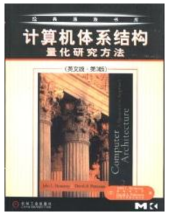
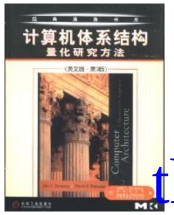


A. Pipelining: Basic Concepts

What is pipelining?

How is the pipelining Implemented?

What makes pipelining hard to implement?



Definition

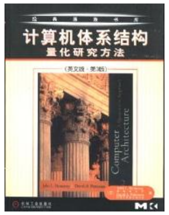
the goal

What's it?

Pipelining is an implementation technique whereby multiple instructions are overlapped in execution; it takes advantage of parallelism that exists among the actions needed to execute an instruction.

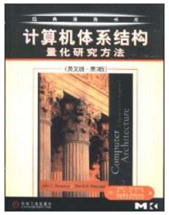
Today, pipelining is the key implementation technique used to make fast CPUs.

Not only for CPU: a 12 level pipeline for geometry transformation in GPU(Clark 1982)

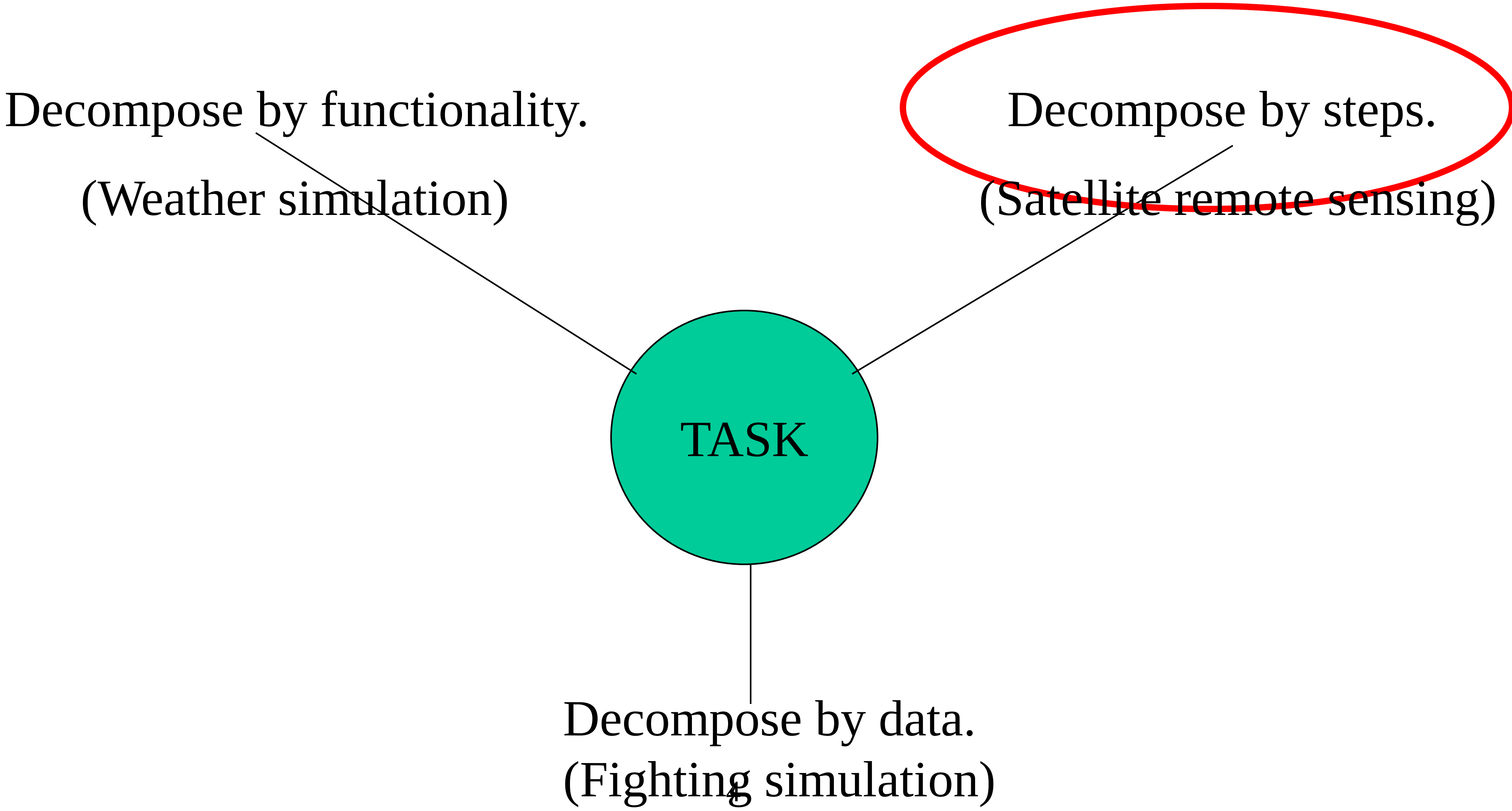


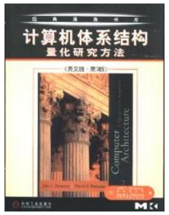
Parallelism

- From dictionary: the quality or condition of being parallel; a parallel relationship.
- The Nature of a TASK. Goal in parallel computing and Arch. design : seek parallelism and try using it to improve performance.



Patterns of parallel task handling

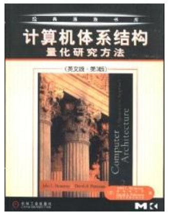




What is Pipelining ?

- Pipelining:
  - "*A technique designed into some computers to increase speed by starting the execution of one instruction before completing the previous one.*"

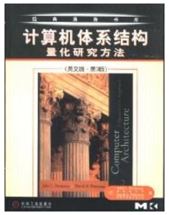
    ----Modern English-Chinese Dictionary
  - implementation technique whereby different instructions are overlapped in execution at the same time.
  - implementation technique to make fast CPUs



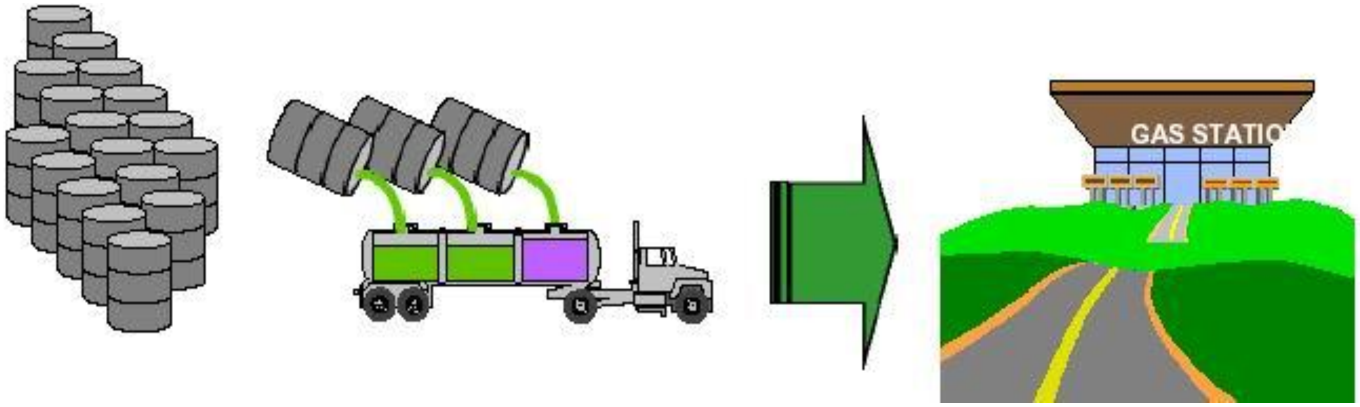
It likes Auto Assembly line

- An arrangement of workers, machines, and equipment in which the **product being assembled passes consecutively from operation to operation until completed.**
- **Ford installs first moving assembly line in 1913.** The right picture shows the moving assembly line at Ford Motor Company's michigan plant.

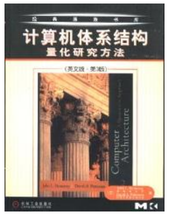
(84 distinct steps)



Trucking gas from depot to gas station



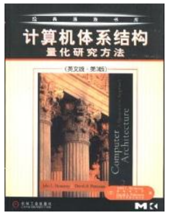
- The steps:
  - Get the barrels
  - Load them into the truck
  - Drive to the gas station
  - Unload the gas
  - Return for more oil
- Let's do the math
  - Each truck can carry 5 barrels
  - Can load a truck with 5 barrels in 1 hour
  - It takes each truck 1 day to drive to and from gas station
  - How many barrels per week are delivered?



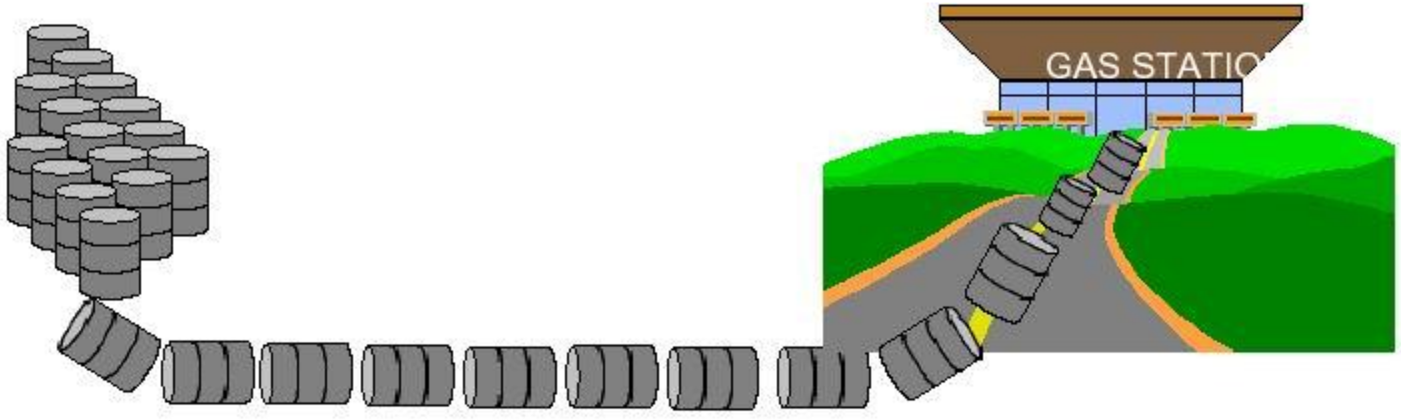
Looks a Lot Like a Multi-cycle Processor

