

12. A Gentle Introduction to Pipelining

EECS 370 – Introduction to Computer Organization – Winter 2009

Prof. Todd Austin & Prof. Marios Papaefthymiou

EECS Department
University of Michigan, Ann Arbor, USA

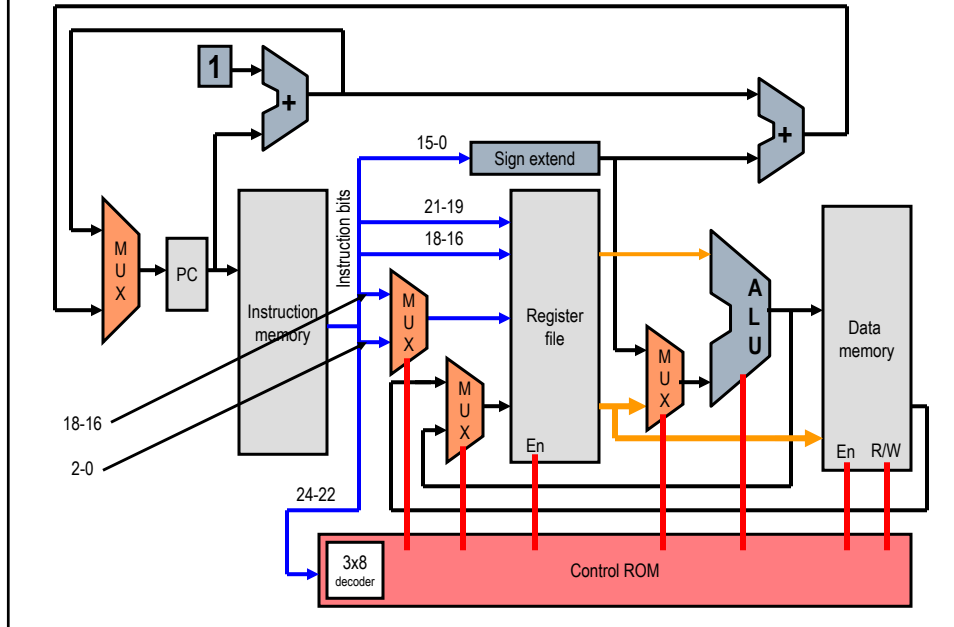
© Austin & Papaefthymiou, 2009

The material in this presentation cannot be
copied in any form without our written permission

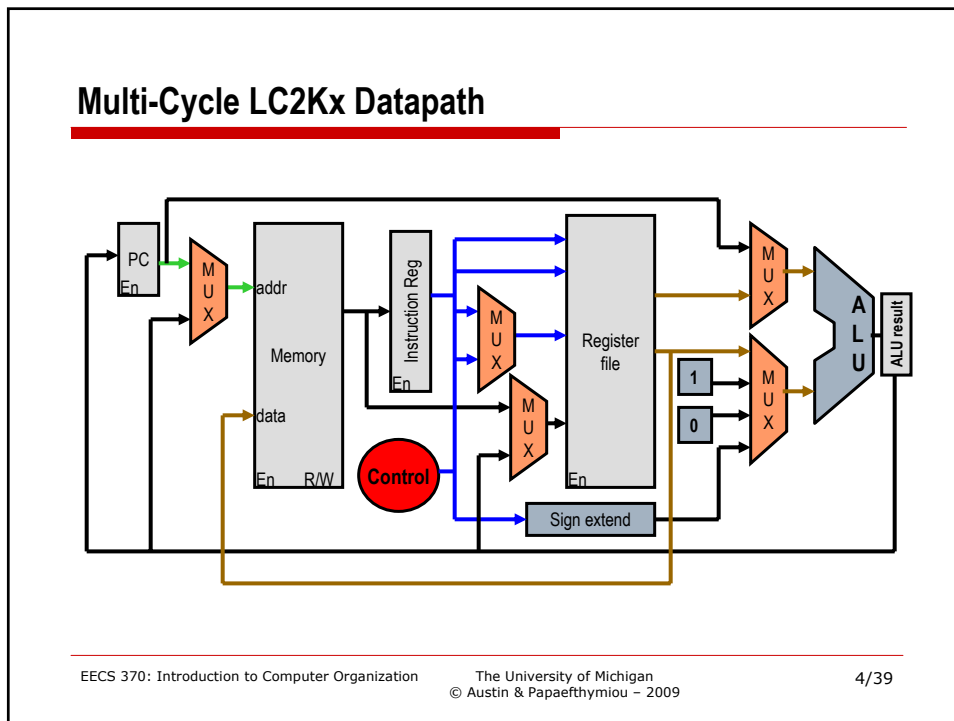
Announcements

- 1. Exam 1: Back Thursday**
- 2. HW4 coming out today**
- 3. PA2 due Friday 3/6**
- 4. PS3 already posted; due 3/27**
- 5. Programming contest: Submit your entries by Friday 3/6**

Single-Cycle LC2Kx Datapath



Multi-Cycle LC2Kx Datapath

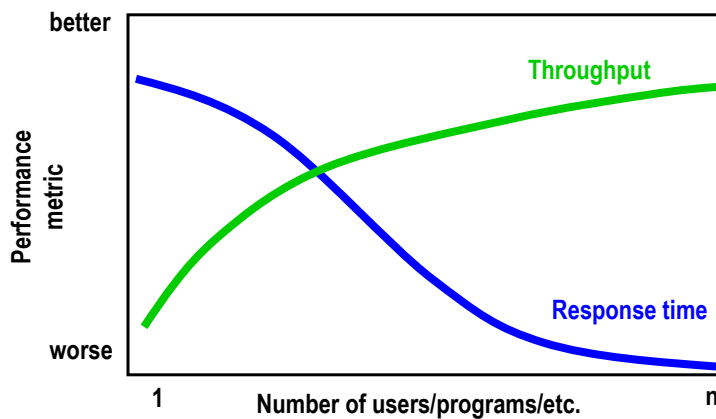


Performance Metrics

- 1. Response time:** when is my job done (time)?
 - When will my books arrive from amazon.com?
 - How long will this program/instruction take?
- 2. Throughput:** how much work can get done within a specified time (work/time)?
 - How many books will amazon.com sell this week?
 - How many programs/instructions complete per hour?
 - Improved relatively easily by using multiprocessors.

