CS152
Computer Architecture and Engineering
Lecture 13

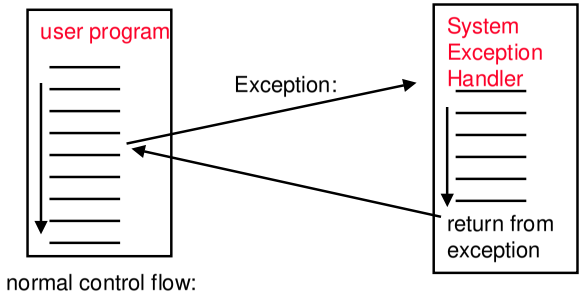
Introduction to Pipelining: Datapath and Control

March 3rd, 2004

John Kubiatowicz (www.cs.berkeley.edu/~kubitron)

lecture slides: http://inst.eecs.berkeley.edu/~cs152/

Recap: Exceptions



normal control flow:
sequential, jumps, branches, calls, returns

° Exception = unprogrammed control transfer
- system takes action to handle the exception
  - must record the address of the offending instruction
  - record any other information necessary to return afterwards
- returns control to user
- must save & restore user state

° Allows constuction of a "user virtual machine"

Recap: Precise Exceptions

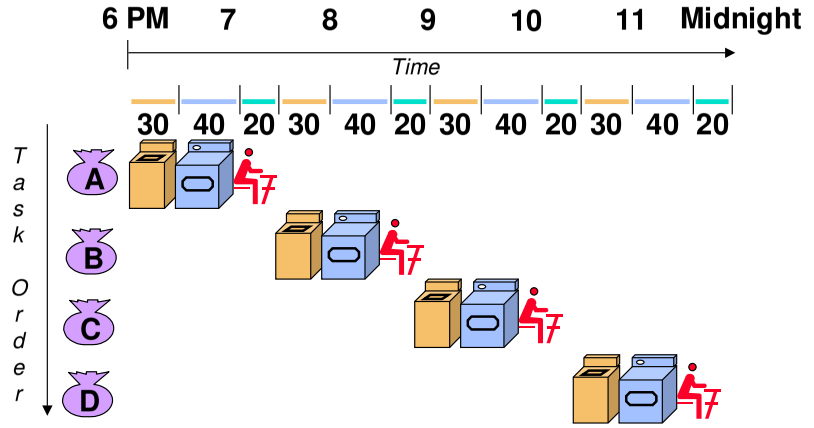
° Precise ⇒ state of the machine is preserved as if program executed up to the offending instruction
- All previous instructions completed
- Offending instruction and all following instructions act as if they have not even started
- Same system code will work on different implementations
- Position clearly established by IBM
- Difficult in the presence of pipelining, out-ot-order execution, ...
- MIPS takes this position

° Imprecise ⇒ system software has to figure out what is where and put it all back together

° Performance goals often lead designers to forsake precise interrupts
- system software developers, user, markets etc. usually wish they had not done this

° Modern techniques for out-of-order execution and branch prediction help implement precise interrupts

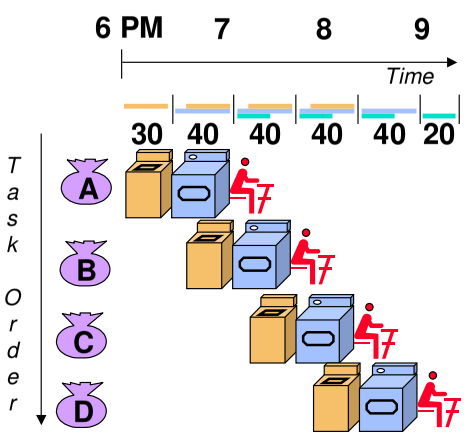
Recap: Sequential Laundry



° Sequential laundry takes 6 hours for 4 loads

° If they learned pipelining, how long would laundry take?

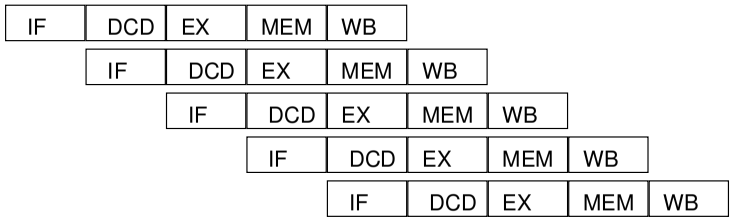
Recap: Pipelining Lessons



- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously using different resources
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup
- Stall for Dependences

Recap: Ideal Pipelining

Assume instructions are completely independent!



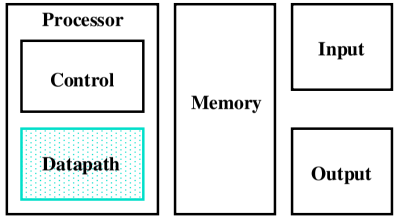
Maximum Speedup ≤ Number of stages

$$\text{Speedup} \leq \frac{\text{Time for unpipelined operation}}{\text{Time for longest stage}}$$

Example: 40ns data path, 5 stages, Longest stage is 10 ns, Speedup ≤ 4

The Big Picture: Where are We Now?