- **What are the steps ?**

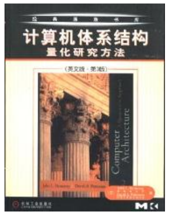
– Fetch an instruction	(Get the barrels)
– Decode the instruction	(Load them into the truck)
– ALU OP	(Drive to the gas station)
– Memory Access	(Unload the gas)
– Write-back	(Return for more oil)



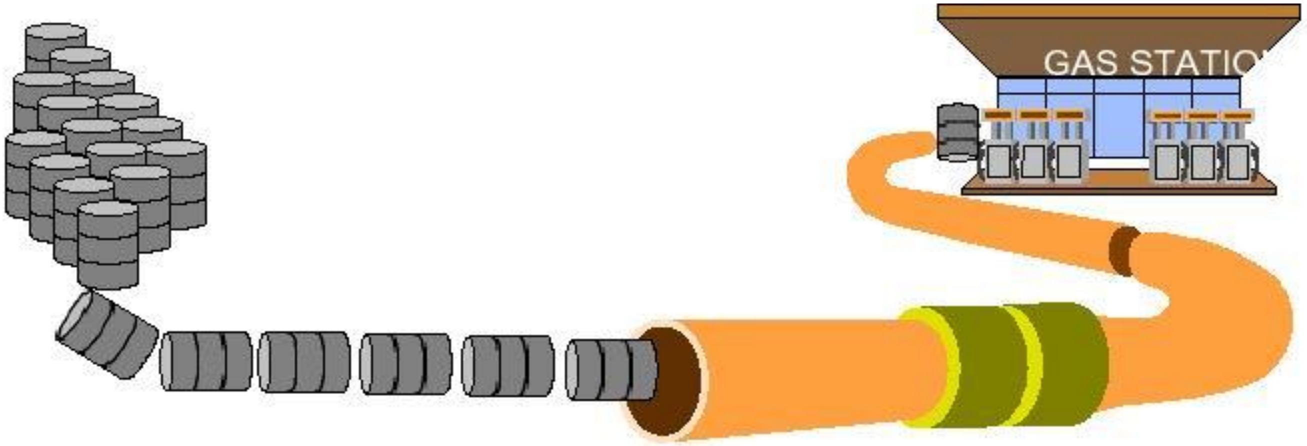
A better way, but dangerous



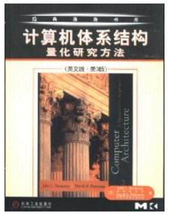
- Roll the barrels down the road
  - Big fire hazard



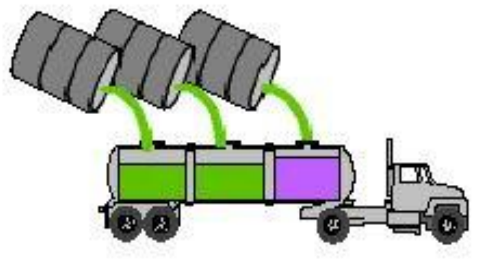
Big idea: Build a pipeline



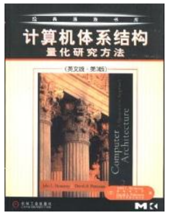
- Now let's do the math
  - Pipeline can accept 1 barrel every hour
  - How many barrels get delivered to the gas station per day?



Trucking vs. Pipelines

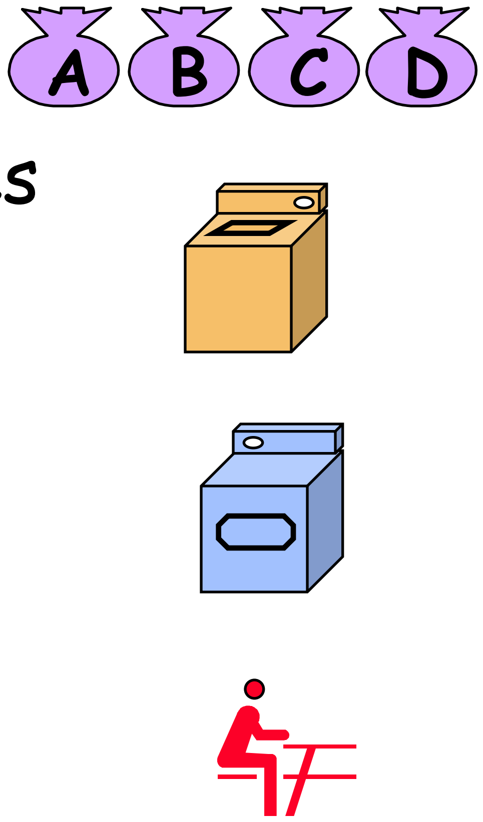


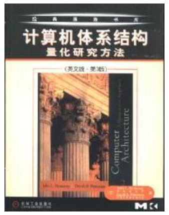
- Trucks
  - Truck with 5 barrels takes 1 day to drive to and from gas station, while need 2 hours for loading and unloading
  - LOTS of TIME when loading area,gas station, and pieces of the road are unused
- Pipelines
  - Pipeline can accept 1 barrel every hour
  - Resources (loading area, gas station,pipelines) are always in use



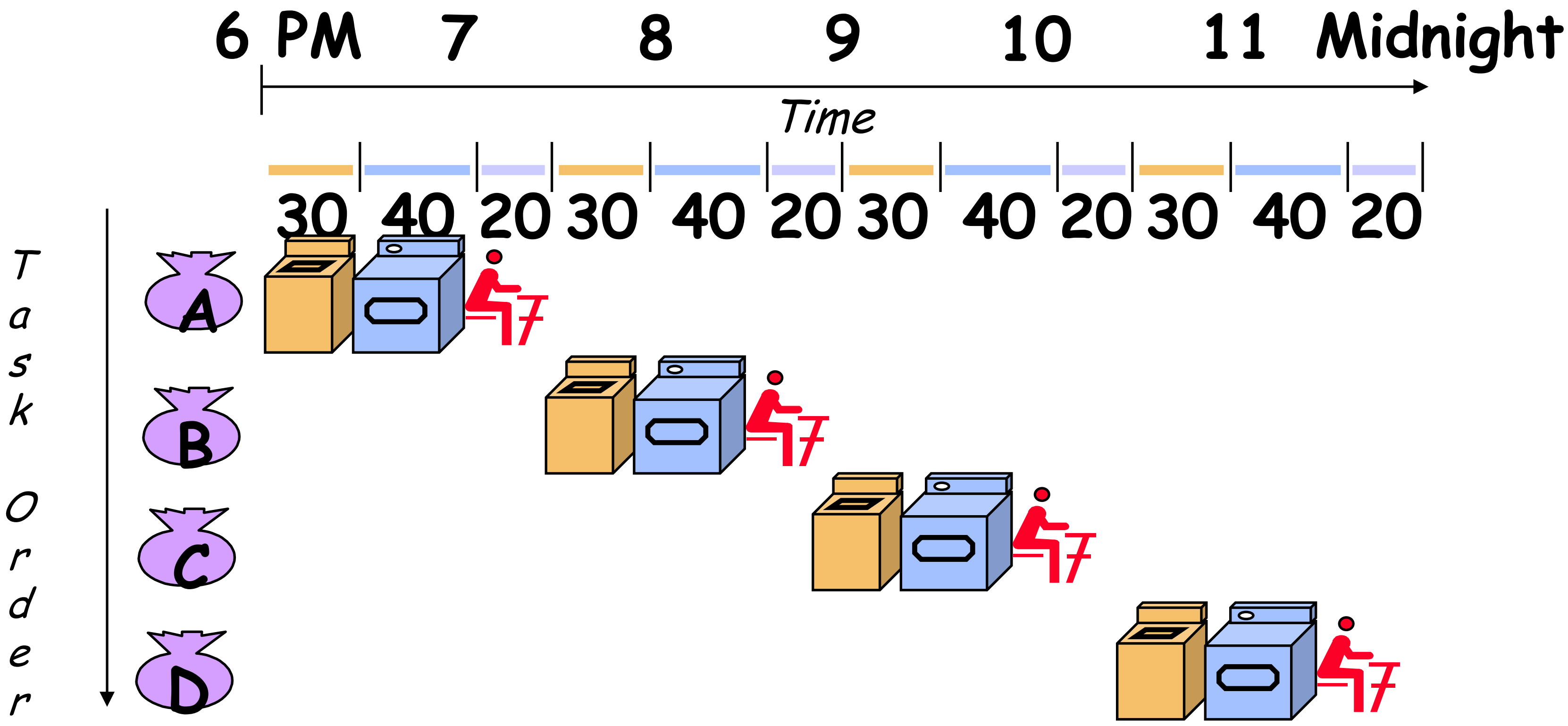
Why Pipelining: Its Natural

- Laundry
  - Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
  - Washer takes 30 minutes
  - Dryer takes 40 minutes
  - “Folder” takes 20 minutes