Relating response time to throughput

More throughput \leftrightarrow lower response time (*Little's law*)



Performance Metrics – Execution Time

- Response time for a program is its **execution time**

Execution time (for an application):

= total instructions executed x CPI x clock period

- CPI = **avg** number of clock cycles per instruction *for an application*
- For multi-cycle processor implementations we need:
 - Cycles necessary for each type of instruction
 - Mix of instructions executed in the application (dynamic instruction execution profile)

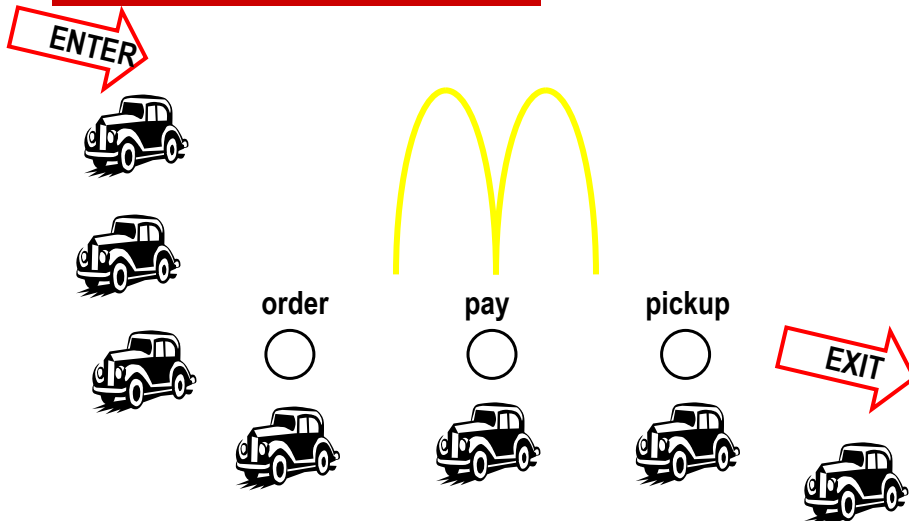
How far have we come?

- Single-cycle processor implementations
 - CPI = ?
 - clock period = ?
- Multi-cycle processor implementations
 - CPI = ?
 - clock period = ?
- Next step: improve CPI without impacting clock period
 - The easiest thing to do is to work on multiple instructions at the same time.

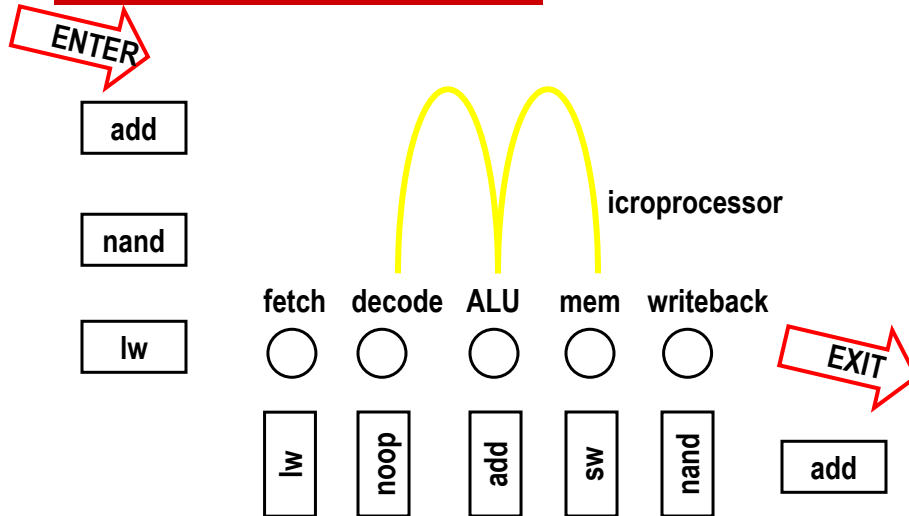
Pipelining

- ❑ Want to execute an instruction?
 - Build a processor (multi-cycle)
 - Find instructions
 - Line up instructions (1, 2, 3, ...)
 - Overlap execution
 - Cycle #1: Fetch 1
 - Cycle #2: Decode 1 Fetch 2
 - Cycle #3: ALU 1 Decode 2 Fetch 3
 -
 - This is called pipelining instruction execution.
 - Used extensively for the first time on IBM 360 (1960s).
 - CPI approaches 1.

Pipelining



Pipelining



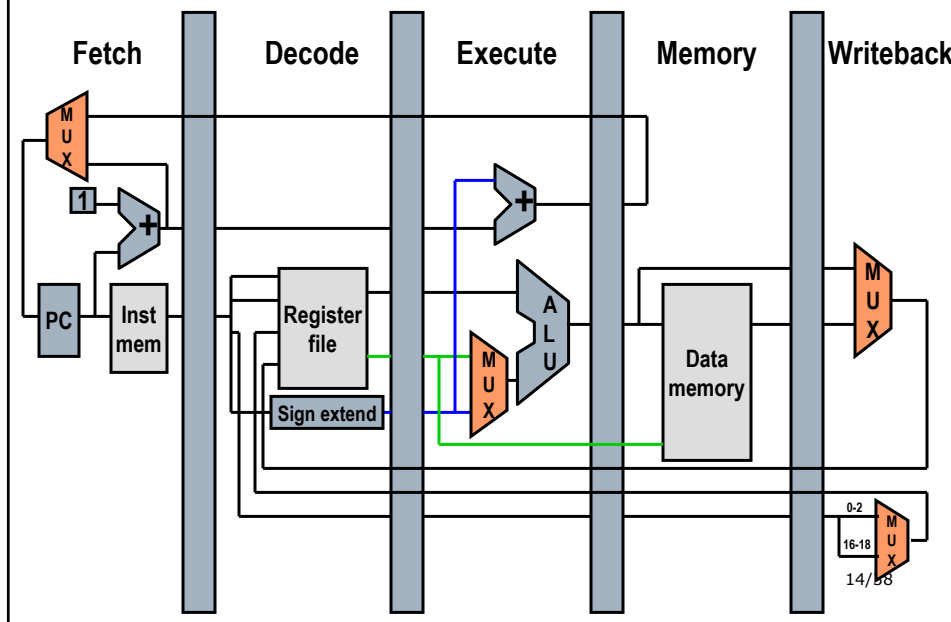
Pipelining Today

- ❑ Execute as many instructions at the same time as possible.
 - Pipelining: 12-20+ cycles
 - Multiple pipelines
- ❑ Pentium:
 - 2 pipelines, 5 cycles each (10 instructions “in flight”)
- ❑ Pentium Pro/II/III
 - 3 pipelines (kinda), 12 cycles each (kinda)
 - Instructions can execute out of their original program order
- ❑ Pentium IV
 - 4 pipelines, 20 cycles deep

