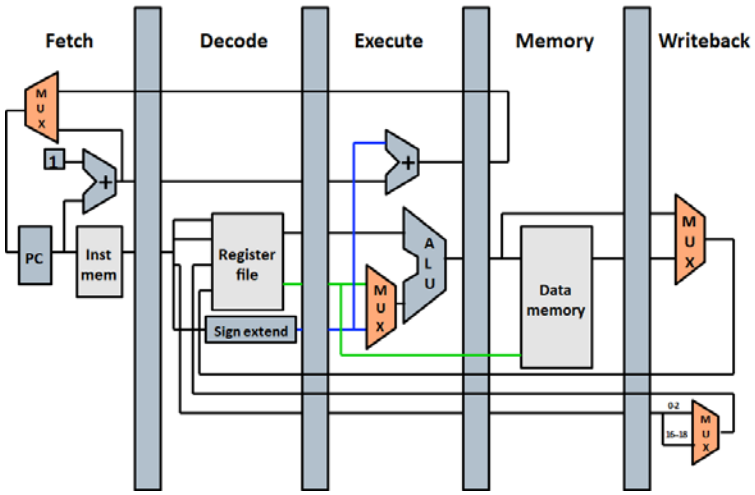
Computer Organization and Design [LC2K-Pipeline]

Pipelined implementation of LC2Kx

- **Break the execution of the instruction into cycles.**
  - Similar to the multi-cycle datapath
- **Design a separate datapath stage for the execution performed during each cycle.**
  - Build pipeline registers to communicate between the stages.

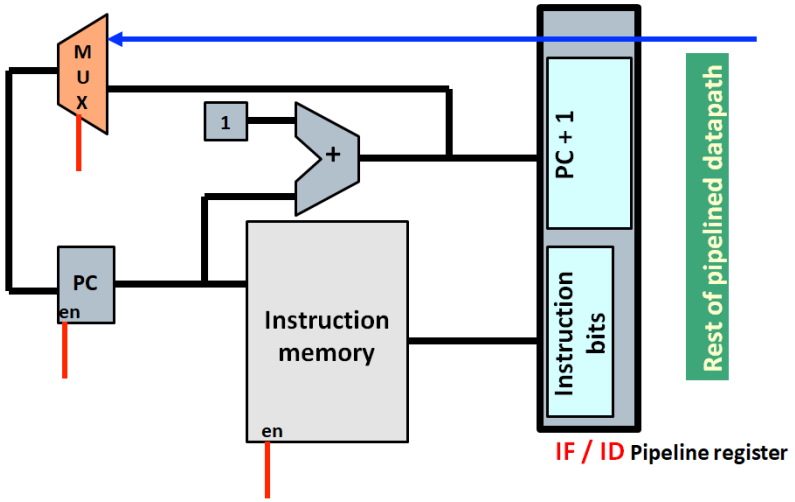
Our new pipelined datapath



Stage 1: Fetch

- **Design a datapath that can fetch an instruction from memory every cycle.**
  - Use PC to index memory to read instruction
  - Increment the PC (assume no branches for now)
- **Write everything needed to complete execution to the pipeline register (IF/ID)**
  - The next stage will read this pipeline register.
  - Note that pipeline register must be edge-triggered