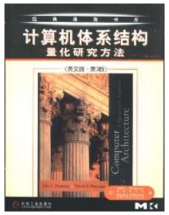




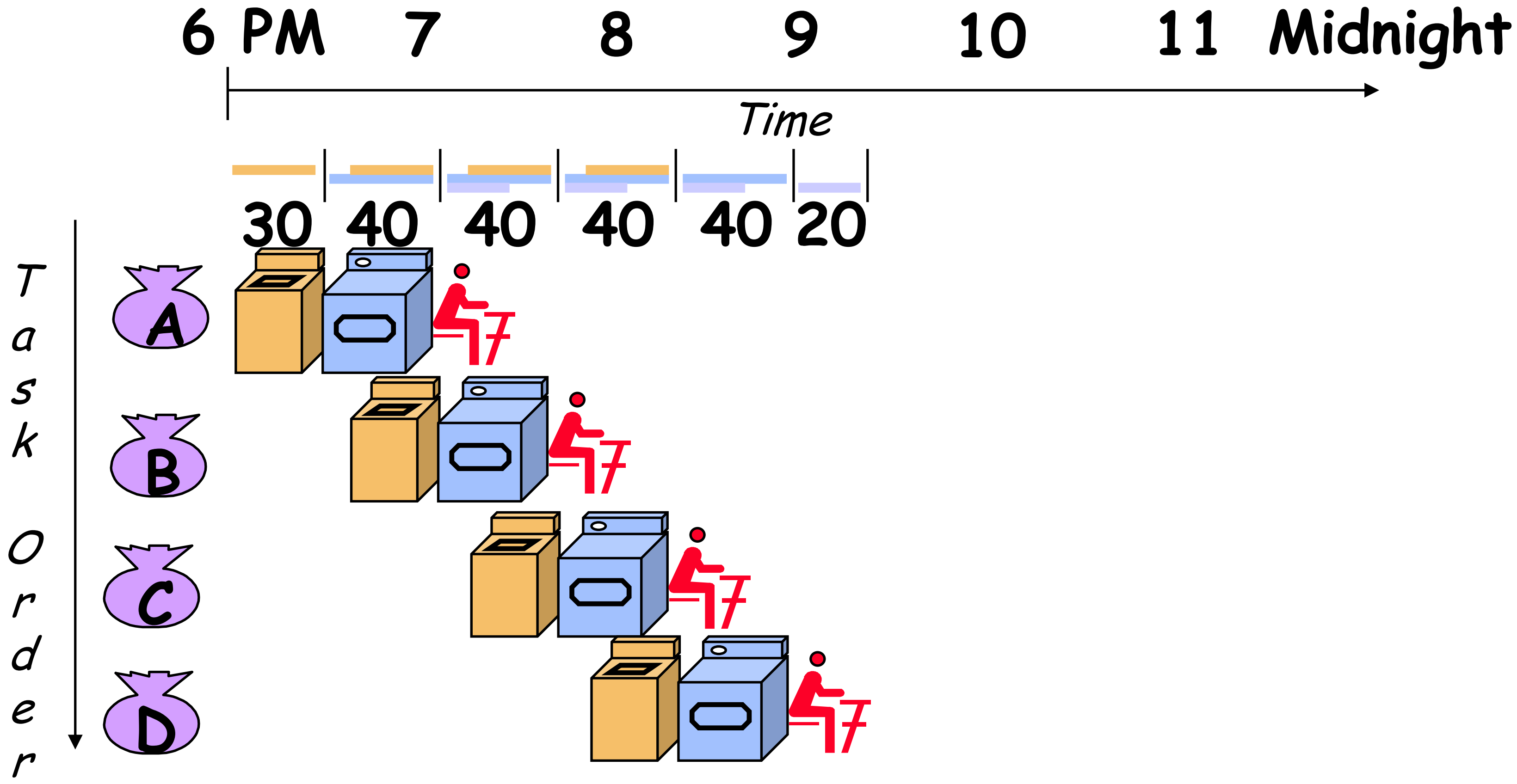
Sequential Laundry



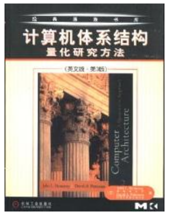
- Sequential laundry takes 6 hours for 4 loads
- **If they learned pipelining, how long would laundry take?**



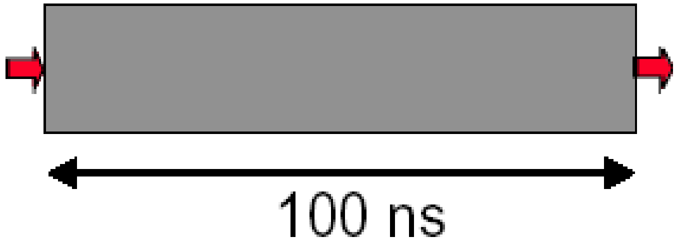
Pipelined Laundry ----Start work ASAP

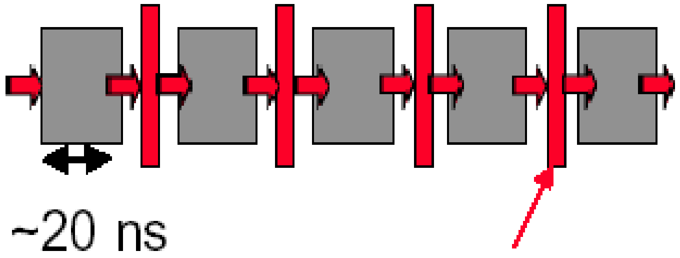


- Pipelined laundry takes 3.5 hours for 4 loads



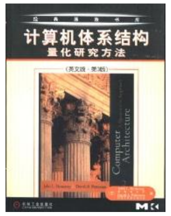
Why pipelining ： overlapped



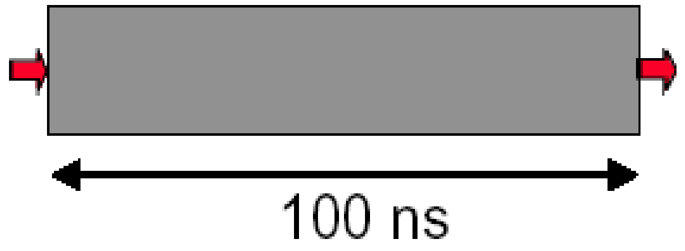


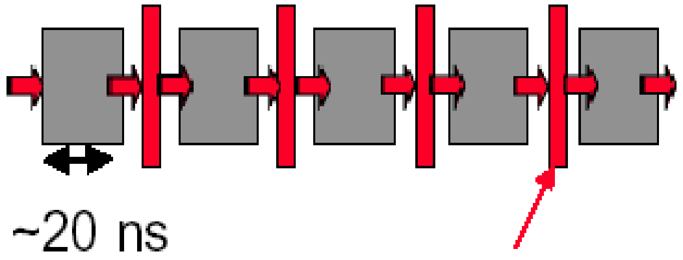
- Only deal one task each time.
- This task takes
  " such a long time"

- Latches, called pipeline registers' break up computation into 5 stages
- Deal 5 tasks at the same time.



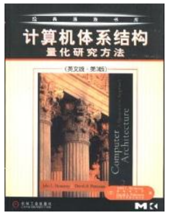
Why pipelining: more faster





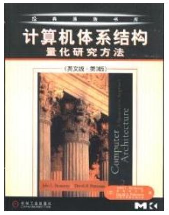
- Can "launch" a new computation every 100ns in this structure
- Can finish 10^7 computations per second

- Can launch a new computation every 20ns in pipelined structure
- Can finish 5×10^7 computations per second



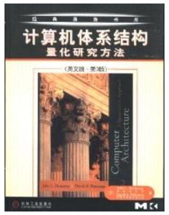
Why pipelining : conclusion

- The key implementation technique used to Make fast CPU: decrease CPUtime.
- Improving of Throughput (rather than individual execution time)
- Improving of efficiency for resources (functional unit)



What is a pipeline ?

- A pipeline is like an auto assemble line
- A pipeline has many stages
- Each stage carries out a different part of instruction or operation
- The stages, which cooperates at a synchronized clock, are connected to form a pipe
- An instruction or operation enters through one end and progresses through the stages and exit through the other end
- Pipelining is an implementation technique that exploits parallelism among the instructions in a sequential instruction stream



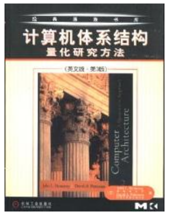
Ideal Performance for Pipelining

- If the stages are perfectly balanced, The time per instruction on the pipelined processor equal to:

$$\frac{\textit{Time per instruction on unpipelined machine}}{\textit{Number of pipe stages}}$$

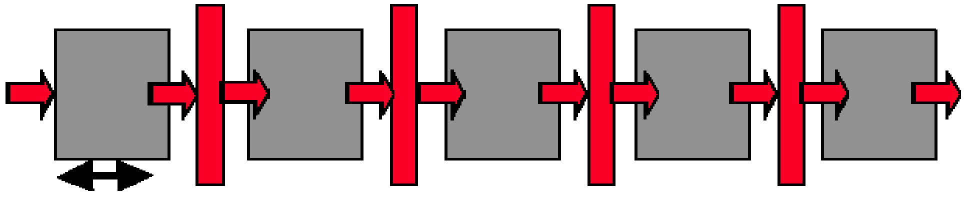
- So, **Ideal speedup equal to**

  Number of pipe stages.



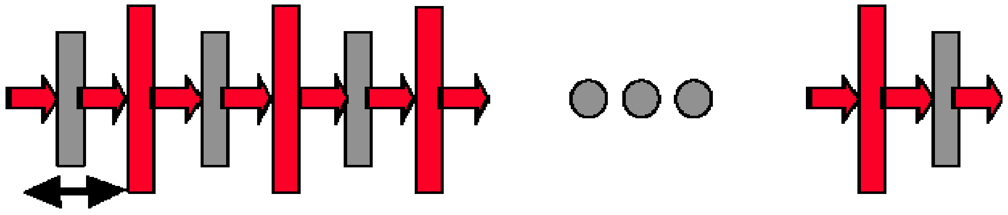
Why not just make a 50-stage pipeline ?

- Some computations just won't divide into any finer (shorter in time) logical implementation.



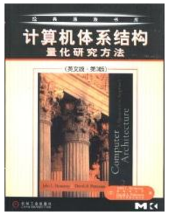
~20 ns

5 stages **OK**



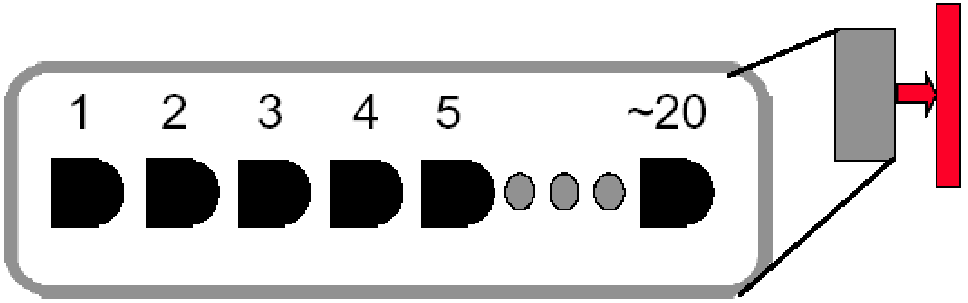
~2 ns

50 stages NO. Sorry!

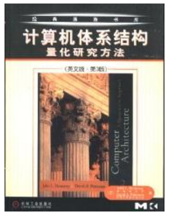


Why not just make a 50-stage pipeline ?

- Those latches are NOT free, they take up area, and there is a real delay to go THRU the latch itself.
  - Machine cycle > latch latency + clock skew
- In modern, deep pipeline (10-20 stages), this is a real effect
- Typically see logic "depths" in one pipe stage of 10-20 "gates".