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

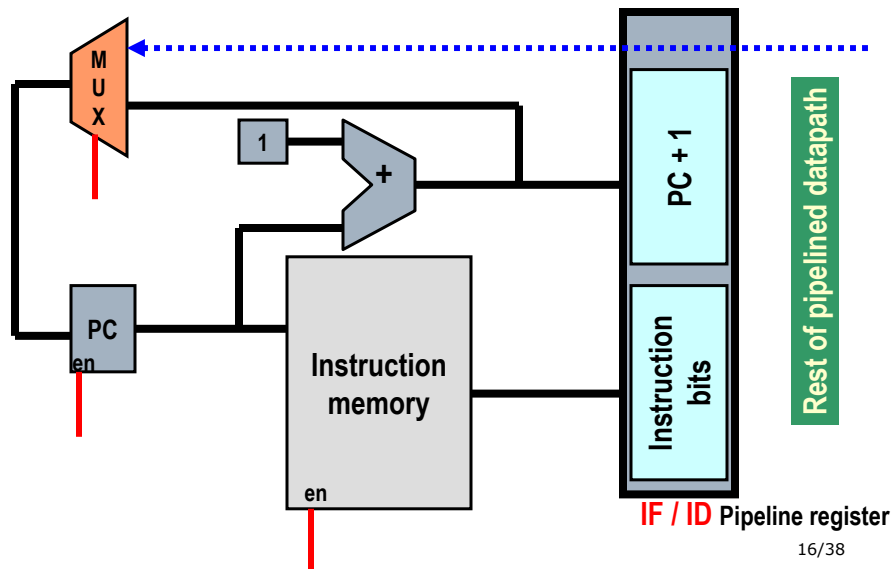


Stage 1: Fetch

- ❑ 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

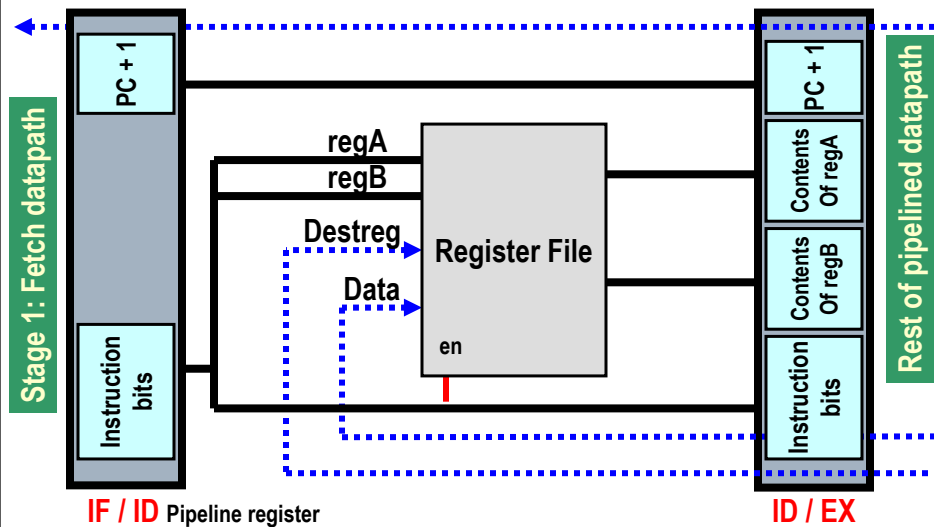
Pipeline datapath – Fetch stage



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

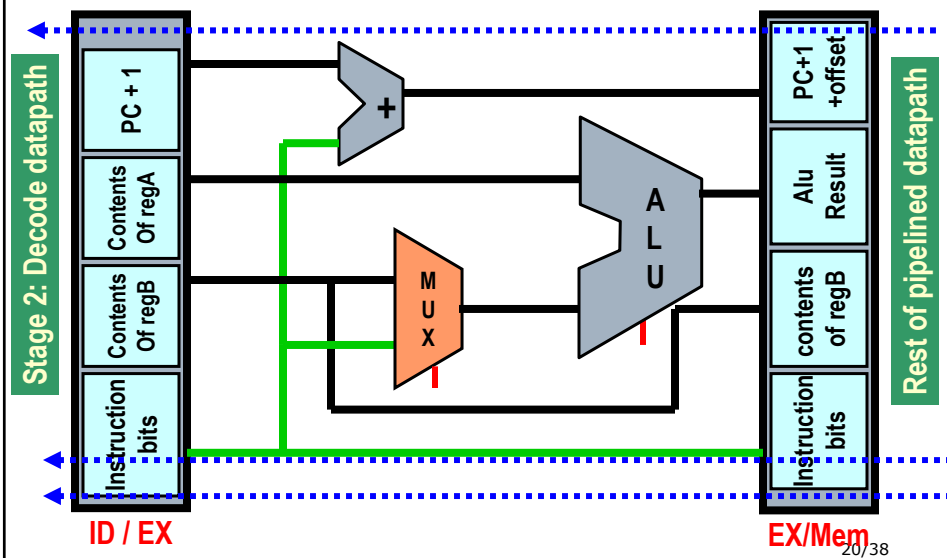


Stage 3: Execute

- ❑ 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

