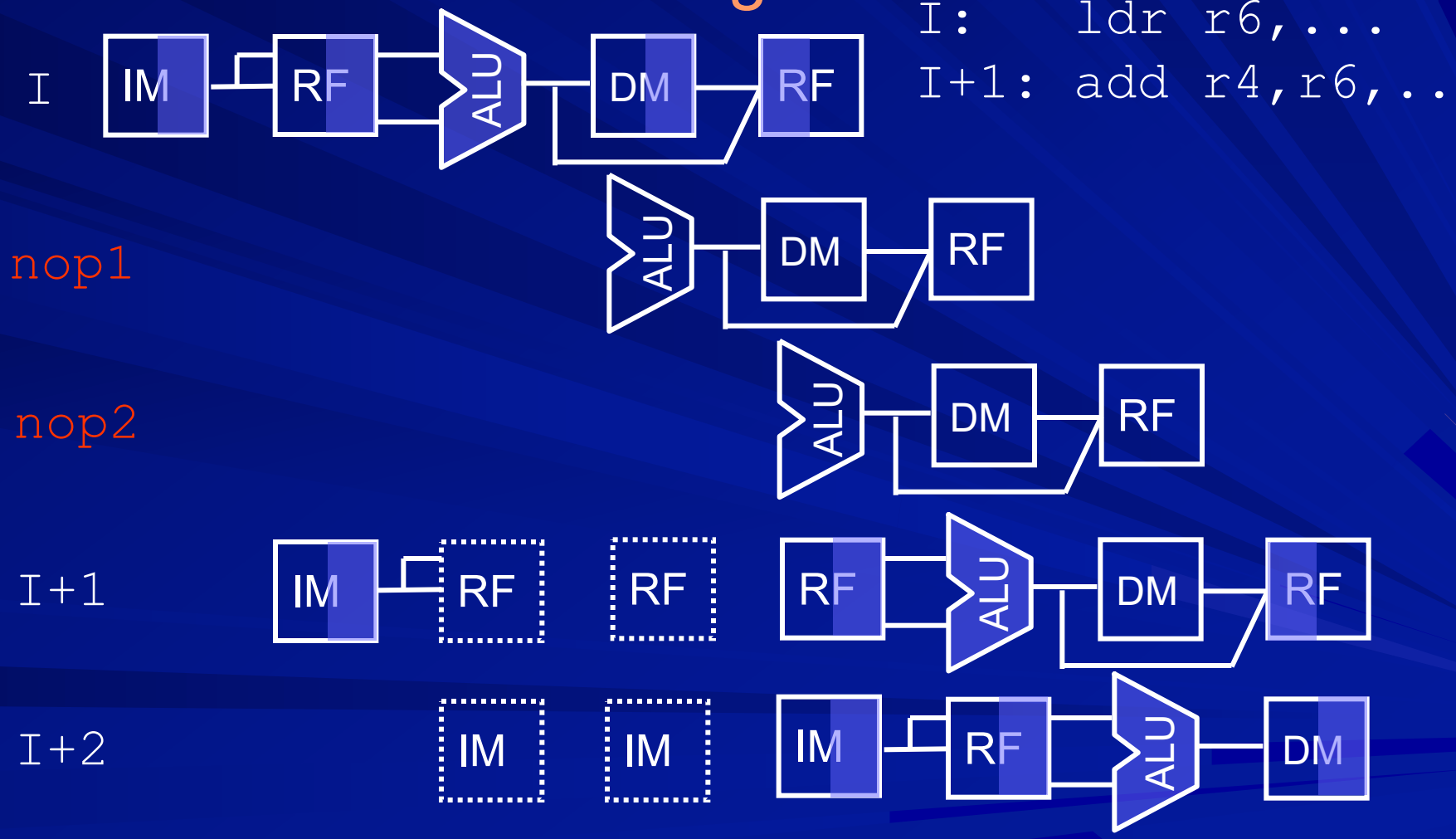
## stage-wise view

```
I:   ldr r6,...
I+1: add r4,r6,..
```



# Stalls due to data hazards

## instruction view again



# Handling hazards

- Data hazards
  - detect instructions with data dependence
  - introduce nop instructions (bubbles) in the pipeline
  - more complex: data forwarding
- Control hazards
  - detect branch instructions
  - flush inline instructions if branching occurs
  - more complex: branch prediction