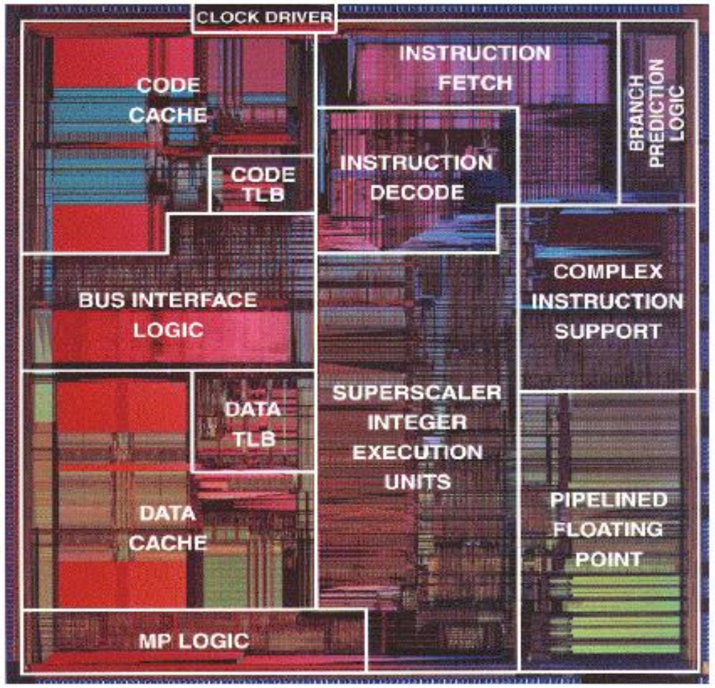


At these speeds, and with this few levels of logic, latch delay is important

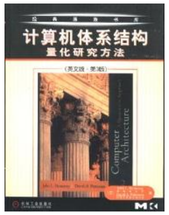


How Many Pipeline Stages?

- E.g., Intel
  - Pentium III, Pentium 4: 20+ stages
  - More than 20 instructions in flight
  - High clock frequency (>1GHz)
  - High IPC

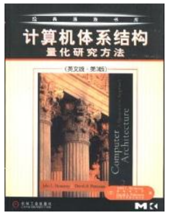


- Too many stages:
  - Lots of complications
  - Should take care of possible dependencies among in-flight instructions
  - Control logic is huge



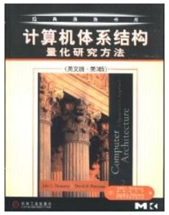
Simple implementation of a RISC Instruction Set (MIPS)

- Start with Implementation without pipelining
  - single-cycle implementation
  - multi-cycle implementation
- Pipelining the RISC Instruction Set
- Pipelining performance issues
- How can we do it efficiently ?
- Examples



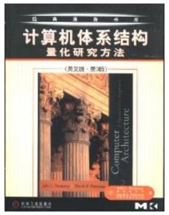
How MIPS instruction set is implemented without pipelining ?

- Five phases
  - IF: Instruction fetch cycle
    - √ Send PC to memory and fetch the Instruction
    - √ Update the PC to NPC by adding 4
  - ID: Instruction decode/ register fetch cycle
    - √ Decode the instruction
    - √ read the registers
    - √ If needed, sign-extend the offset field of the instruction.

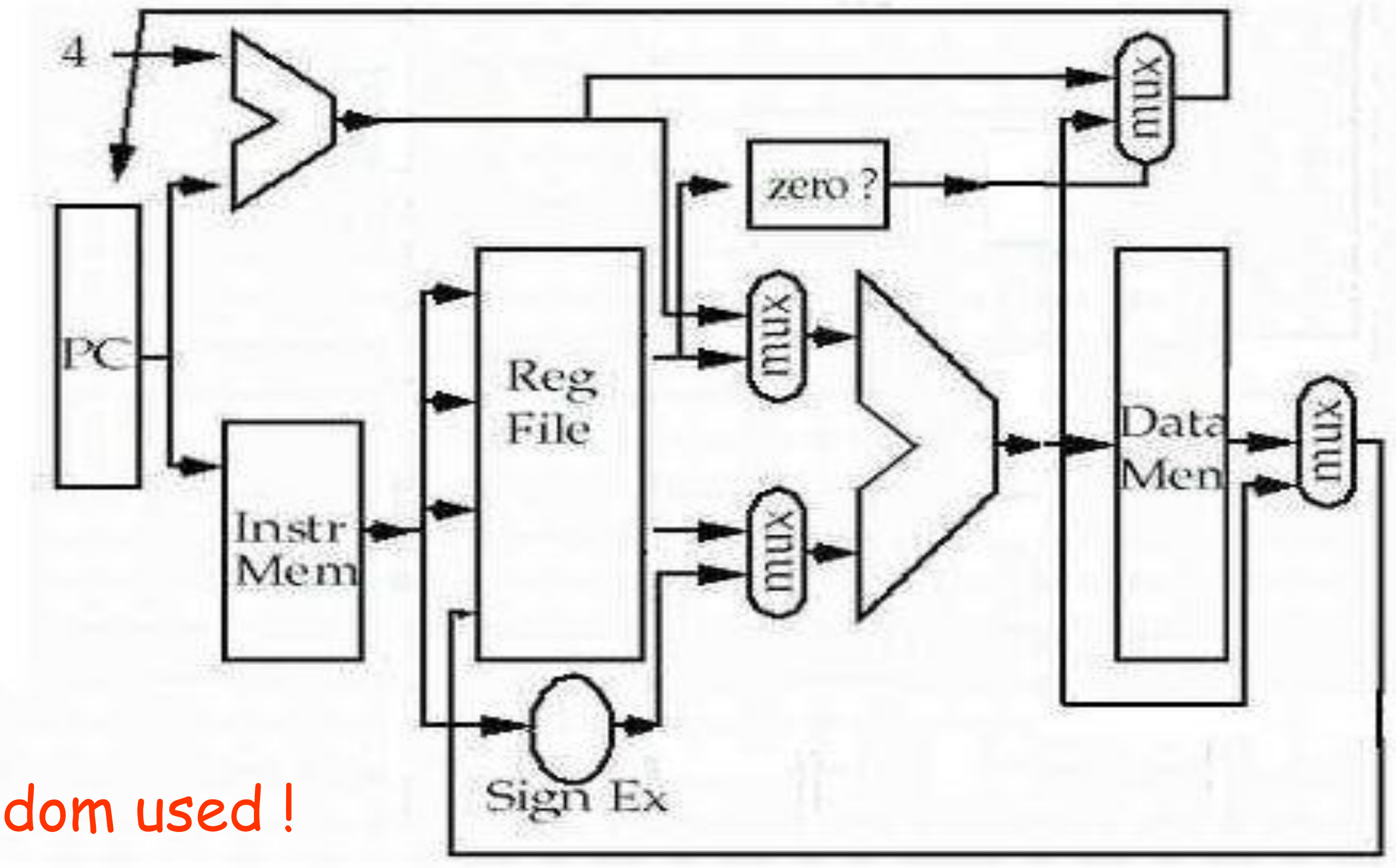


The other three phases

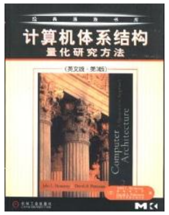
- EX: Execution/ effective address cycle
  - √ **Memory reference:** calculate the address
  - √ **R-R/ R-I ALU:** ALU operation on R-R or R-I
  - √ **Branch:** Do the equality test, compute the possible target address and send it to NPC if the equality test is true.
- MEM: Memory access
  - √ **Load:** send the effective address to the data memory and fetch the data
  - √ **Store:** write the data from the ID phases using the effective address.
- WB: Write-back cycle
  - √ **Load or ALU:** write the result into the register file.



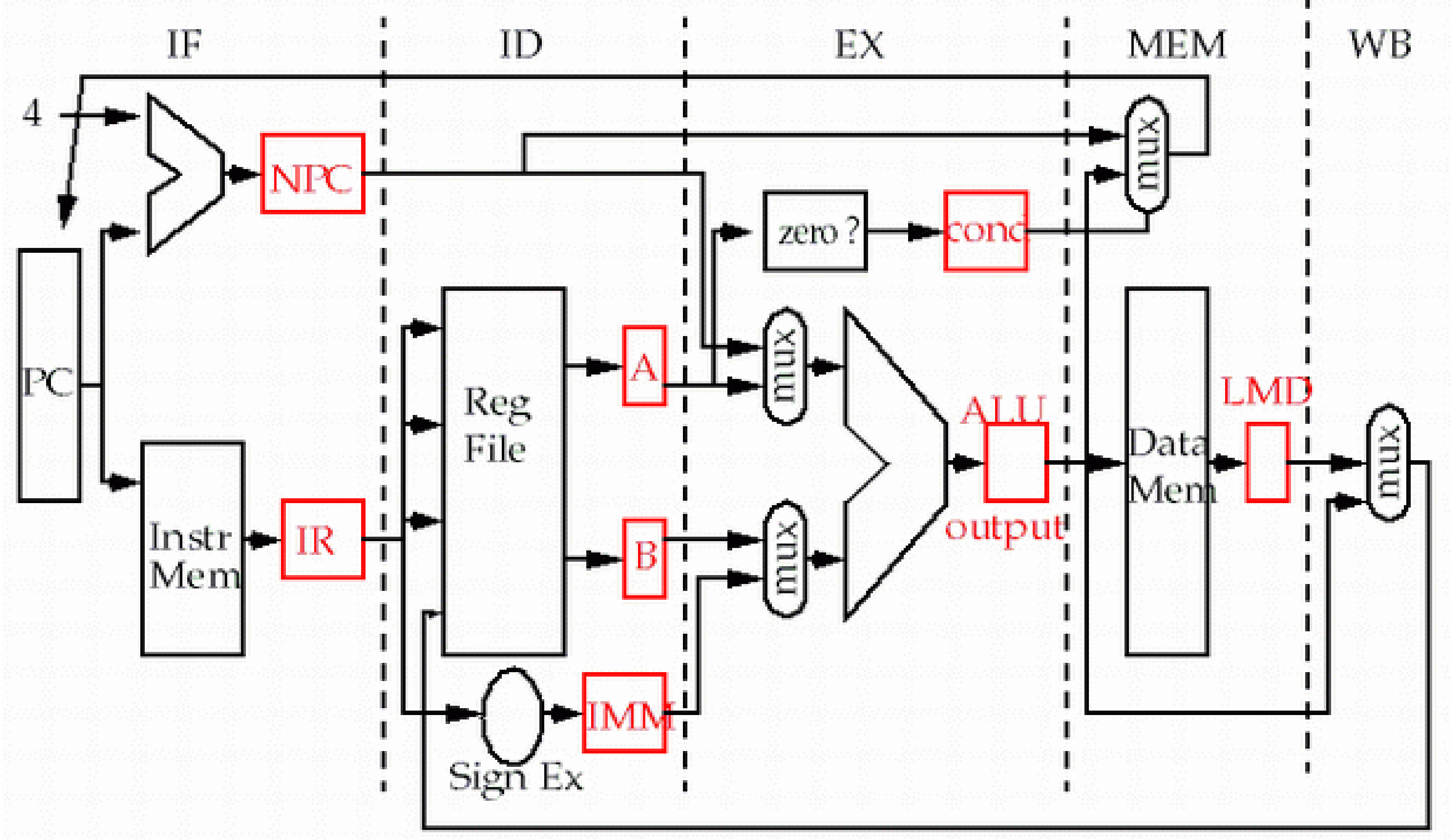
Single-cycle implementation

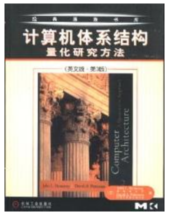


seldom used !