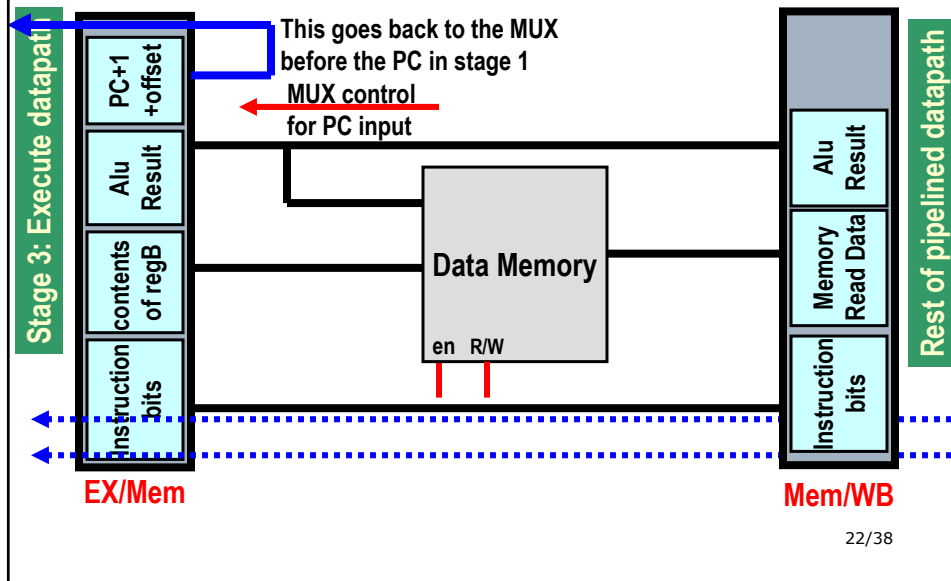
Pipeline datapath – Execute stage



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

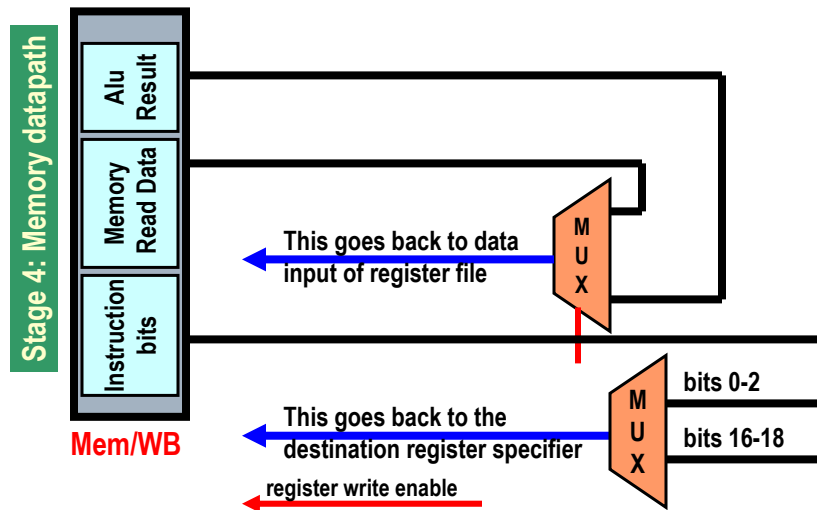
Pipeline datapath – Memory stage

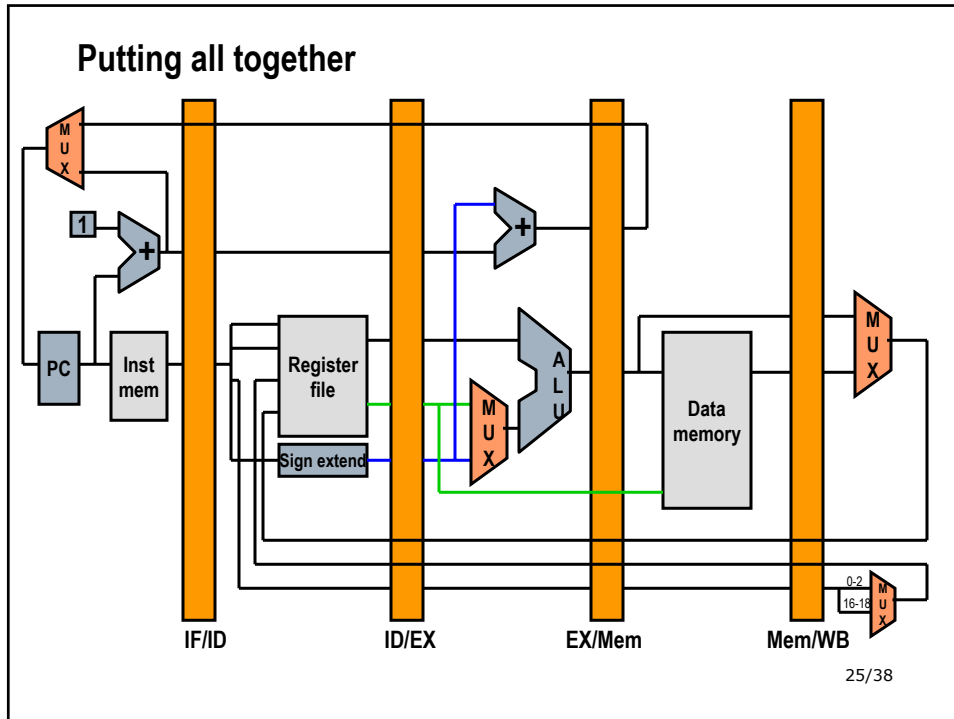


Stage 5: Write back

- 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



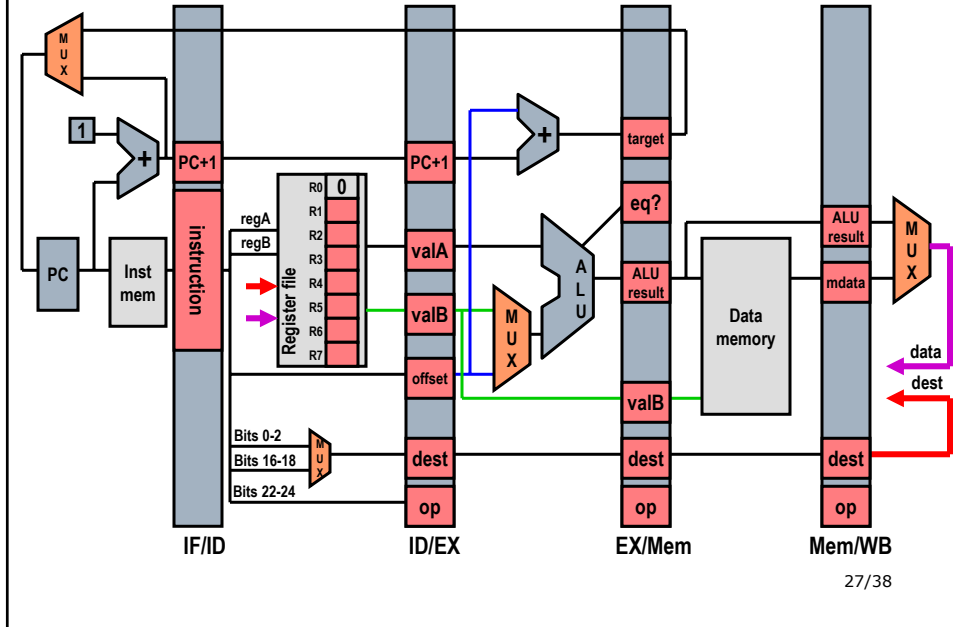


Sample Code (Simple)