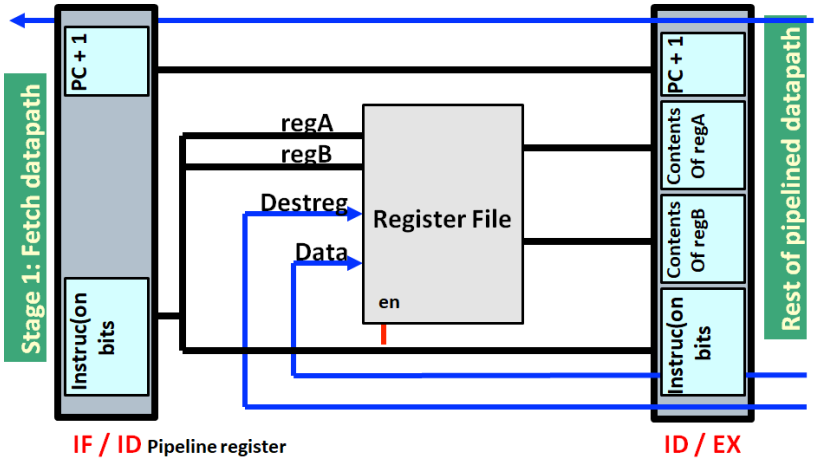
Pipeline datapath – Fetch stage



Stage 2: Decode

- **Design a datapath that reads the IF/ID pipeline register, decodes instruction and reads register file (specified by regA and regB of instruction bits).**
  - Decode is easy, just pass on the opcode and let later stages figure out their own control signals for the instruction.
- **Write everything needed to complete execution to the pipeline register (ID/EX)**
  - Pass on the offset field and both destination register specifiers (or simply pass on the whole instruction!).
  - Including PC+1 even though decode didn't use it.

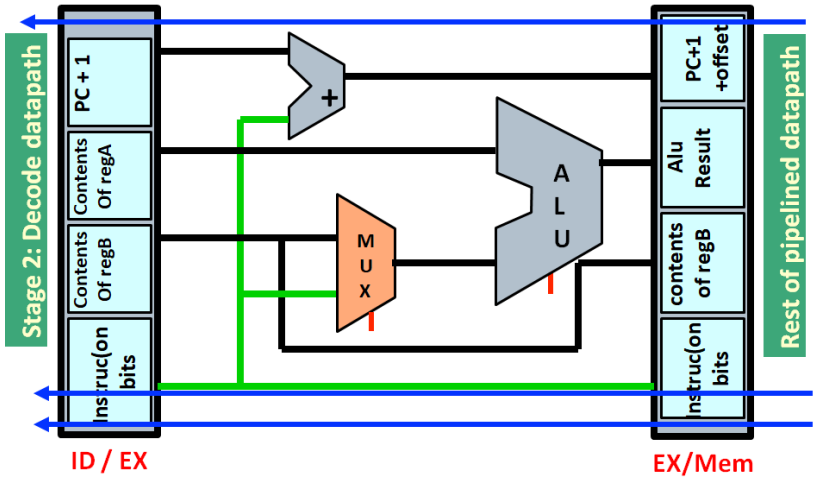
Pipeline datapath – Decode stage



Stage 3: Execute

- **Design a datapath that performs the proper ALU operation for the instruction specified and the values present in the ID/EX pipeline register.**
  - The inputs are the contents of regA and either the contents of regB or the offset field on the instruction.
  - Also, calculate PC+1+offset in case this is a branch.
- **Write everything needed to complete execution to the pipeline register (EX/Mem)**
  - ALU result, contents of regB and PC+1+offset
  - Instruction bits for opcode and destReg specifiers
  - Result from comparison of regA and regB contents