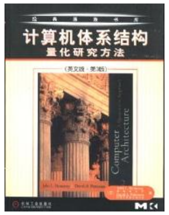
Multi-cycle implementation





About Multi-cycle implementation

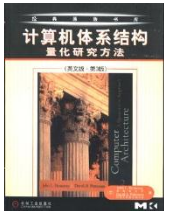
- The temporary storage locations were added to the datapath of the unpipelined machine to make it easy to pipeline.
- Note that branch and store instructions take 4 clock cycles.
  - Assuming branch frequency of 12% and a store frequency of 10%, CPI is 4.78.
- This implementation is not optimal.



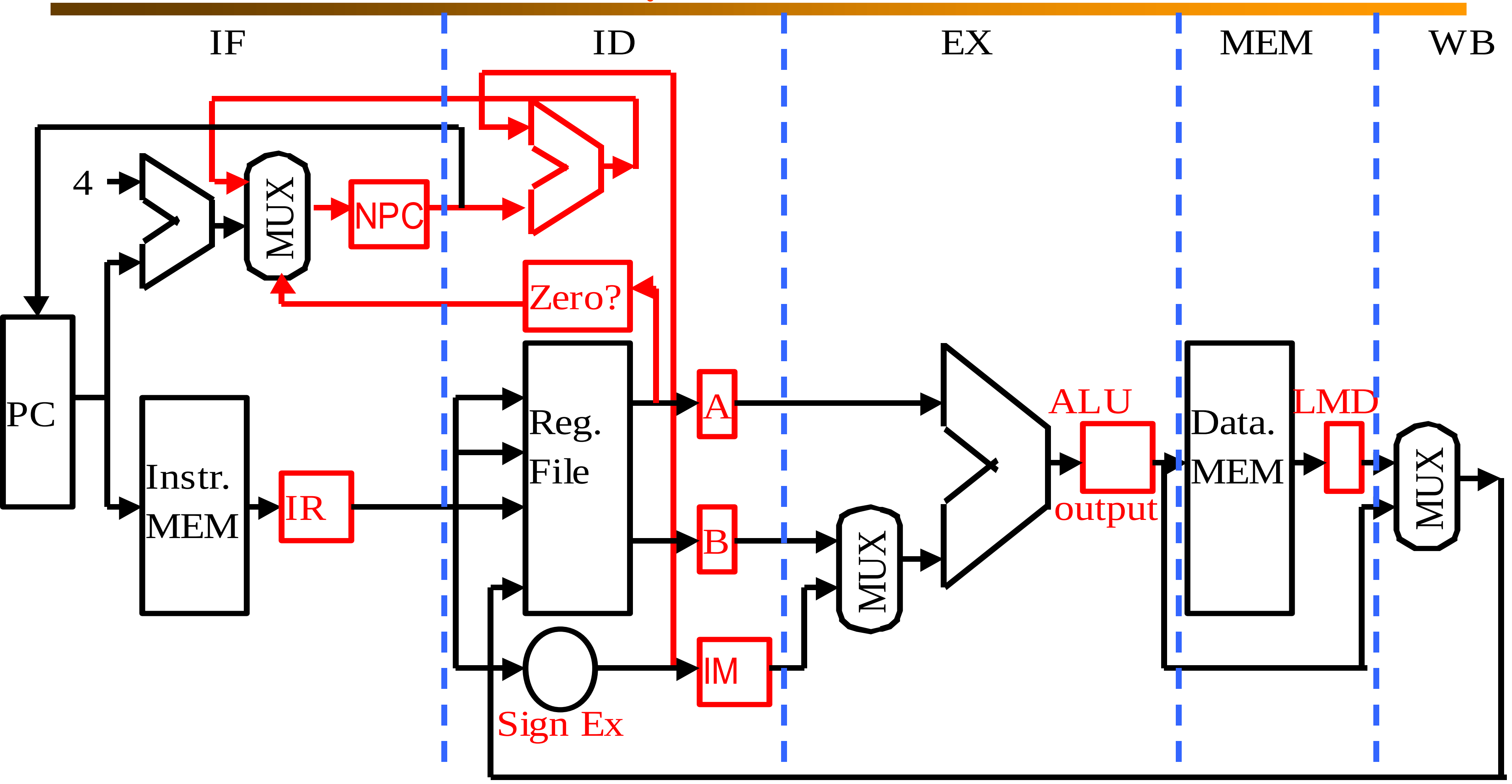
How to improve the performance ?

- For a possible branch, do the equality test and compute the possible branch target by adding the sign-extended offset to the incremented PC earlier in ID.
- Completing ALU instructions during the MEM cycle
- So, branch instructions take only 2 cycles, store and ALU instructions take 4 cycles, and load instruction takes the longest time 5 cycles.
- CPI drops to 4.07 assuming 47% ALU operation frequency.

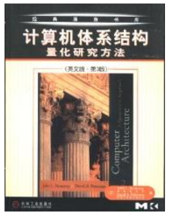
$$2\times12\% + 4\times(10\% + 47\%) + 31\%\times5 = 4.07$$



Optimized Multi-cycle implementation

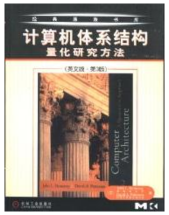


🟥 Temporary storage locations



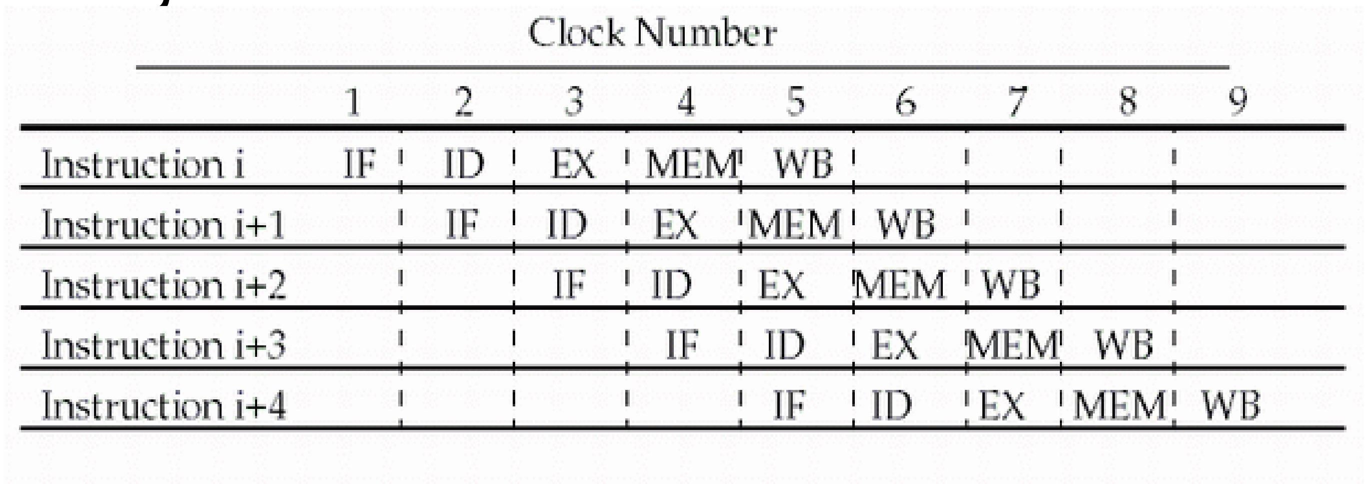
Improvement on hardware redundancy

- ALU can be shared.
- Data and instruction memory can be combined since access occurs on different clock cycles.



Pipelining MIPS instruction set

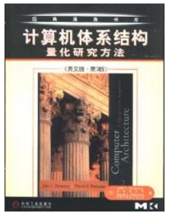
- Since there are five separate stages, we can have a pipeline in which one instruction is in each stage.
- CPI is decreased to 1, since one instruction will be issued (or finished) each cycle.
- During any cycle, one instruction is present in each stage.