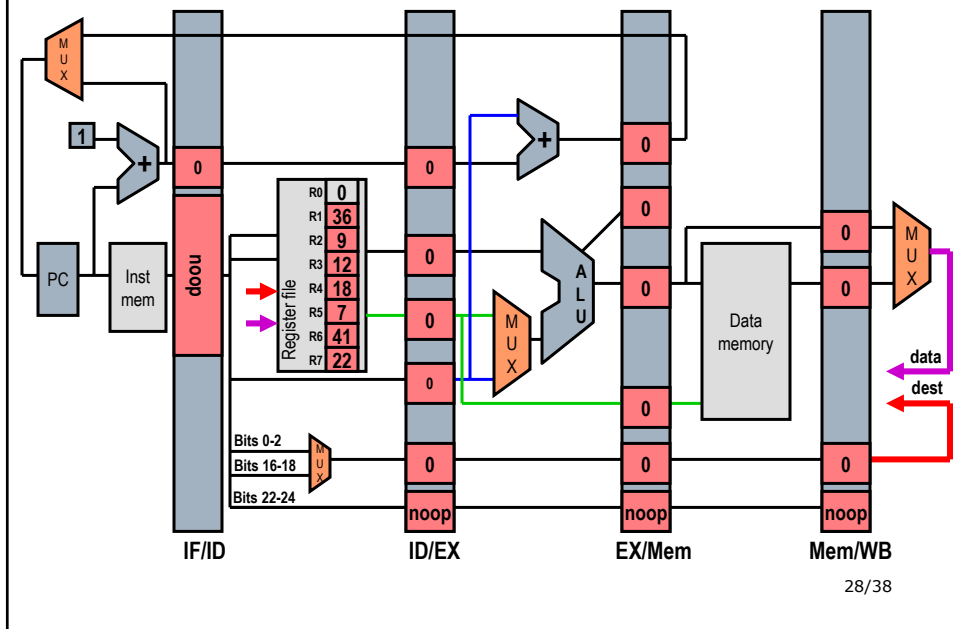
Let's run the following code on pipelined LC2K2x:

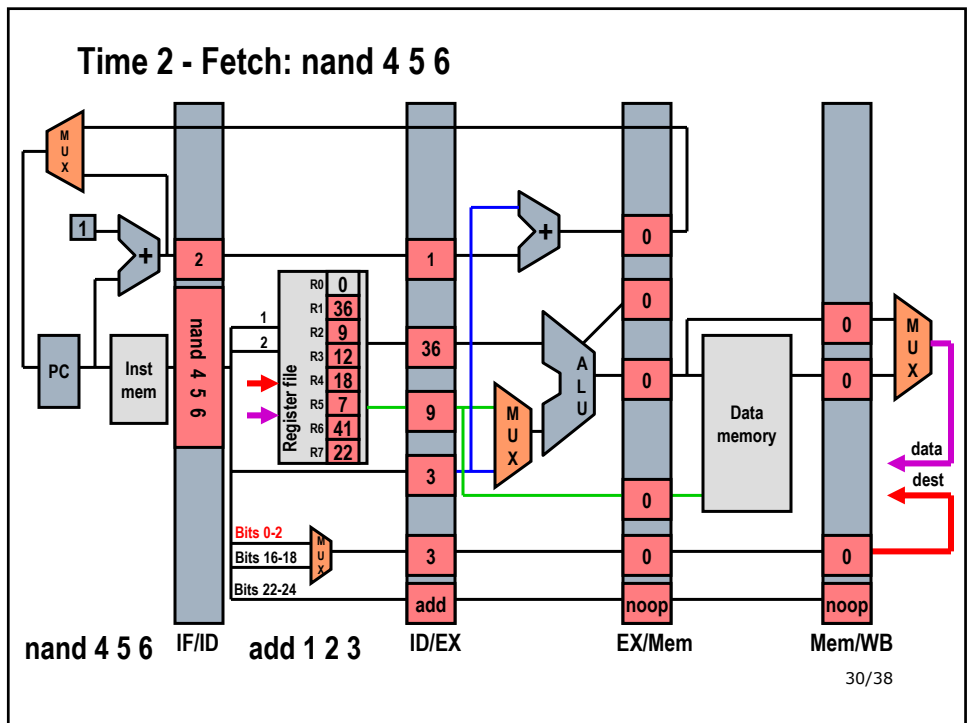
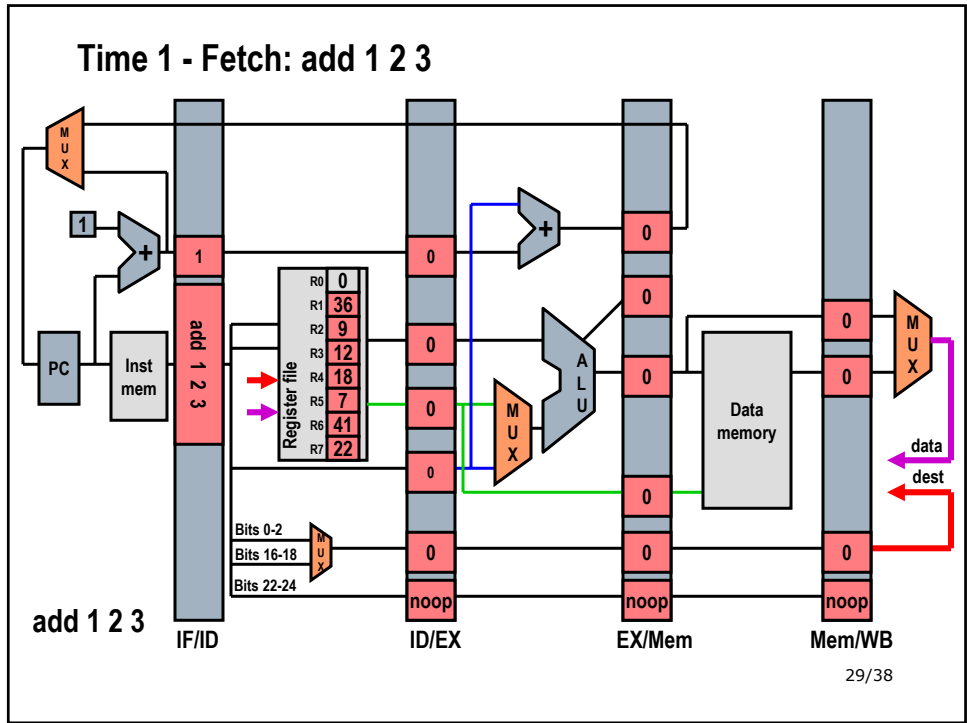
- add 1 2 3 ; reg 3 = reg 1 + reg 2
- nand 4 5 6 ; reg 6 = reg 4 & reg 5
- lw 2 4 20 ; reg 4 = Mem[reg2+20]
- add 2 5 5 ; reg 5 = reg 2 + reg 5
- sw 3 7 10 ; Mem[reg3+10]=reg 7