- The Five Classic Components of a Computer

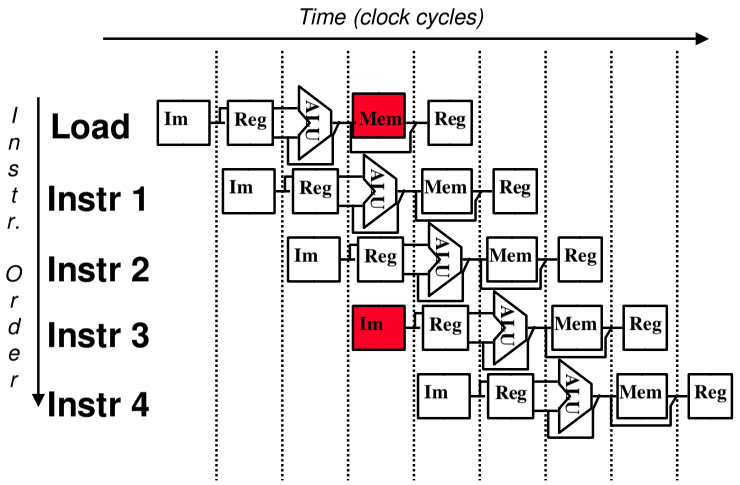


- Today's Topics:
  - Recap last lecture/finish datapath
  - Pipelined Control/ Do it yourself Pipelined Control
  - Administrivia
  - Hazards/Forwarding
  - Exceptions
  - Review MIPS R3000 pipeline

Can pipelining get us into trouble?

- Yes: Pipeline Hazards
  - structural hazards: attempt to use the same resource two different ways at the same time
    - E.g., combined washer/dryer would be a structural hazard or folder busy doing something else (watching TV)
  - data hazards: attempt to use item before it is ready
    - E.g., one sock of pair in dryer and one in washer; can't fold until get sock from washer through dryer
    - instruction depends on result of prior instruction still in the pipeline
  - control hazards: attempt to make a decision before condition is evaulated
    - E.g., washing football uniforms and need to get proper detergent level; need to see after dryer before next load in
    - branch instructions
- Can always resolve hazards by waiting
  - pipeline control must detect the hazard
  - take action (or delay action) to resolve hazards

Single Memory is a Structural Hazard

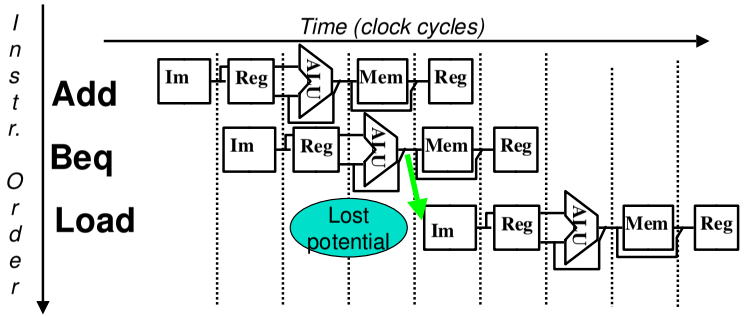


Detection is easy in this case! Fix: Stall Instr 3 fetch.

Structural Hazards limit performance

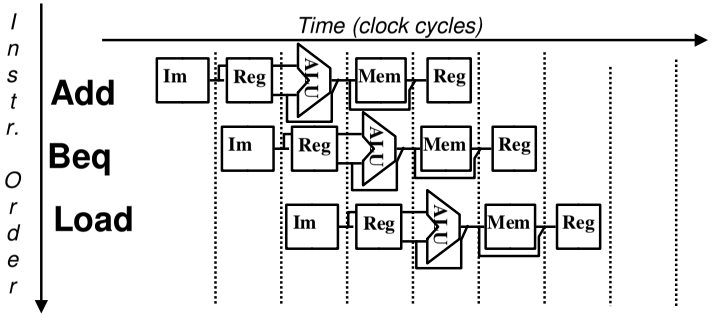
° Example: if 1.3 memory accesses per instruction and only one memory access per cycle then
- average CPI ≥ 1.3
- otherwise resource is more than 100% utilized

Control Hazard Solution #1: Stall



° Stall: wait until decision is clear
° Impact: 2 lost cycles (i.e. 3 clock cycles per branch instruction) => slow
° Move decision to end of decode
- save 1 cycle per branch

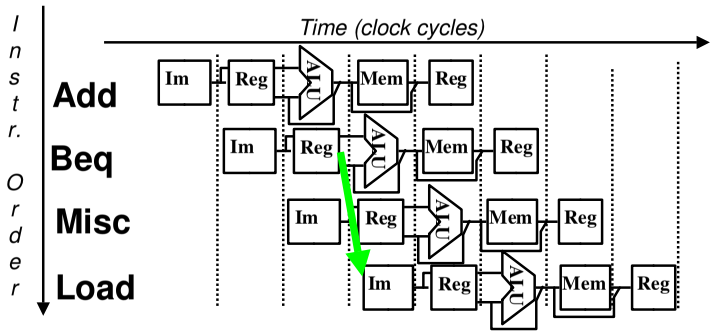
Control Hazard Solution #2: Predict



° Predict: guess one direction then back up if wrong
° Impact: 0 lost cycles per branch instruction if right, 1 if wrong (right - 50% of time)
- Need to "Squash" and restart following instruction if wrong
- Produce CPI on branch of (1 *.5 + 2 * .5) = 1.5
- Total CPI might then be: 1.5 * .2 + 1 * .8 = 1.1 (20% branch)

° More dynamic scheme: history of 1 branch (- 90%)