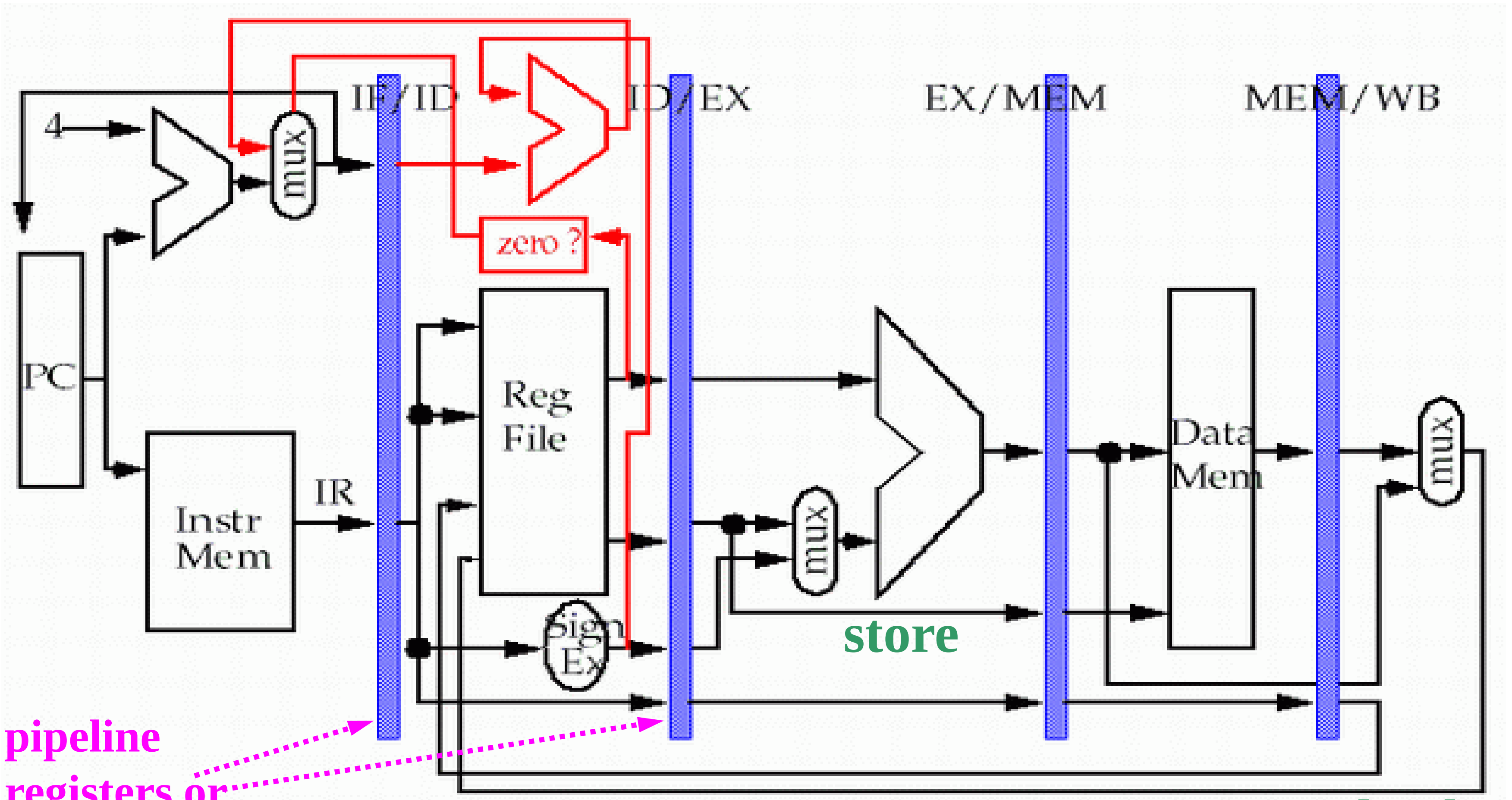
Clock Number

	1	2	3	4	5	6	7	8	9
Instruction i	IF	ID	EX	MEM	WB				
Instruction i+1		IF	ID	EX	MEM	WB			
Instruction i+2			IF	ID	EX	MEM	WB		
Instruction i+3				IF	ID	EX	MEM	WB	
Instruction i+4					IF	ID	EX	MEM	WB

- Ideally, performance is increased five fold !

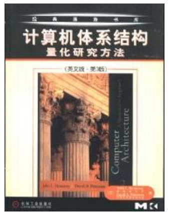


5-stage Version of MIPS Datapath



pipeline
registers or
latches

load

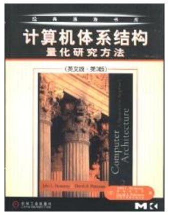


How pipelining decrease the execution time ?

If your starting point is

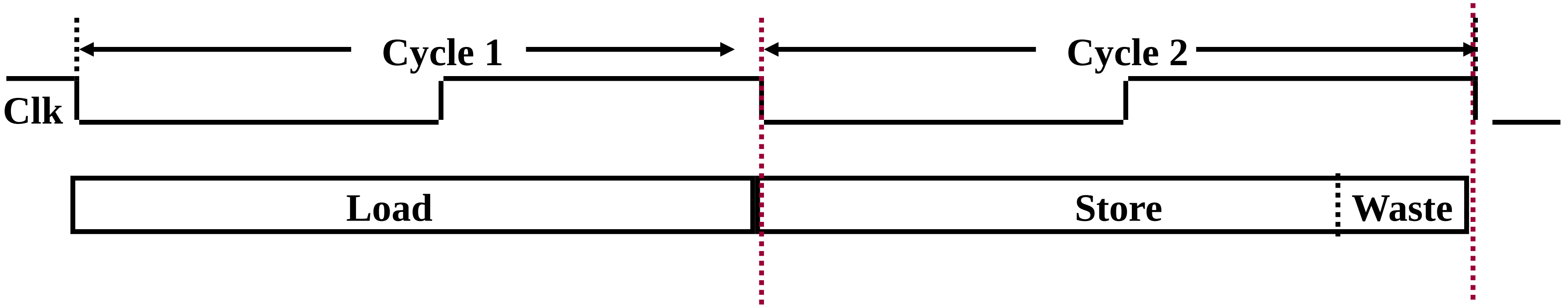
- a single clock cycle per instruction machine then
  - pipelining decreases cycle time.

- a multiple clock cycle per instruction machine then
  - pipelining decreases CPI.

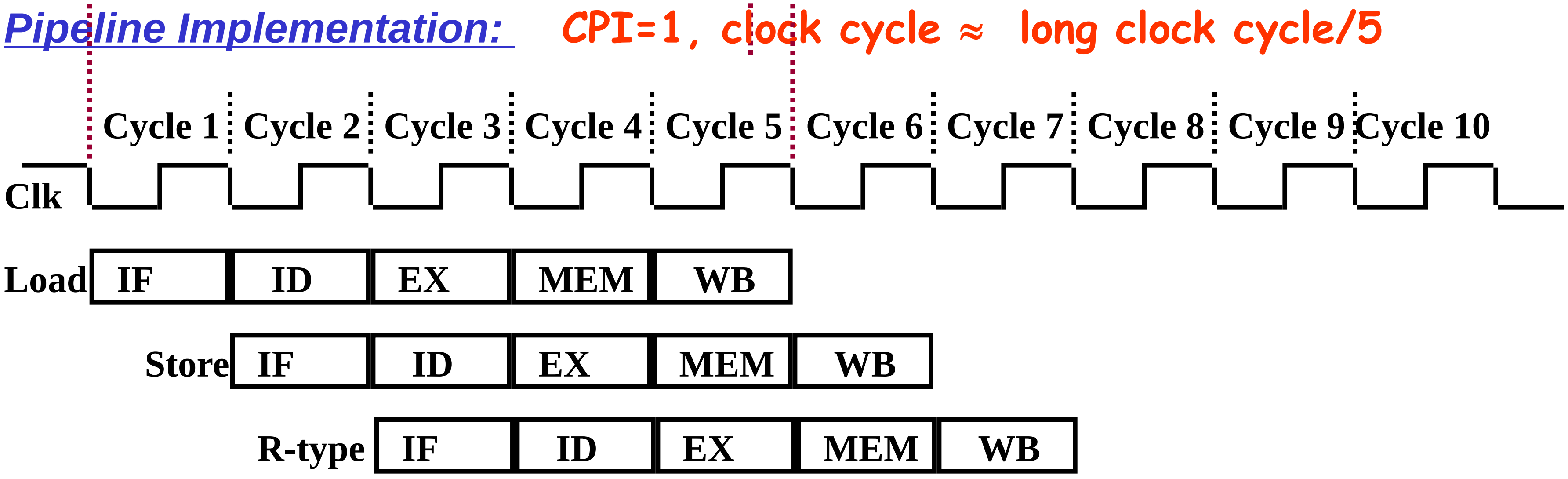


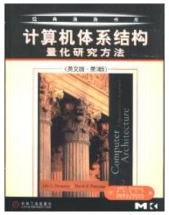
Single-cycle implementation vs. pipelining

Single Cycle Implementation: CPI=1, long clock cycle



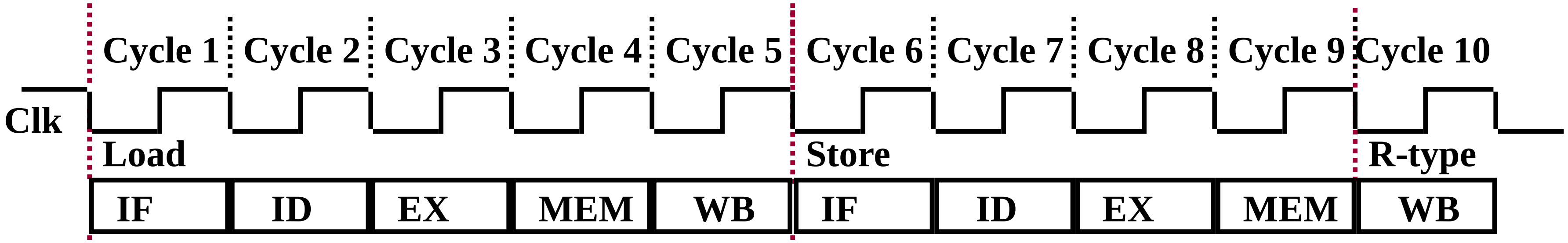
Pipeline Implementation: CPI=1, clock cycle ≈ long clock cycle/5



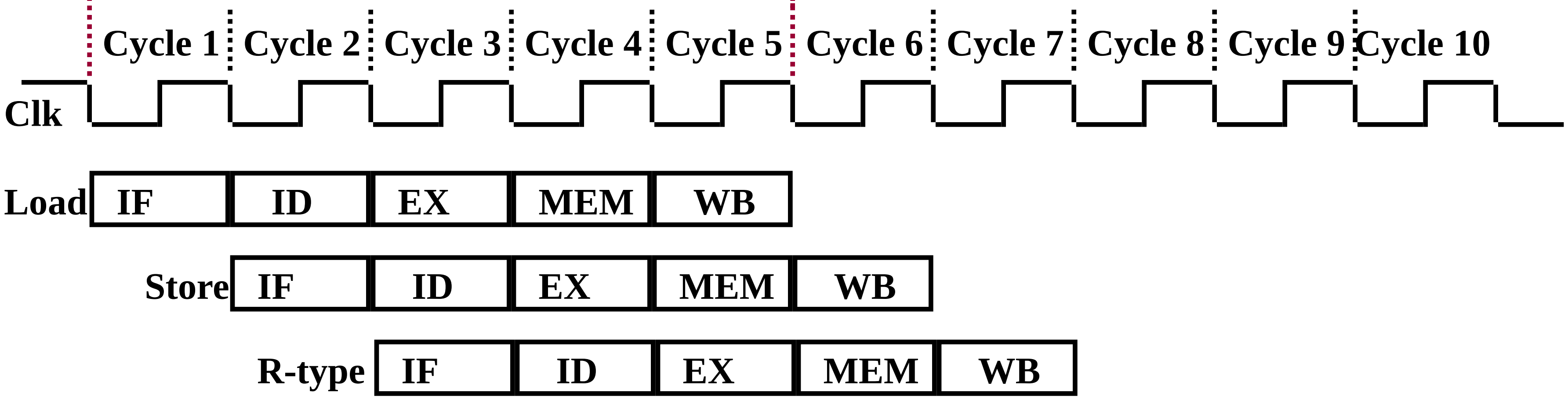


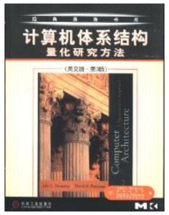
Multi-cycle implementation vs. pipelining