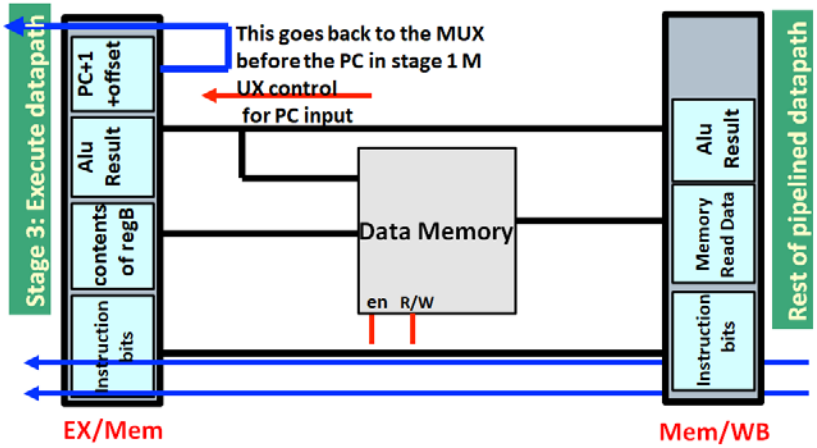
Pipeline datapath – Execute stage



Stage 4: Memory Operation

- **Design a datapath that performs the proper memory operation for the instruction specified and the values present in the EX/Mem pipeline register.**
  - ALU result contains address for ld and st instructions.
  - Opcode bits control memory R/W and enable signals.
- **Write everything needed to complete execution to the pipeline register (Mem/WB)**
  - ALU result and MemData
  - Instruction bits for opcode and destReg specifiers

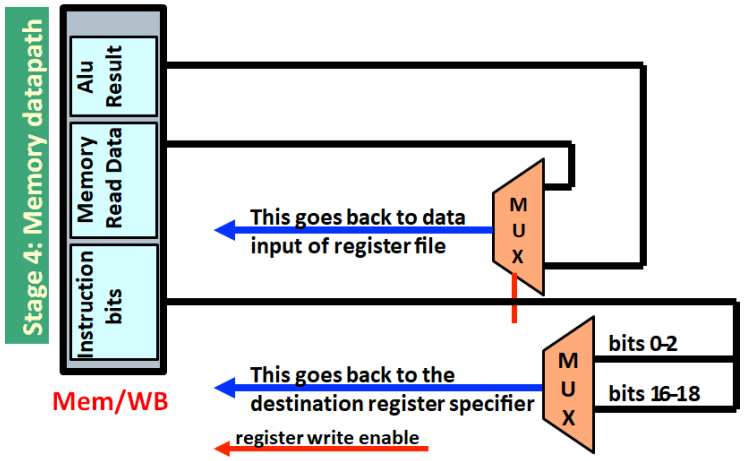
Pipeline datapath – Memory stage



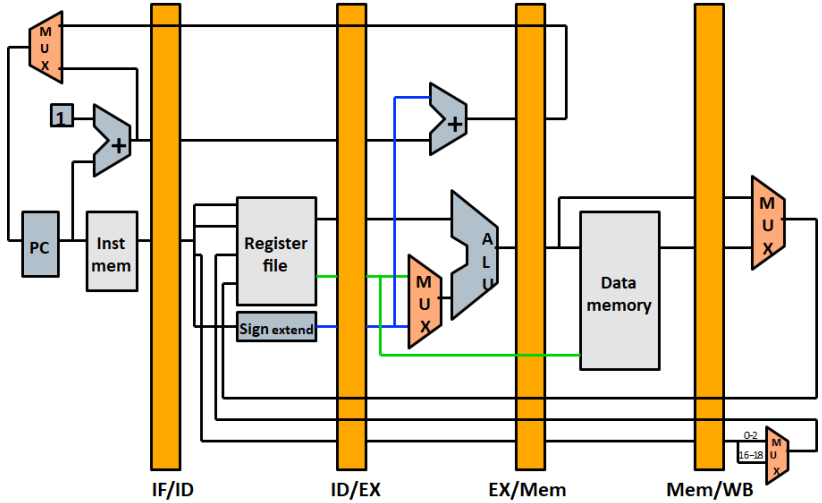
Stage 5: Write back

- **Design a datapath that completes the execution of this instruction, writing to the register file if required.**
  - Write MemData to destReg for ld instruction
  - Write ALU result to destReg for add or nand instructions.
  - Opcode bits also control register write enable signal.