Control Hazard Solution #3: Delayed Branch



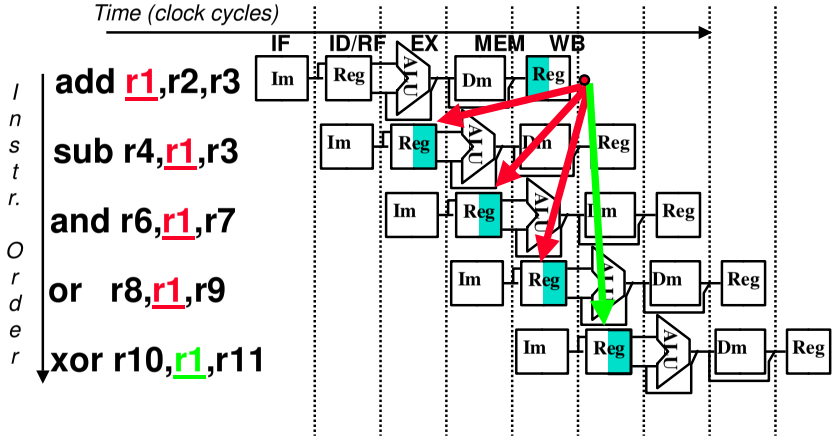
- Delayed Branch: Redefine branch behavior (takes place after next instruction)
- Impact: 0 clock cycles per branch instruction if can find instruction to put in "slot" (- 50% of time)
- As launch more instruction per clock cycle, less useful

Data Hazard on r1: Read after write hazard (RAW)

```
add r1,r2,r3
sub r4,r1,r3
and r6,r1,r7
or  r8,r1,r9
xor r10,r1,r11
```

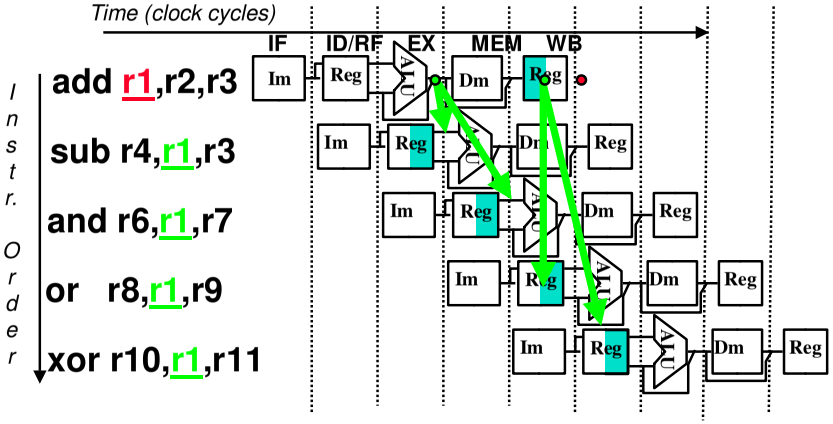

Data Hazard on r1: Read after write hazard (RAW)

- Dependencies backwards in time are hazards



Data Hazard Solution: Forwarding

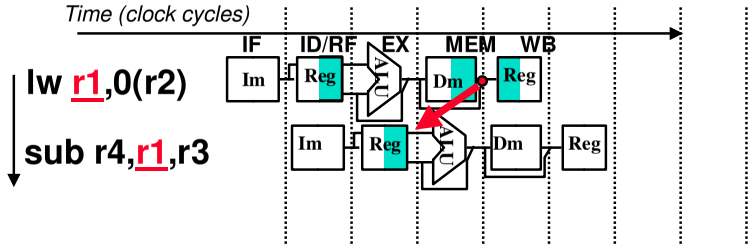
- "Forward" result from one stage to another



- "or" OK if define read/write properly

Forwarding (or Bypassing): What about Loads?

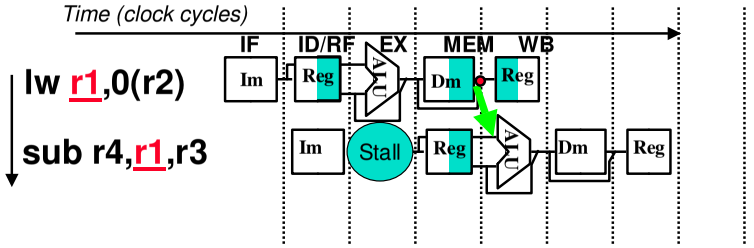
- Dependencies backwards in time are hazards



- Can't solve with forwarding:
- Must delay/stall instruction dependent on loads

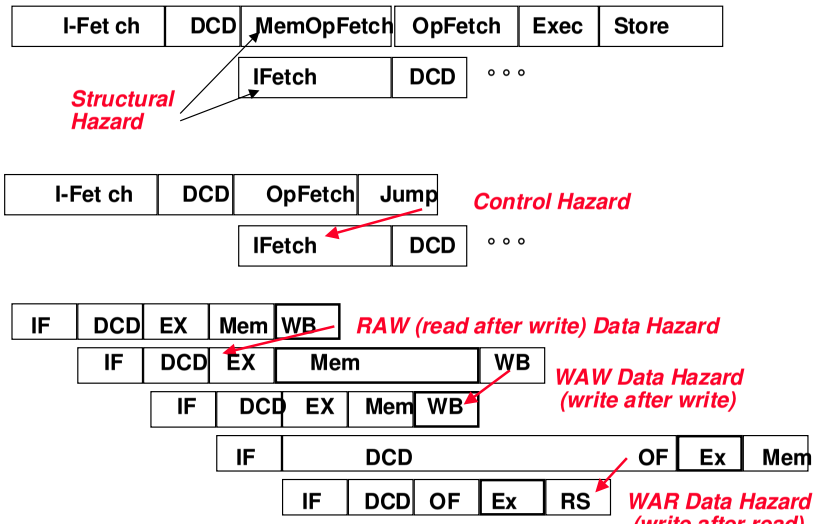
Forwarding (or Bypassing): What about Loads