Multip-Cycle Implementation: CPI=5,

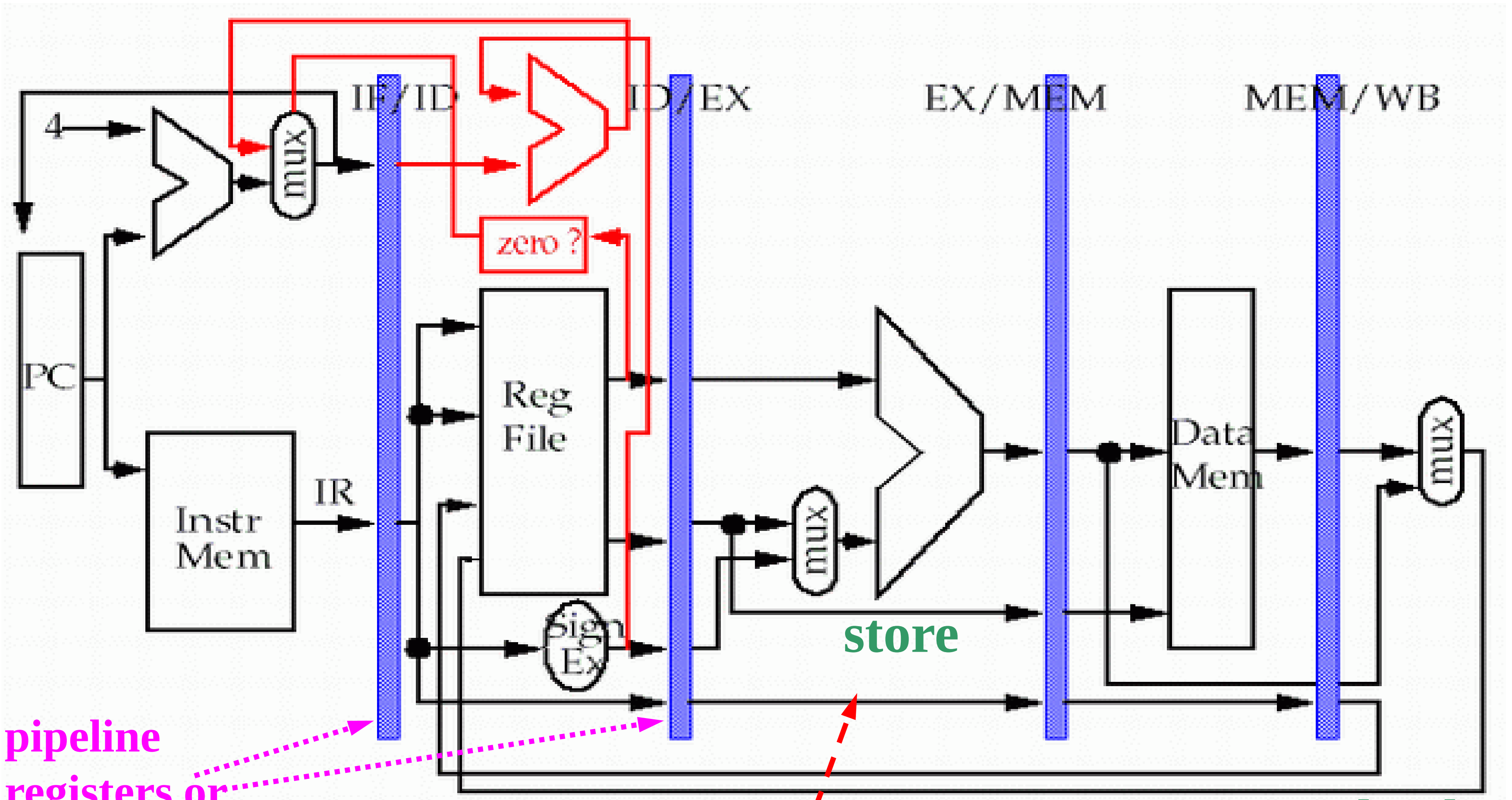


Pipeline Implementation: CPI=1,





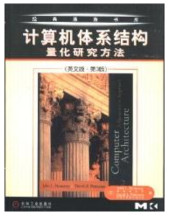
How simple as this ! Really ?



pipeline registers or latches

load

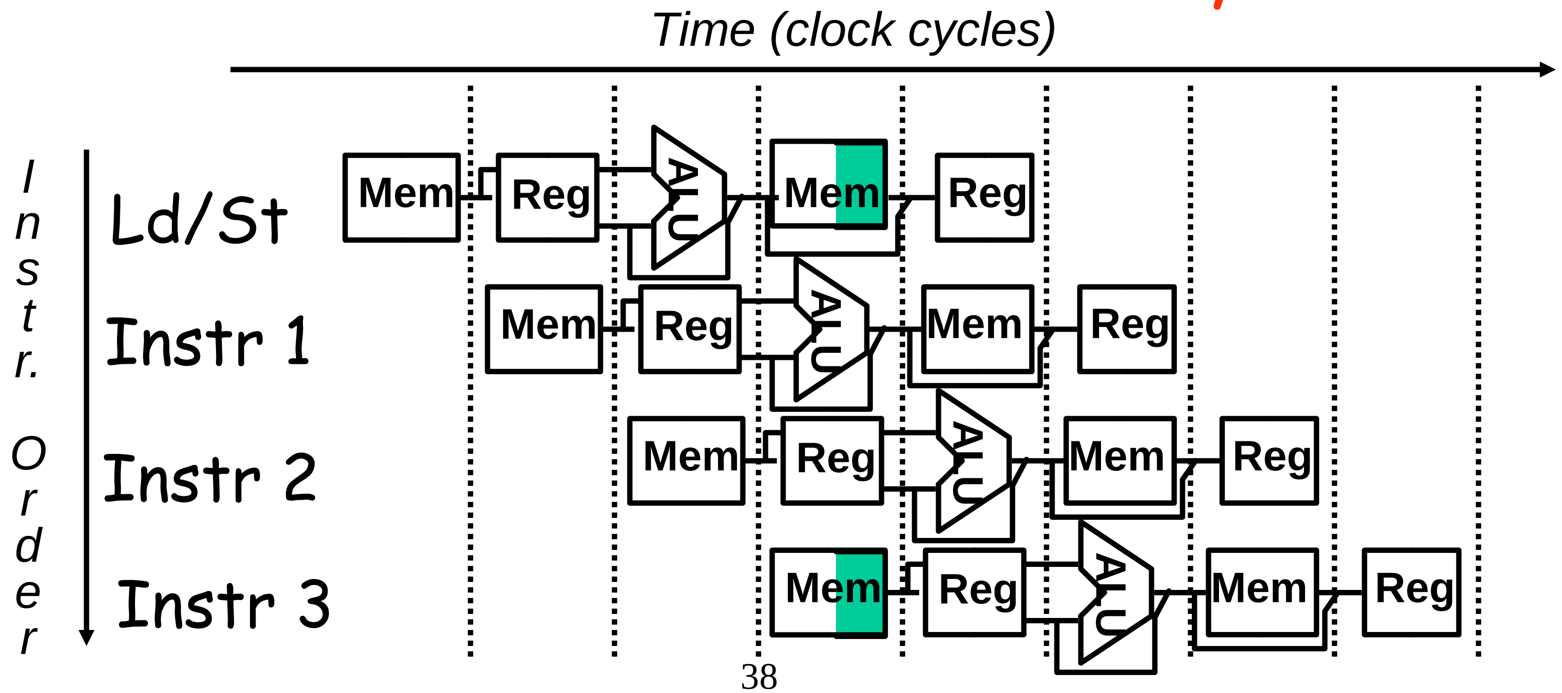
Why need to add this line?

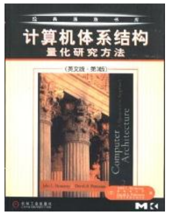


Problems that pipelining introduces

Focus: no different operations with the same data path resource on the same clock cycle. (structure hazard)

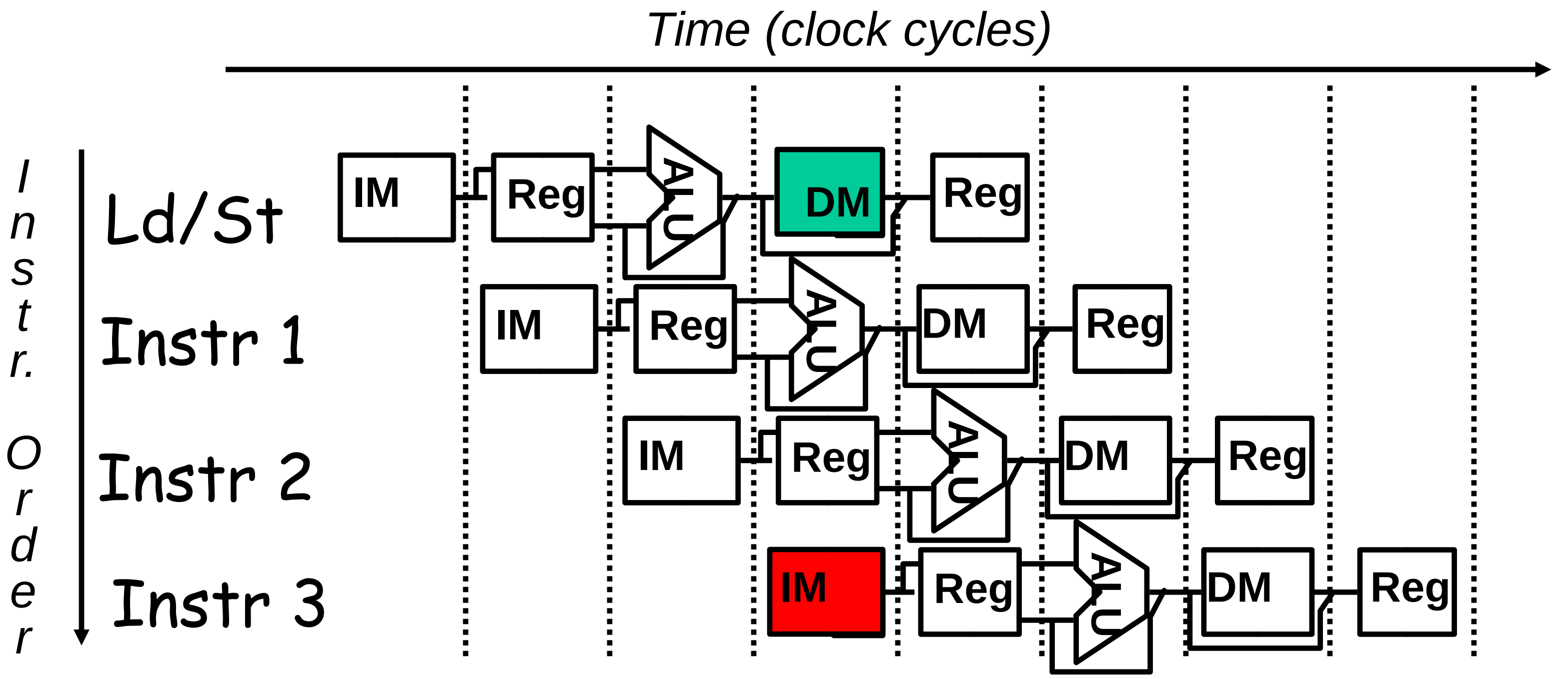
- There is conflict about the memory !