Pipeline datapath

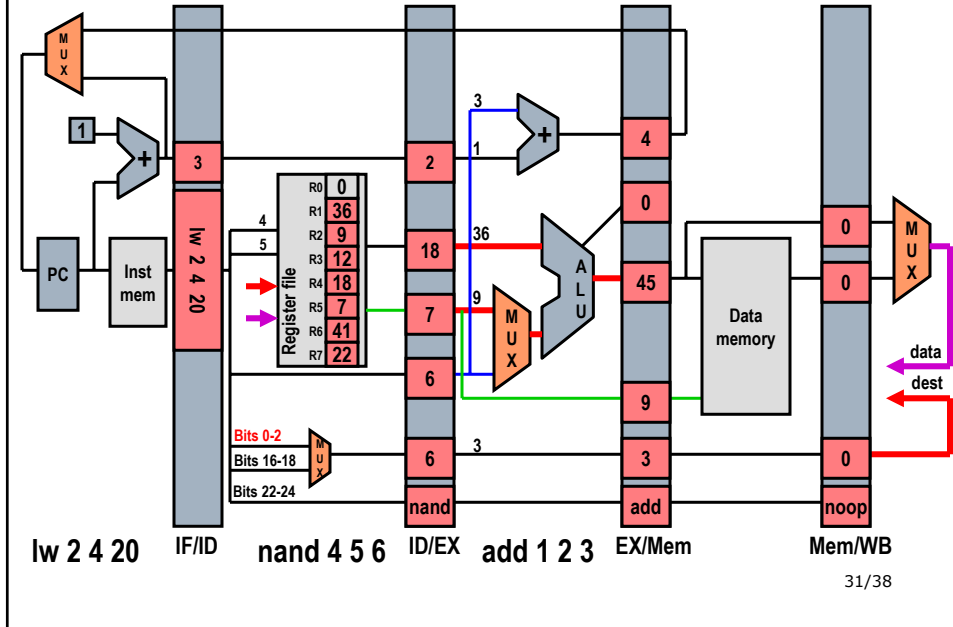


Time 0 - Initial state

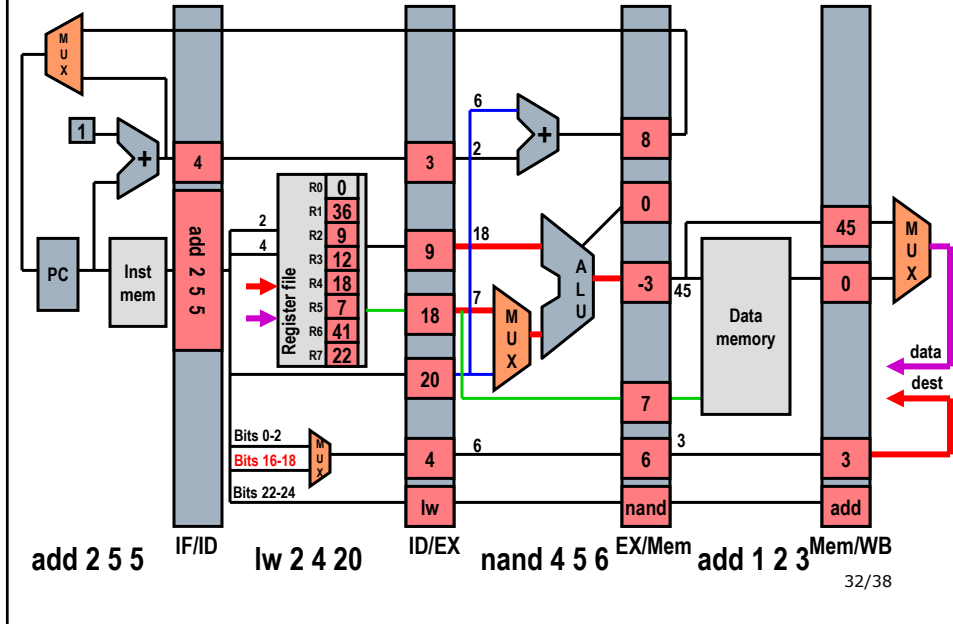




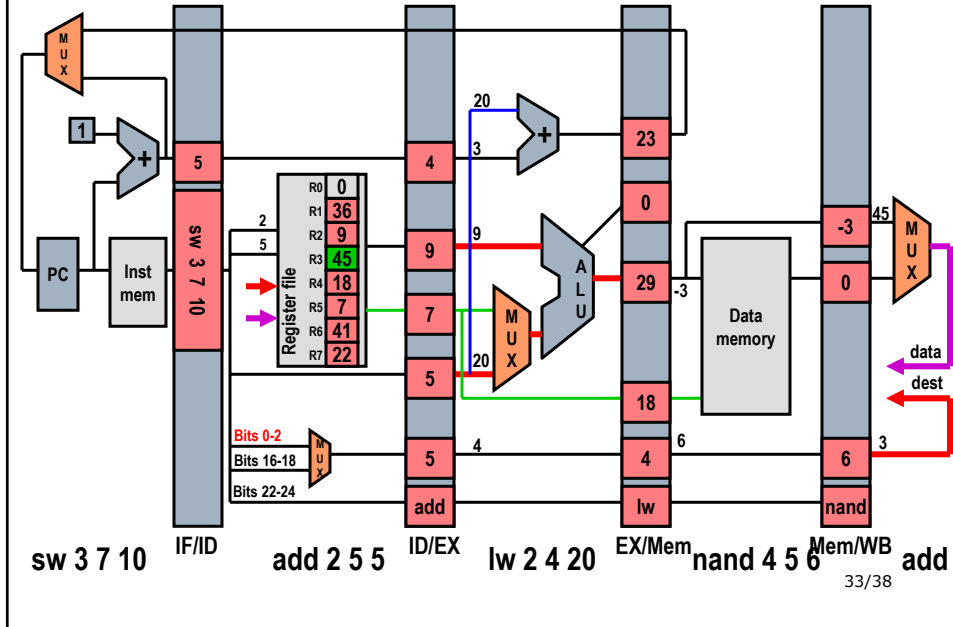
Time 3 - Fetch: lw 2 4 20



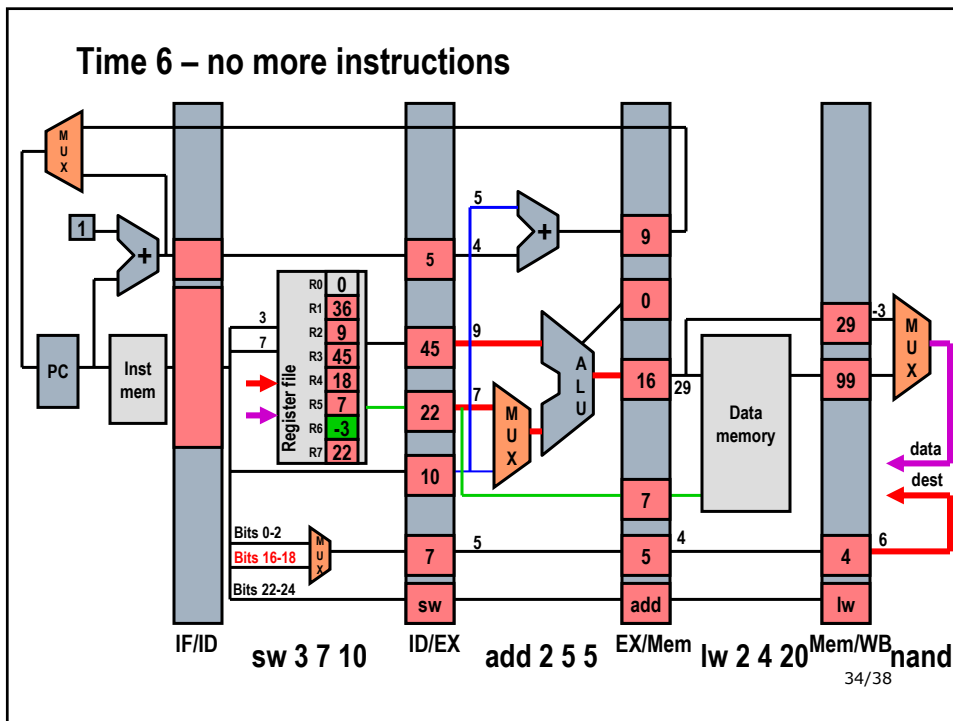
Time 4 - Fetch: add 2 5 5



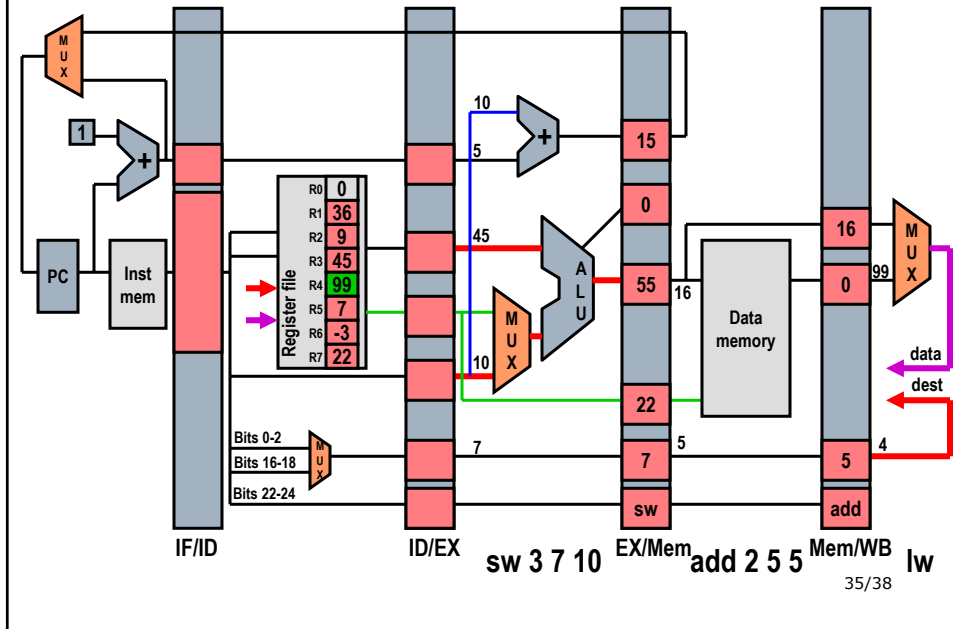
Time 5 - Fetch: sw 3 7 10