- Dependencies backwards in time are hazards



- Can't solve with forwarding:
- Must delay/stall instruction dependent on loads

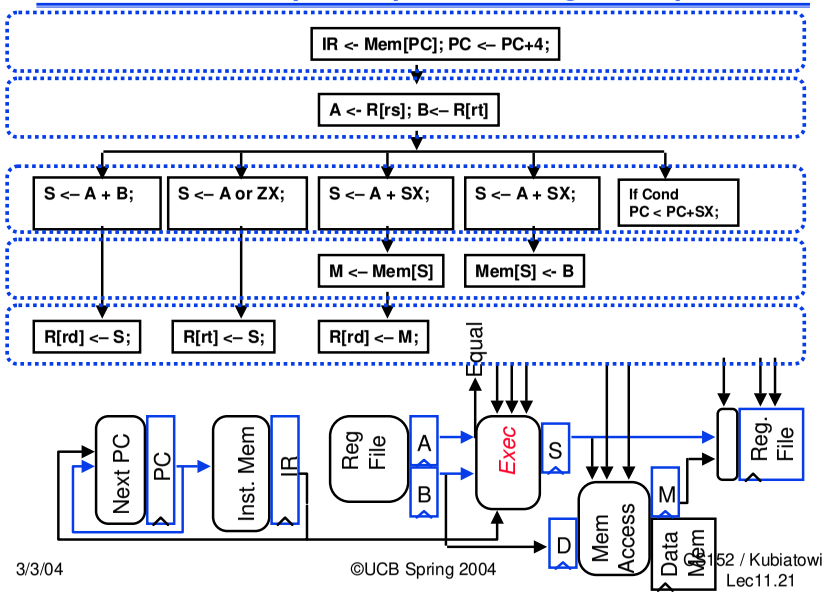
Recap: Data Hazards



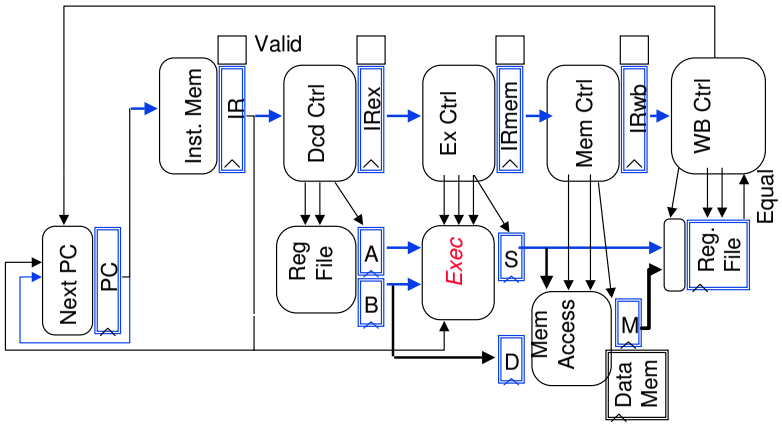
Designing a Pipelined Processor

- Go back and examine your datapath and control diagram
- associated resources with states
- ensure that flows do not conflict, or figure out how to resolve conflicts
- assert control in appropriate stage

Control and Datapath: Split state diag into 5 pieces

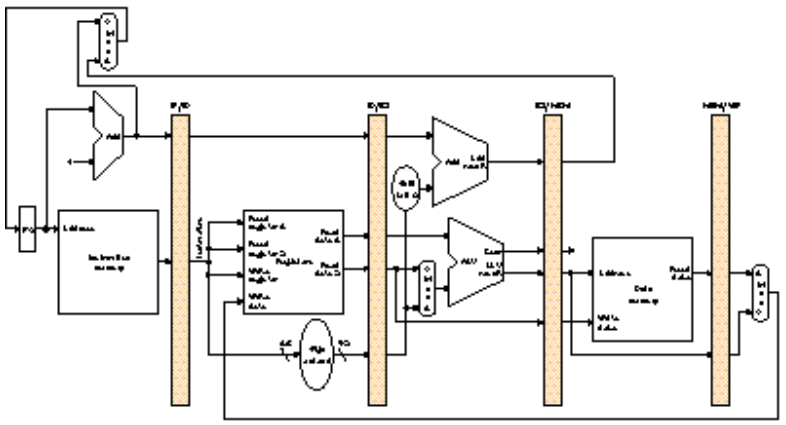


Pipelined Processor (almost) for slides



° What happens if we start a new instruction every cycle?

Pipelined Datapath (as in book): hard to read



Administrivia

° Midterm I on Wednesday 3/10
- 5:30 - 8:30 in 306 Soda Hall
- Bring a Calculator!
- One 8 1/2 by 11 page (both sides) of notes
- *Make up exam: Tuesday 5:30 – 8:30 in 606 Soda Hall*

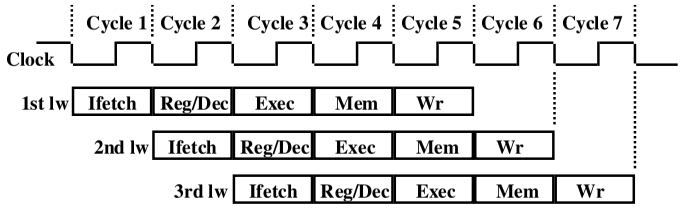
° Meet at LaVal's afterwards for Pizza
- North-side Lavals!
- I'll Buy!