# Are there software solutions?

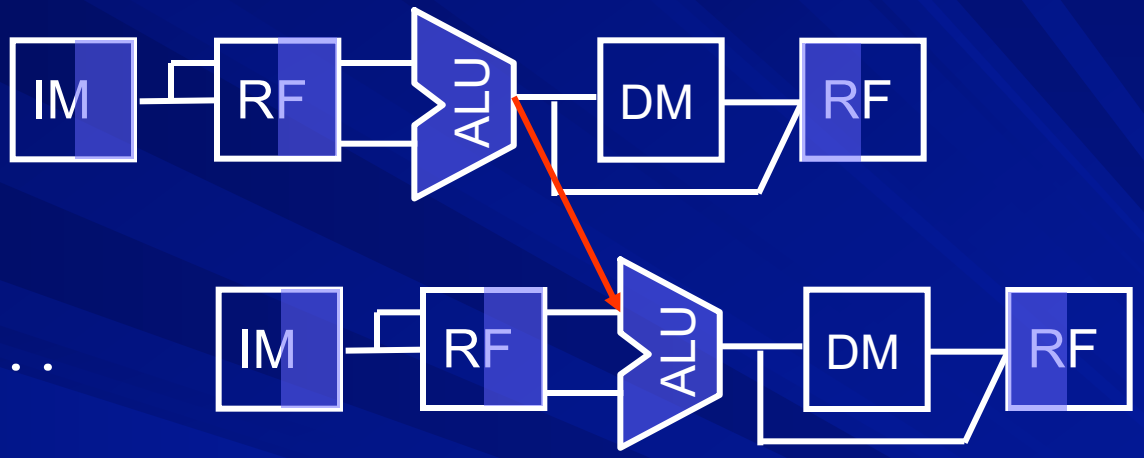
- Separate dependent instructions by reordering code
- Insert nop instructions in worst case

- Treat branches as delayed branches and insert suitable instructions in delay slots