Pipeline datapath – Writeback stage



putting all together

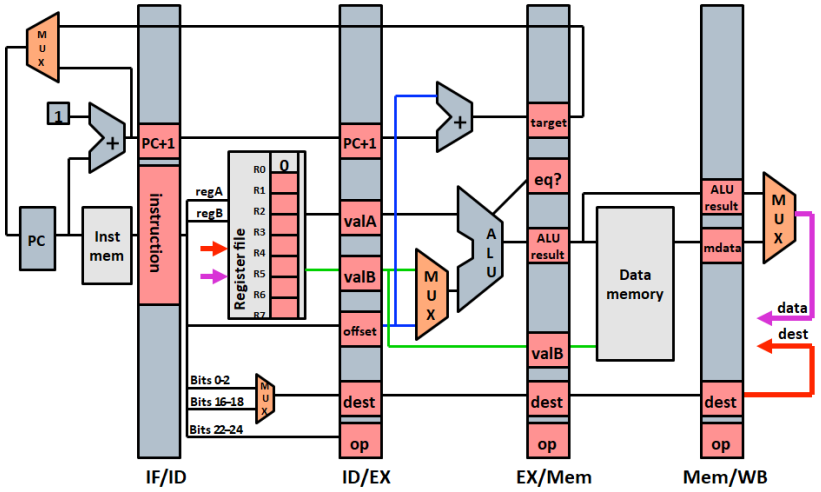


Sample Code (Simple)

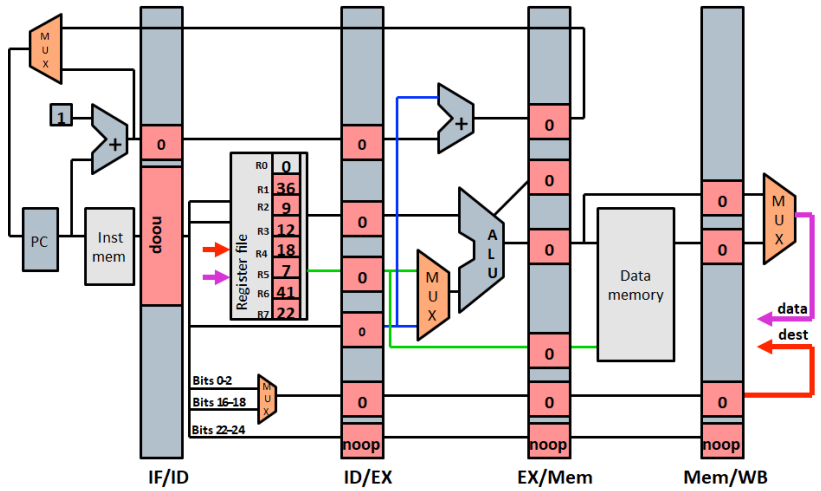
Let's run the following code on pipelined LC2K2x:

- add 1 2 3 ; reg 3 = reg 1 + reg 2
- nor 4 5 6 ; reg 6 = reg 4 nor reg 5
- lw 2 4 20 ; reg 4 = Mem[reg2+20]
- add 2 5 5 ; reg 5 = reg 2 + reg 5
- sw 3 7 10 ; Mem[reg3+10] =reg 7