° Materials through Chapter 5, some of Chapter 6, Appendix A, B & C
- Should understand single-cycle processor
- multicycle processor, including exceptions

° Get moving on Lab 3!
- This is a hard lab, since there are so many new things

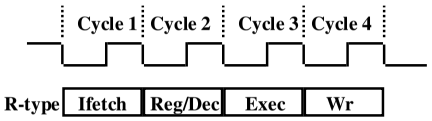
° Homework 4 out later today

Pipelining the Load Instruction



° The five independent functional units in the pipeline datapath are:
- Instruction Memory for the Ifetch stage
- Register File's Read ports (bus A and busB) for the Reg/Dec stage
- ALU for the Exec stage
- Data Memory for the Mem stage
- Register File's Write port (bus W) for the Wr stage

The Four Stages of R-type



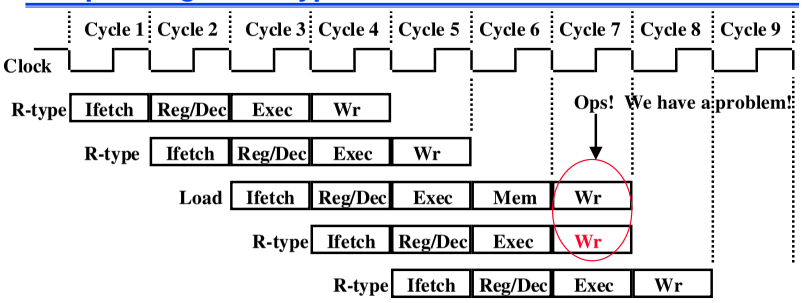
° Ifetch: Instruction Fetch
- Fetch the instruction from the Instruction Memory

° Reg/Dec: Registers Fetch and Instruction Decode

° Exec:
- ALU operates on the two register operands
- Update PC

° Wr: Write the ALU output back to the register file

Pipelining the R-type and Load Instruction

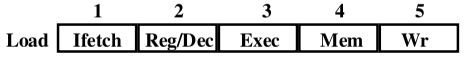


° We have pipeline conflict or structural hazard:
- Two instructions try to write to the register file at the same time!
- Only one write port

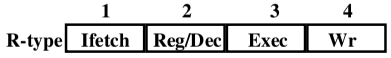
Important Observation

° Each functional unit can only be used once per instruction

° Each functional unit must be used at the same stage for all instructions:
- Load uses Register File's Write Port during its 5th stage

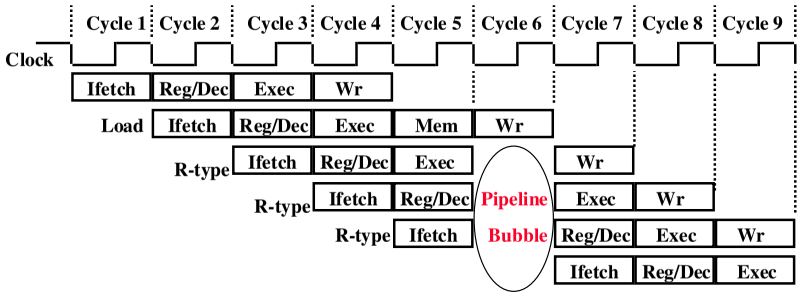


- R-type uses Register File's Write Port during its 4th stage



° 2 ways to solve this pipeline hazard.

Solution 1: Insert "Bubble" into the Pipeline

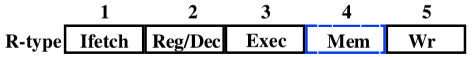


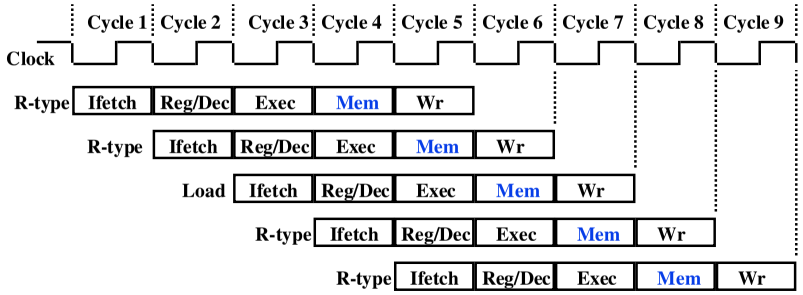
° Insert a "bubble" into the pipeline to prevent 2 writes at the same cycle
- The control logic can be complex.
- Lose instruction fetch and issue opportunity.

° No instruction is started in Cycle 6!

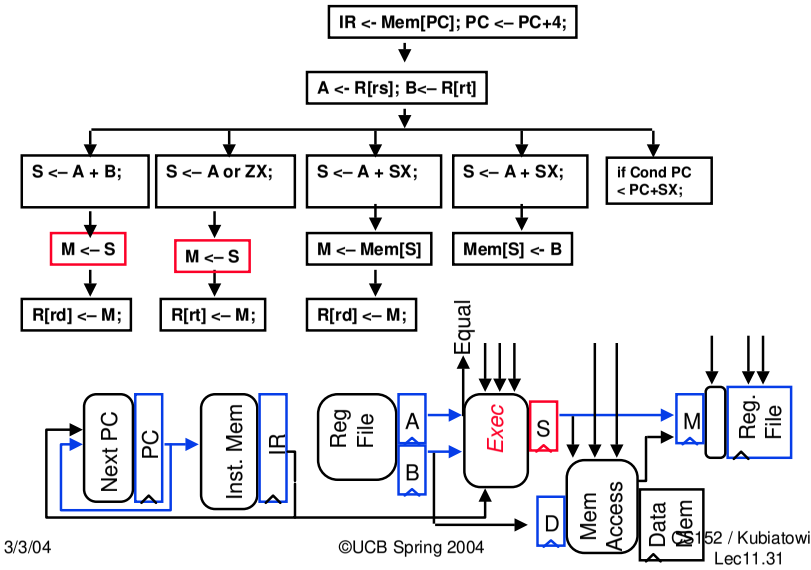
Solution 2: Delay R-type's Write by One Cycle

° Delay R-type's register write by one cycle:
- Now R-type instructions also use Reg File's write port at Stage 5
- Mem stage is a NOOP stage: nothing is being done.