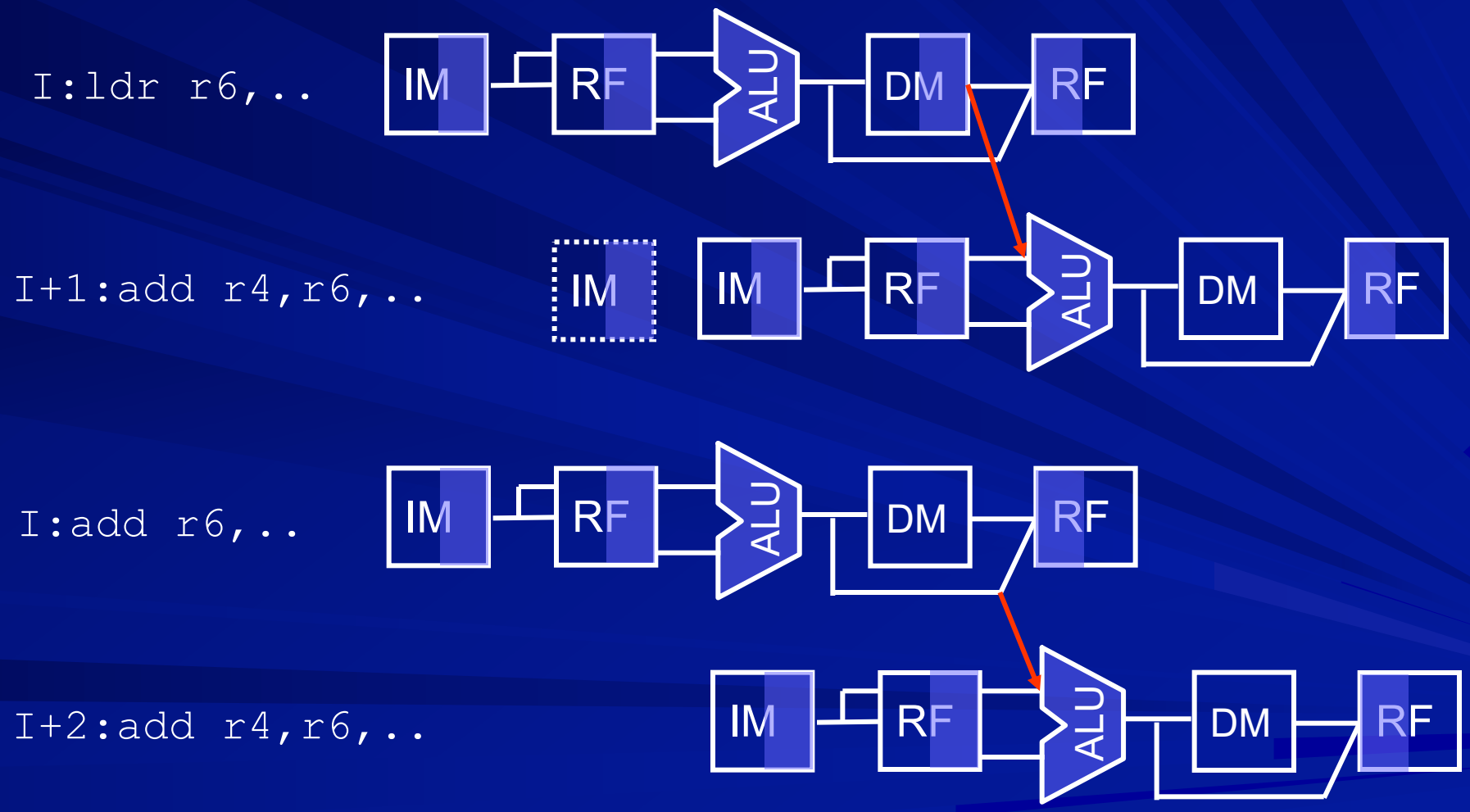
# Data forwarding path P1

I:add r6,..

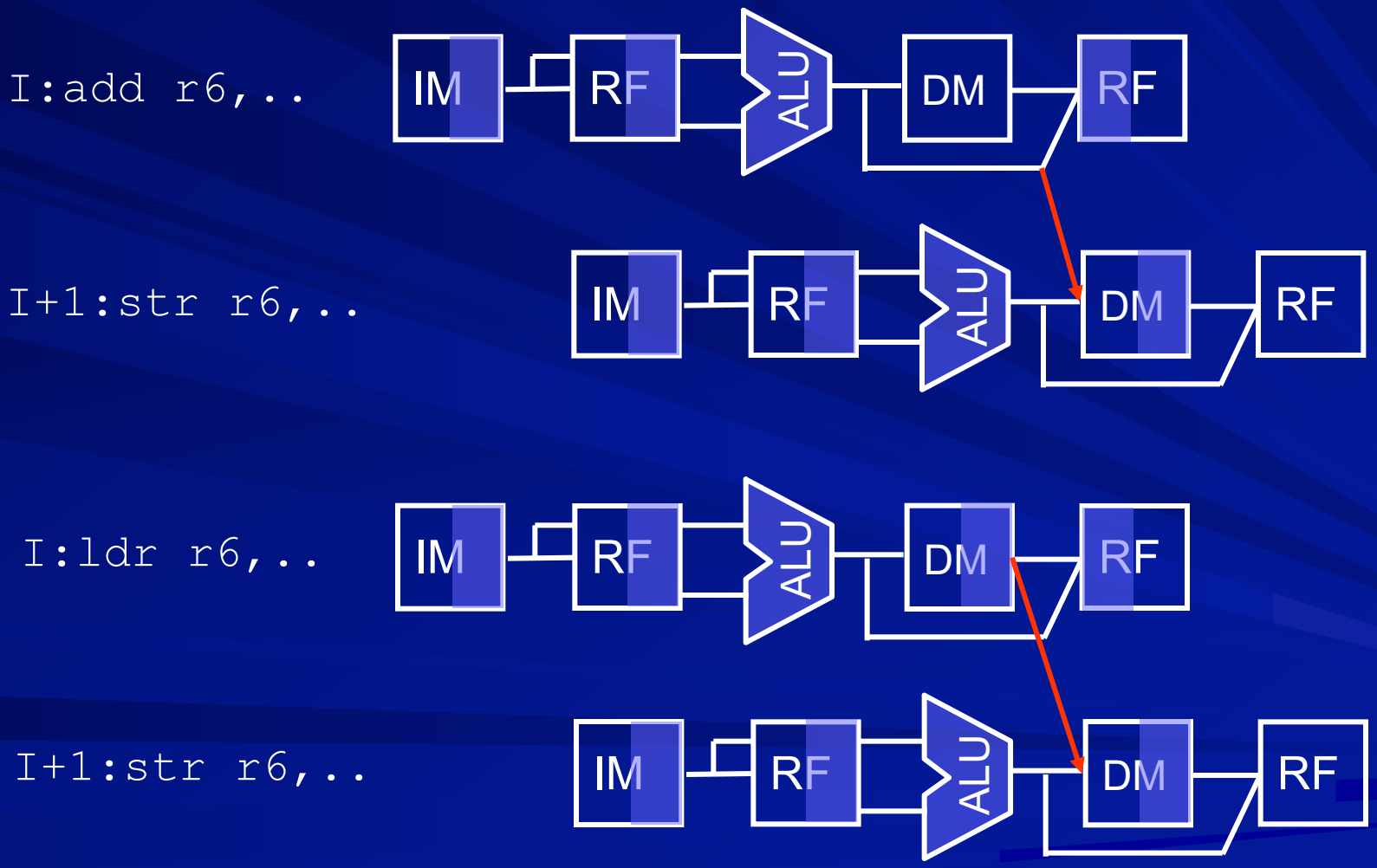
I+1:add r4,r6,..



# Data forwarding path P2



# Data forwarding path P3



# Thanks