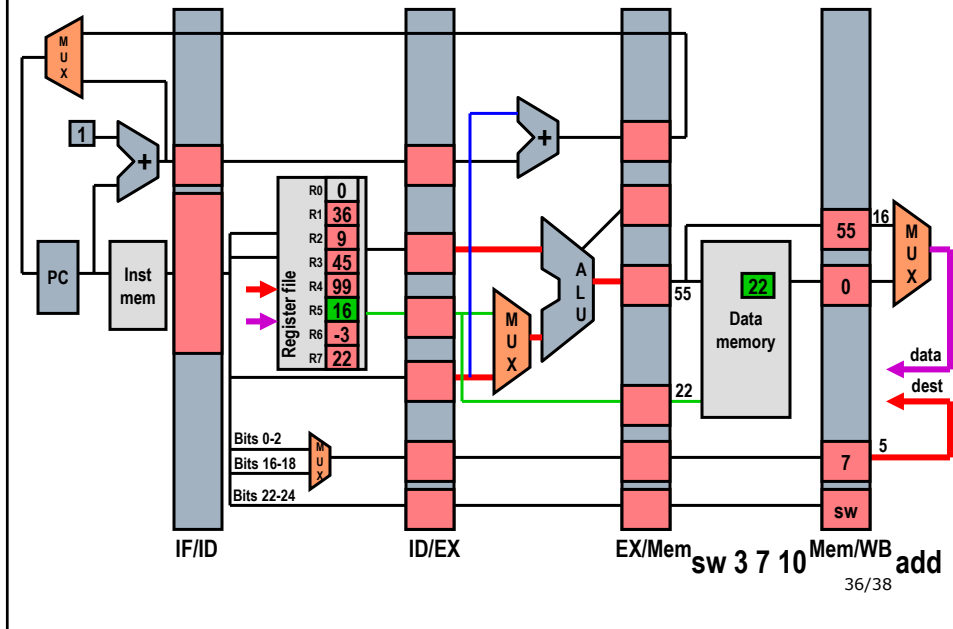
Time 6 - no more instructions

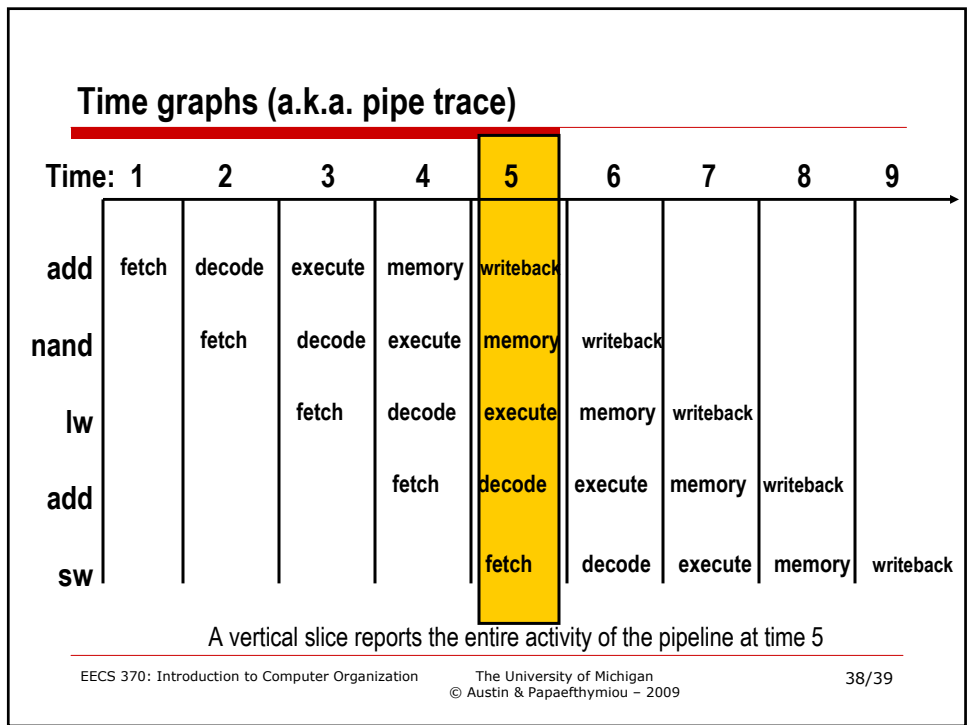
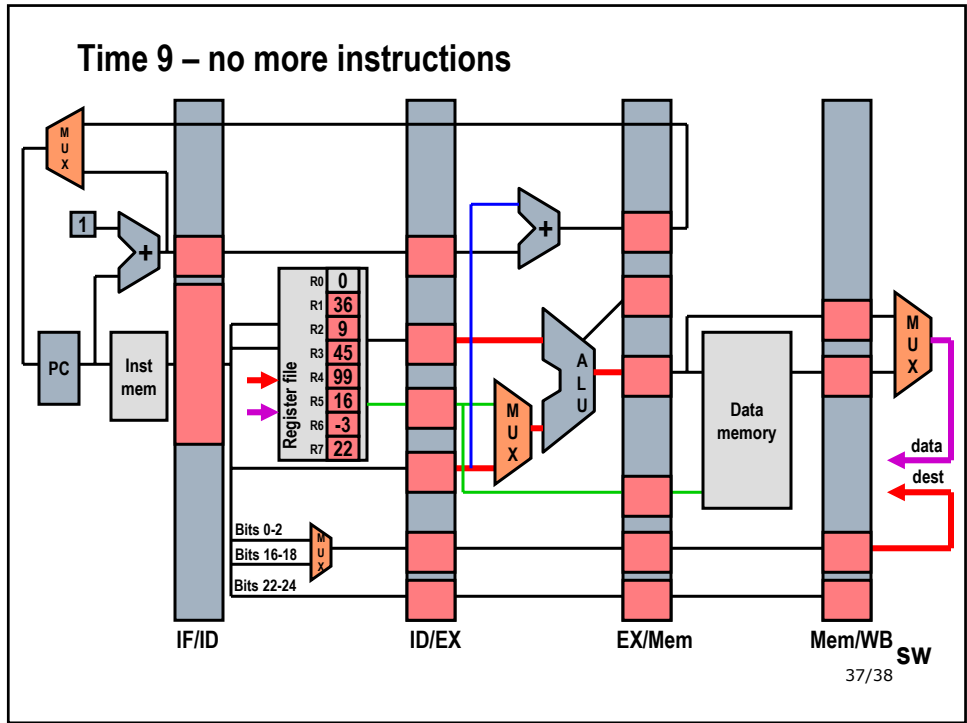


Time 7 – no more instructions



Time 8 – no more instructions





What can go wrong?

- ❑ **Data hazards:** since register reads occur in stage 2 and register writes occur in stage 5 it is possible to read the wrong value if it is about to be written.
- ❑ **Control hazards:** A branch instruction may change the PC, but not until stage 4. What do we fetch before that?
- ❑ **Exceptions:** How do you handle exceptions in a pipelined processor with 5 instructions in flight?

- ❑ Next Lecture: Data Hazards