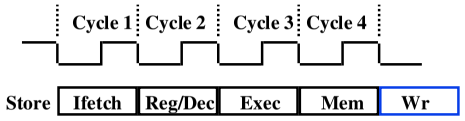




Modified Control & Datapath



The Four Stages of Store



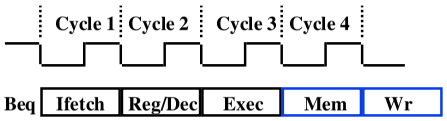
° Ifetch: Instruction Fetch
- Fetch the instruction from the Instruction Memory

° Reg/Dec: Registers Fetch and Instruction Decode

° Exec: Calculate the memory address

° Mem: Write the data into the Data Memory

The Three Stages of Beq

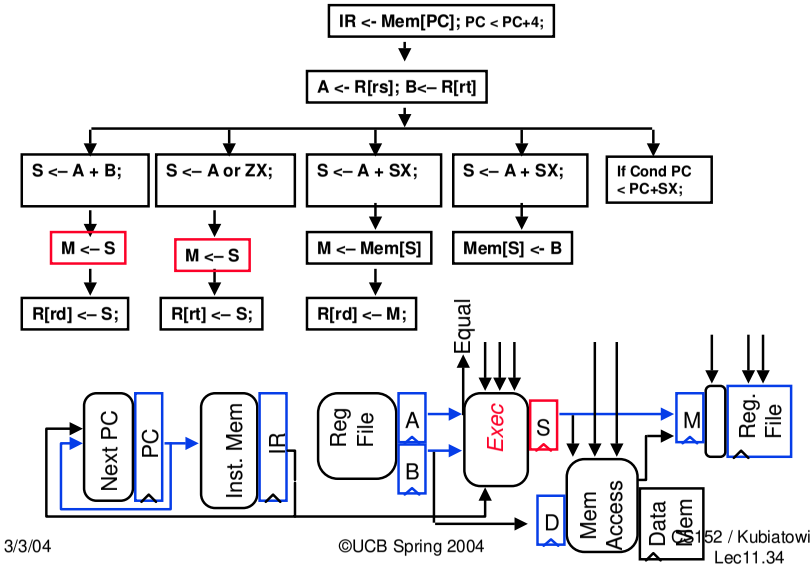


° **Ifetch: Instruction Fetch**
- Fetch the instruction from the Instruction Memory

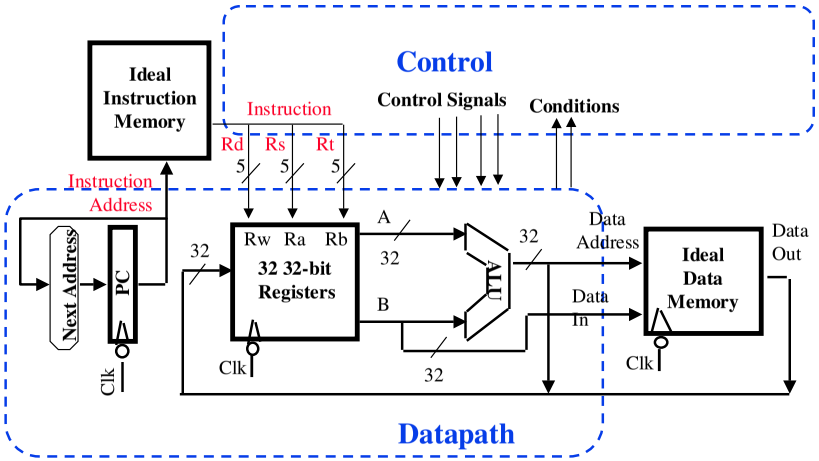
° **Reg/Dec:**
- Registers Fetch and Instruction Decode

° **Exec:**
- compares the two register operand,
- select correct branch target address
- latch into PC

Control Diagram

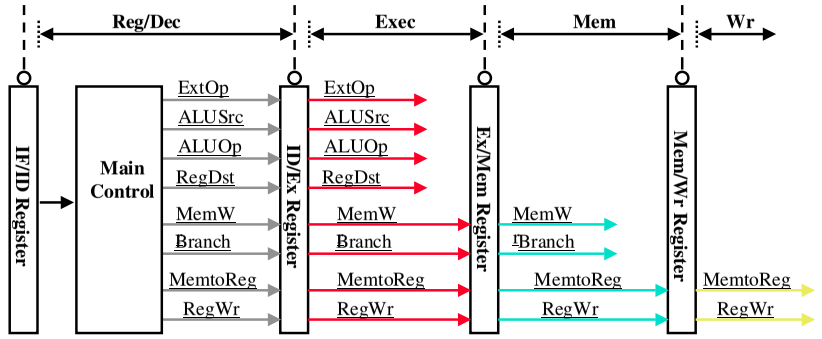


Recall: Single cycle control!