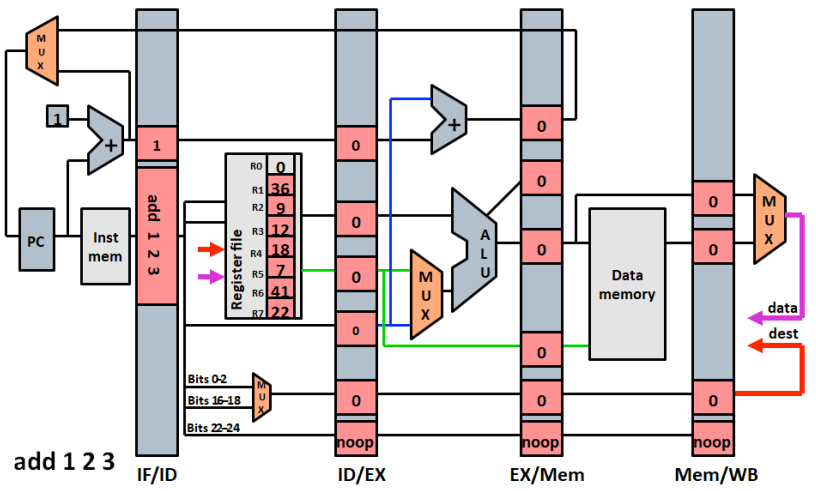
Pipeline datapath



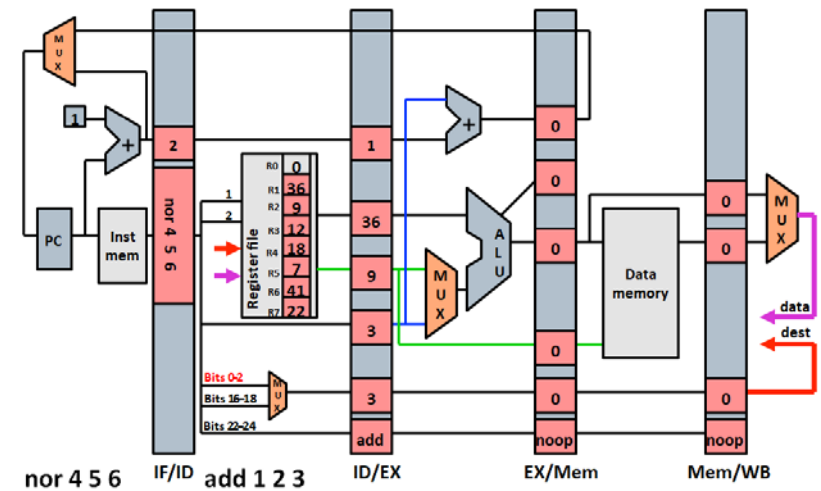
Time 0 -Initial state



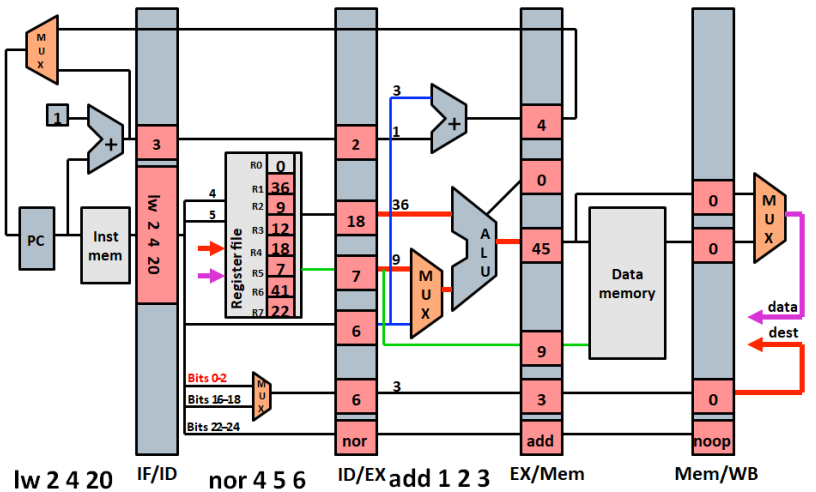
Time 1 – Fetch: add 1 2 3



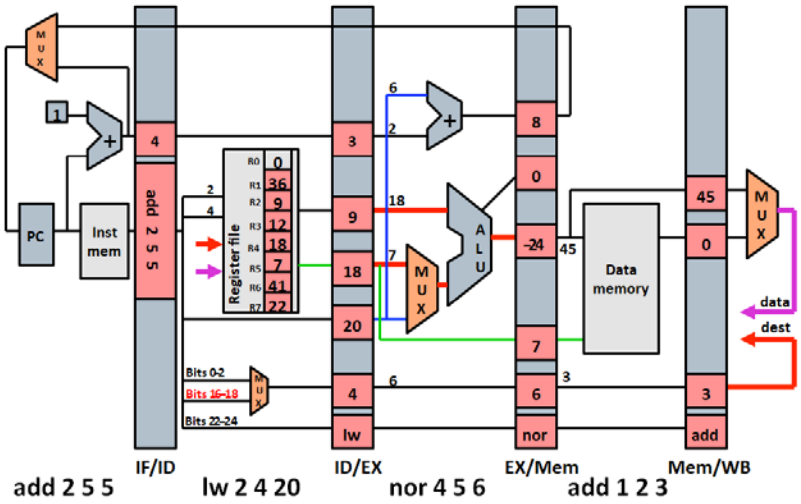
Time 2 – Fetch: nor 4 5 6



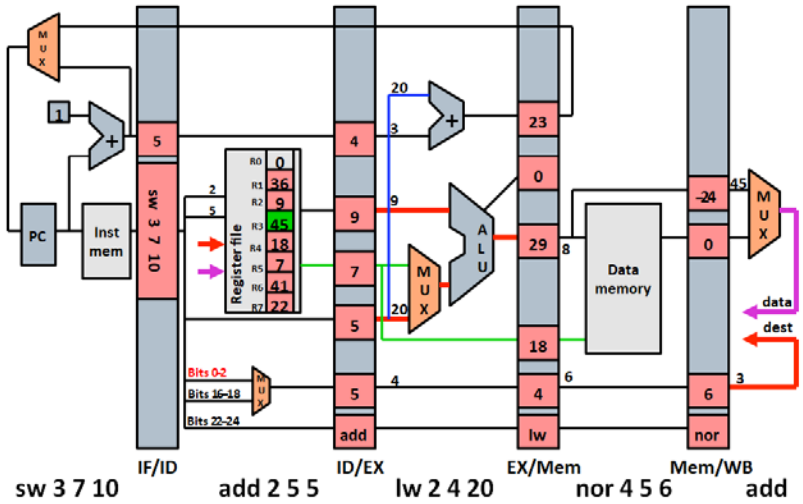
Time 3 – Fetch: lw 2 4 20