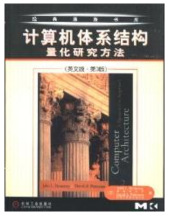


Separate instruction and data memories

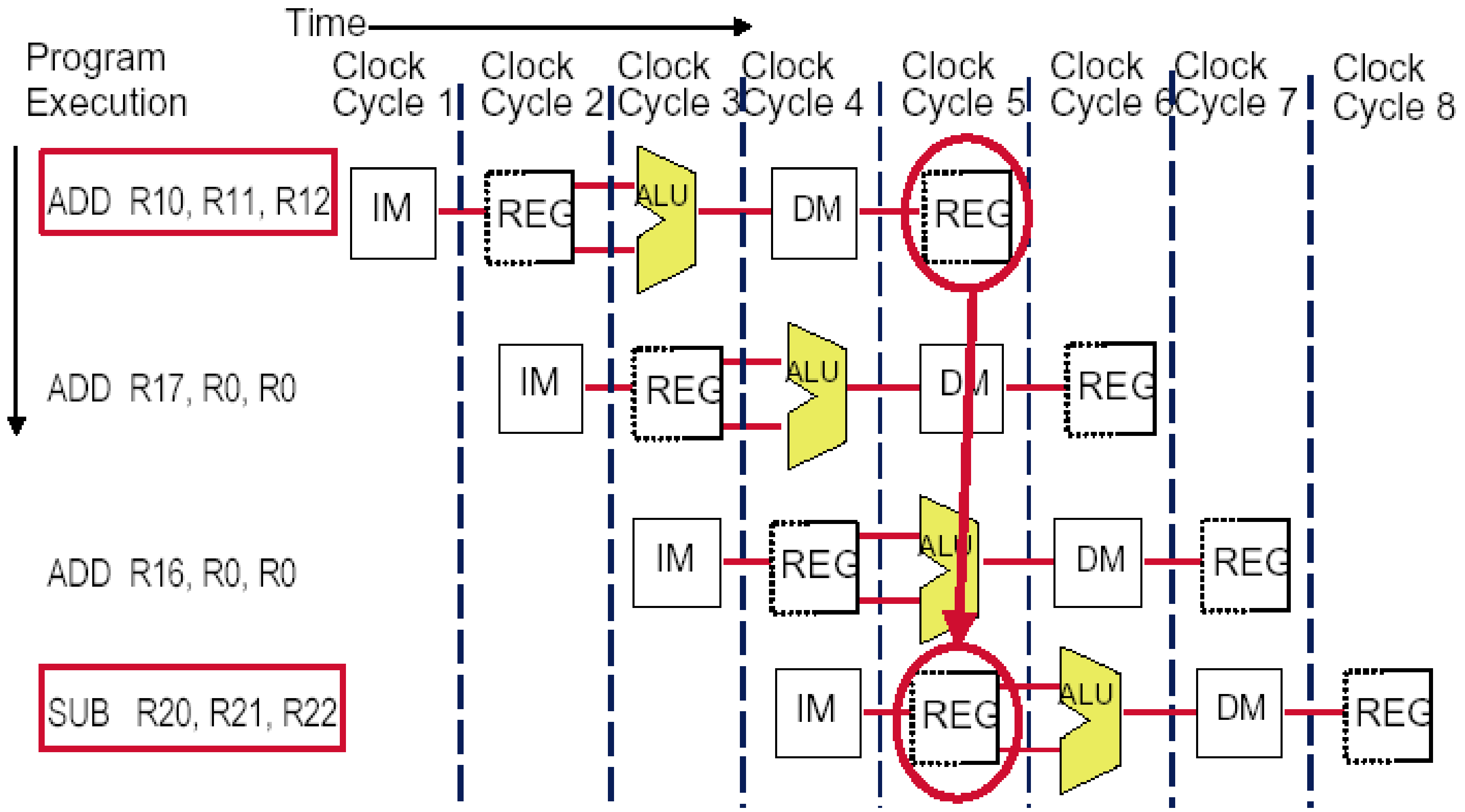
- use split instruction and data cache

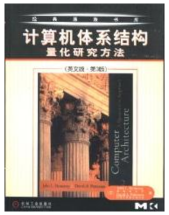


- the memory system must deliver 5 times the bandwidth over the unpipelined version.



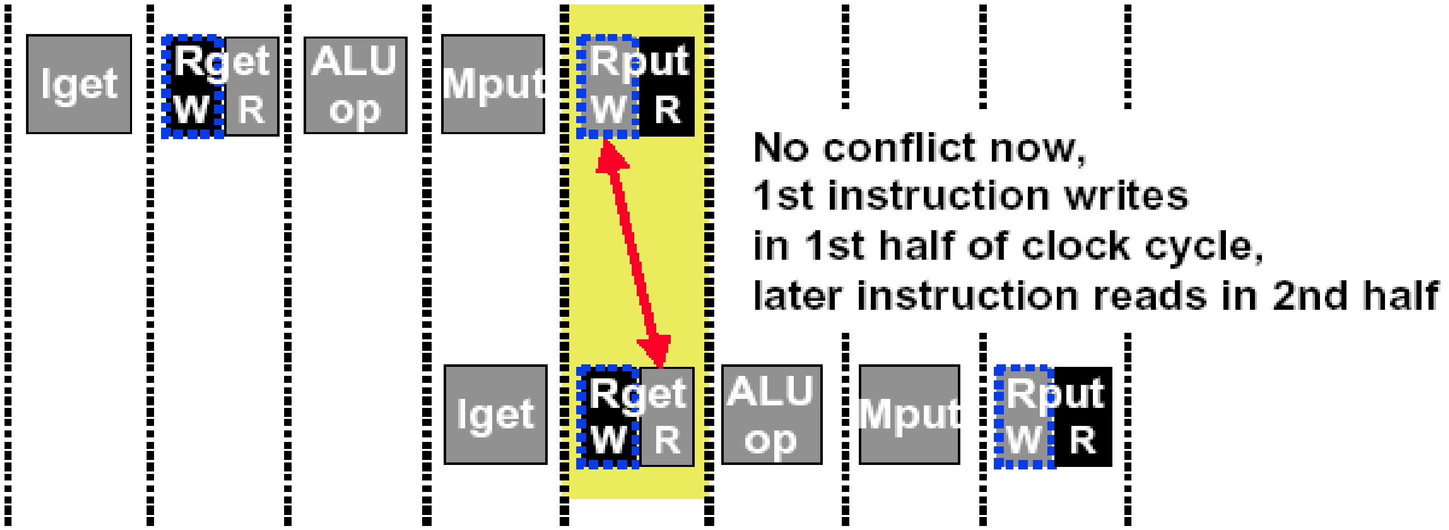
The conflict about the registers !





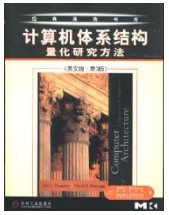
Sometimes we can redesign the resource

- allow WRITE-then-READ in one clock cycle (double pump)



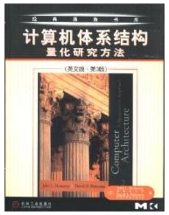
Iget
Rget W R
ALU op
Mput
Rput W R
No conflict now, 1st instruction writes in 1st half of clock cycle, later instruction reads in 2nd half
Iget
Rget W R
ALU op
Mput
Rput W R

  - Two reads and one write required per clock.
  - Need to provide two read port and one write port.
- What happens when a read and a write occur to the same register ? (Data hazard)



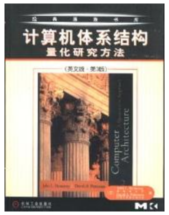
Conflict occurs when PC update

- Must increment and store the PC every clock.
- What happens when meet a branch ?
  - Branches change the value of the PC -- but the condition is not evaluated until ID !
  - If the branch is taken, the instructions fetched behind the branch are invalid !
- This is clearly a serious problem (Control hazard) that needs to be addressed. We will deal it later.



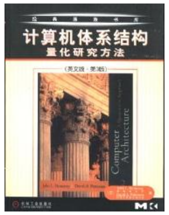
Must latches be engaged ? Yeah !

- Ensure the instructions in different stages do not interfere with one another .
- Through the latches, can the stages be combined one by one to form a pipeline.
- The latches are the pipeline registers , which are much more than those in multi-cycle version
  - IR: IF/ID.IR; ID/EX.IR; EX/DM.IR; DM/WB.IR
  - B: ID/EX.B; EX/DM.B
  - ALUoutput: EX/DM.ALUoutput, DM/WB.ALUoutput
- Any value needed on a later stage must be placed in a register and copied from one register to the next, until it is no longer needed.



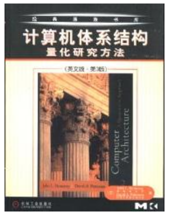
performance issues in pipelining

- Latency: The execution time of each instruction in pipelining does not decrease, instead, always longer than that of unpipelined machine.
- Imbalance among stages reduces performance
- Overhead rise from register delay and clock skew also contribute to the lower limit of machine cycle.
- Pipeline hazards are the major hurdle of pipeline, which prevent the machine from reaching the ideal performance.
- Time to "fill" pipeline and time to "drain" it reduces speedup



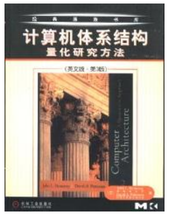
That' all for today.

See you next Time



- 1、If you are asked to design an instruction set architecture for a processor, what are the tasks that you are supposed to accomplish ?

- 2、Please define the terms “CISC” AND “RISC”, and describe their advantages and disadvantages.



- 3、Suppose that there are four types of operations in an application. After making enhancements to the original function units, each type of operation gains performance improvement as shown in the following table.

Operation Type	IC (total 100)	CPI before enhancement	CPI after enhancement
Op1	10	2	1
Op2	30	20	15
Op3	35	10	3
Op4	25	4	1

- 1) What's the enhanced speedup of each operation respectively after improving?
- 2) What's the overall speedup of the application respectively after only improving Op2 or Op4?
  - Hint: Calculate the overall speedup considering only one of the enhanced function units is used. There are 2 situations.
- 3) What's the overall speedup of the application if all the four enhanced function units are used together?