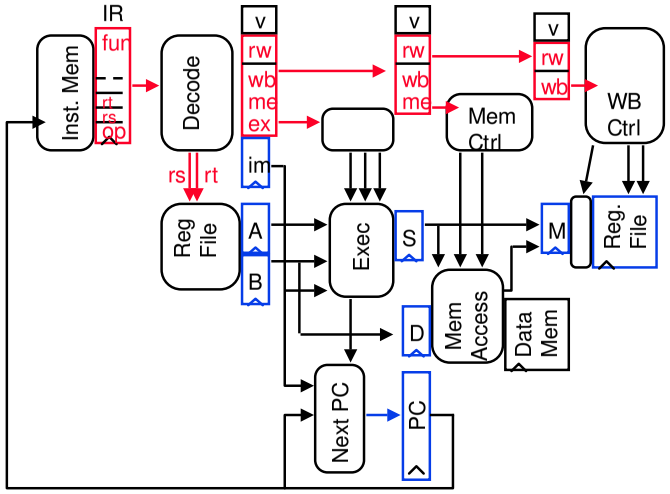


Data Stationary Control

° **The Main Control generates the control signals during Reg/Dec**
- Control signals for Exec (ExtOp, ALUSrc, ...) are used 1 cycle later
- Control signals for Mem (MemWr Branch) are used 2 cycles later
- Control signals for Wr (MemtoReg MemWr) are used 3 cycles later



Datapath + Data Stationary Control

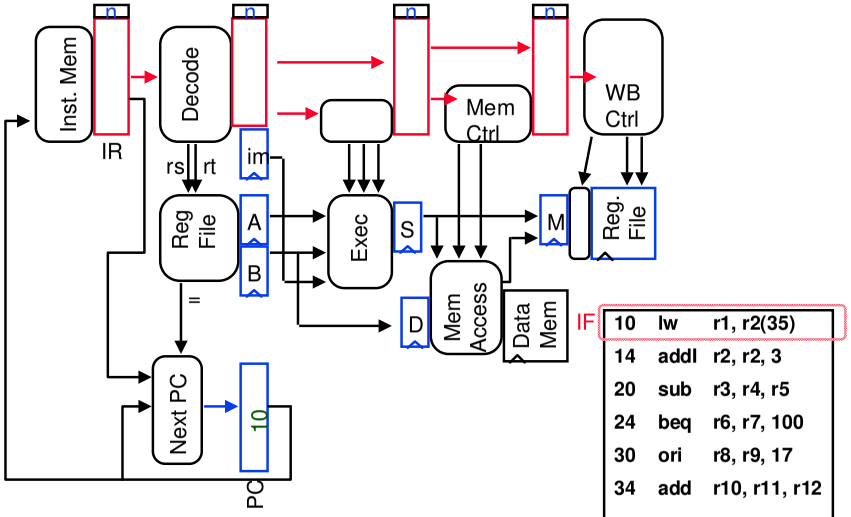


Let's Try it Out

10	lw	r1, r2(35)
14	addl	r2, r2, 3
20	sub	r3, r4, r5
24	beq	r6, r7, 100
30	ori	r8, r9, 17
34	add	r10, r11, r12
100	and	r13, r14, 15

these addresses are octal

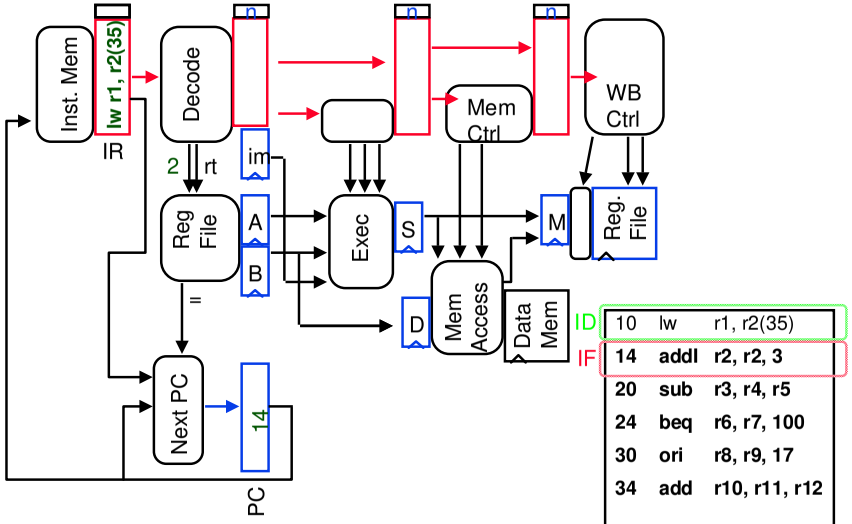
Start: Fetch 10



IF

10	lw	r1, r2(35)
14	addl	r2, r2, 3
20	sub	r3, r4, r5
24	beq	r6, r7, 100
30	ori	r8, r9, 17
34	add	r10, r11, r12
100	and	r13, r14, 15

Fetch 14, Decode 10



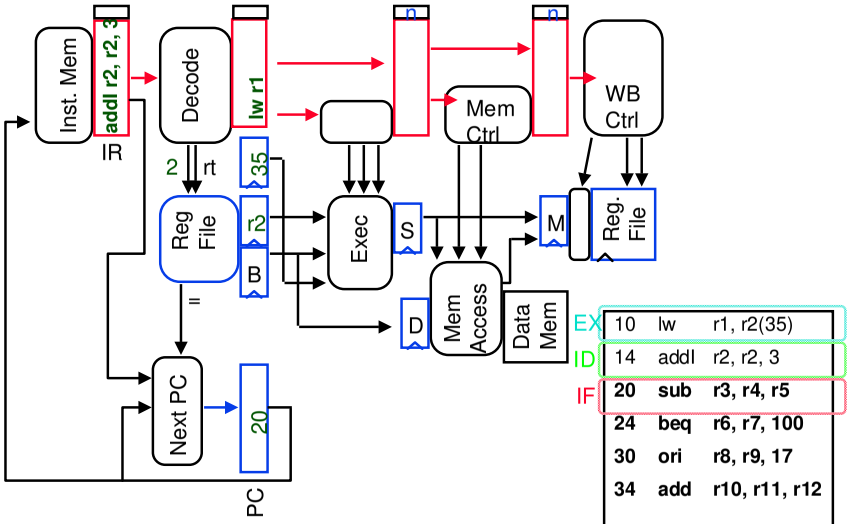
ID

10	lw	r1, r2(35)