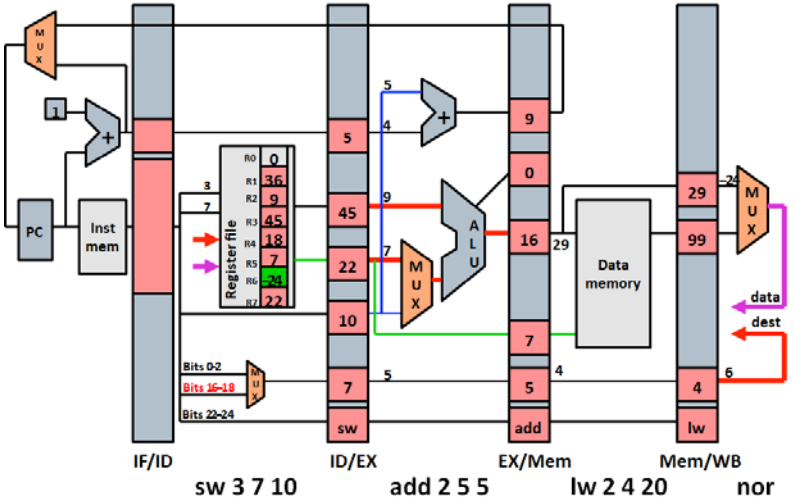
Time 4 – Fetch: add 2 5 5



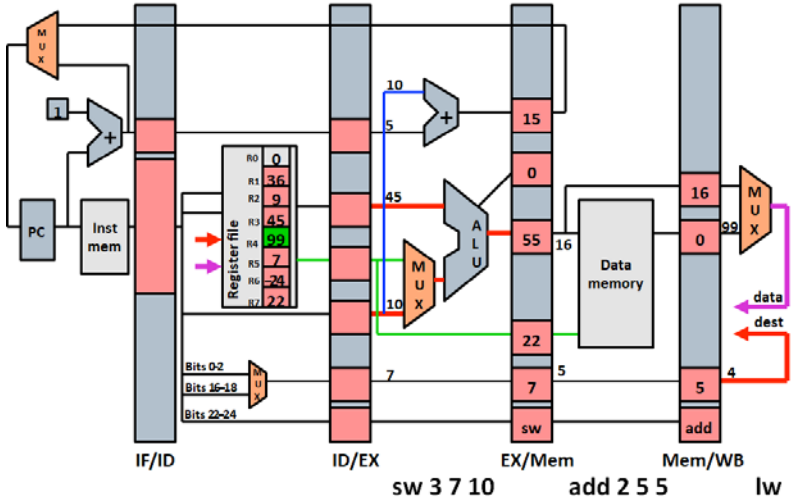
Time 5 – Fetch: sw 3 7 10



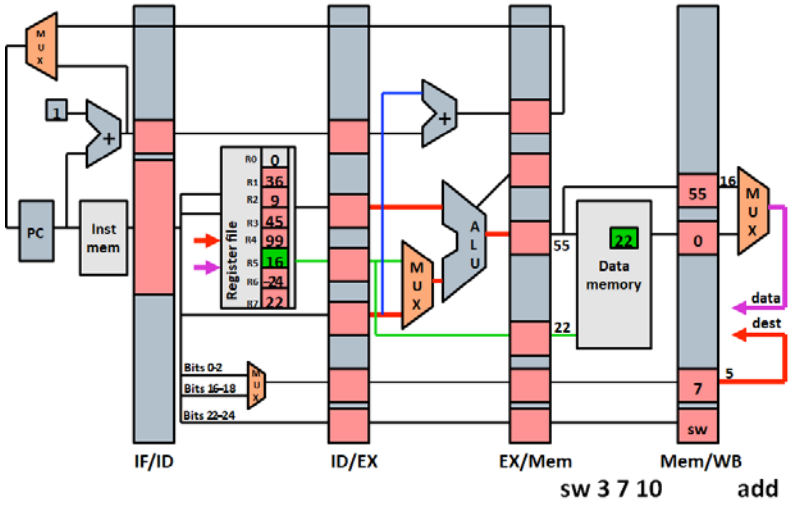
Time 6 – no more instructions



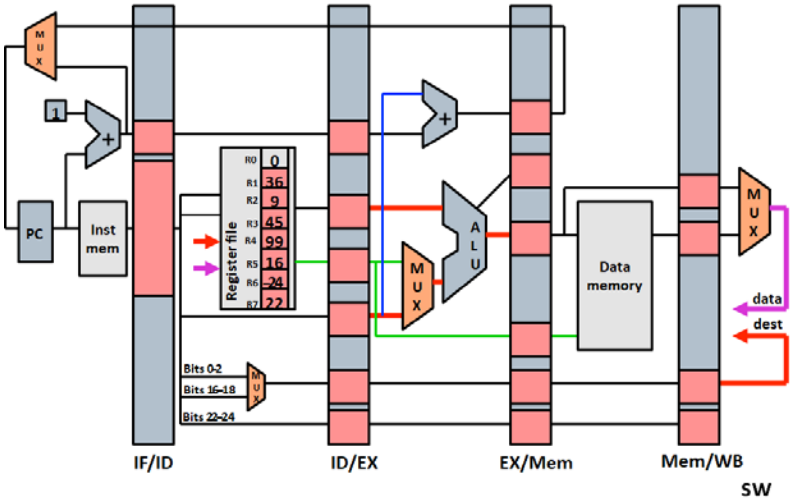
Time 7 – no more instructions



Time 8 – no more instructions



Time 9 – no more instructions



Time graphs (a.k.a. pipe trace)

Time:	1	2	3	4	5	6	7	8	9
add	fetch	decode	execute	memory	writeback				
nor		fetch	decode	execute	memory	writeback			
lw			fetch	decode	execute	memory	writeback		
add				fetch	decode	execute	memory	writeback	
sw					fetch	decode	execute	memory	writeback

A vertical slice reports the entire activity of the pipeline at time 5