IF

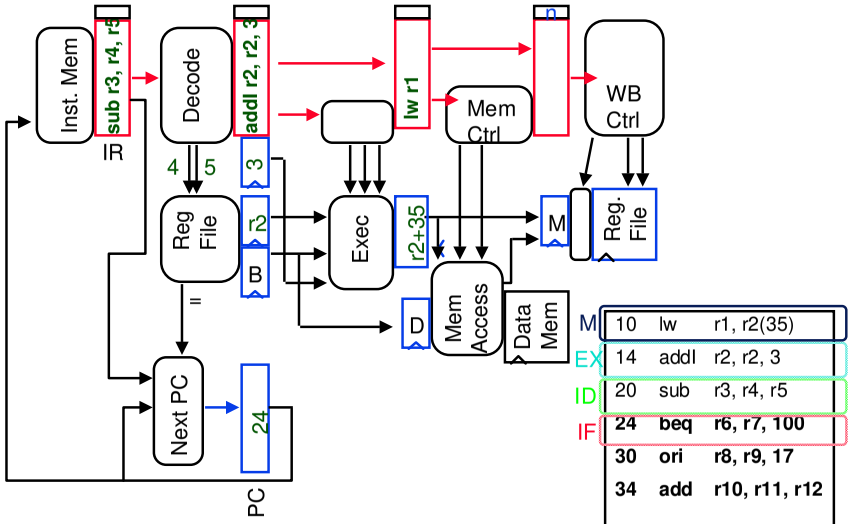
14	addl	r2, r2, 3
20	sub	r3, r4, r5
24	beq	r6, r7, 100
30	ori	r8, r9, 17
34	add	r10, r11, r12
100	and	r13, r14, 15

Fetch 20, Decode 14, Exec 10



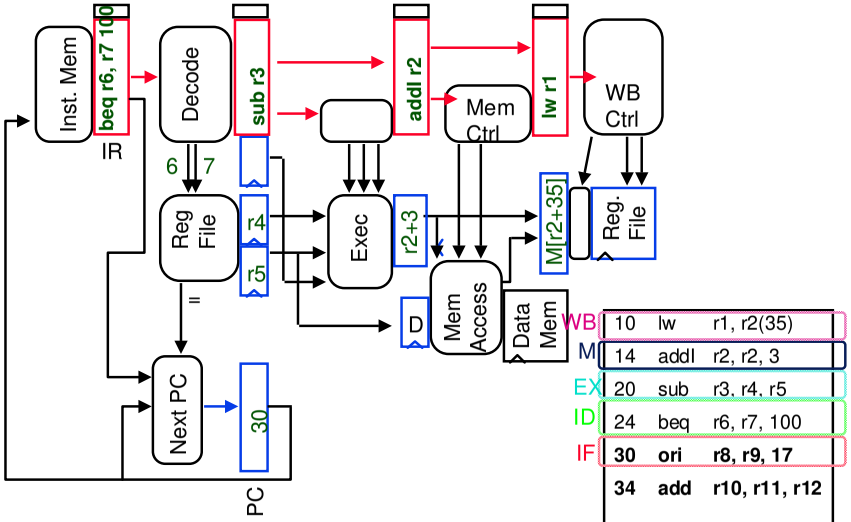
©UCB Spring 2004

Fetch 24, Decode 20, Exec 14, Mem 10



©UCB Spring 2004

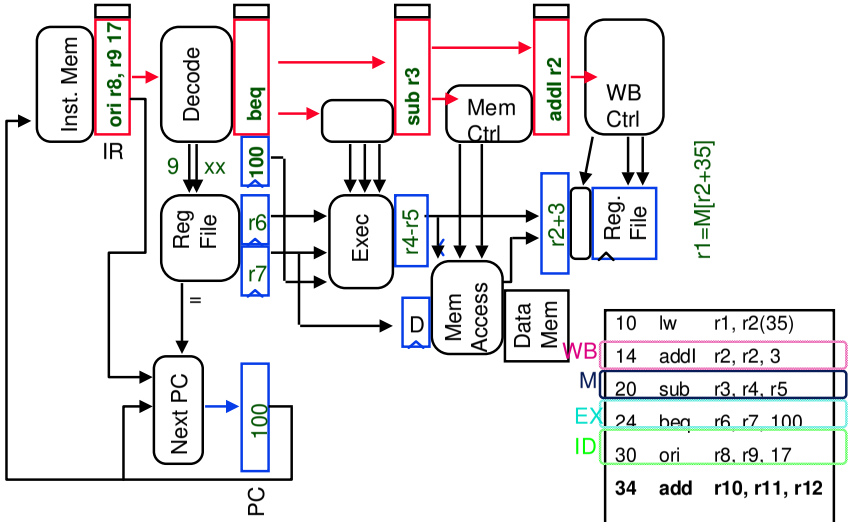
Fetch 30, Dcd 24, Ex 20, Mem 14, WB 10



Note Delayed Branch: always execute `ori` after `beq`

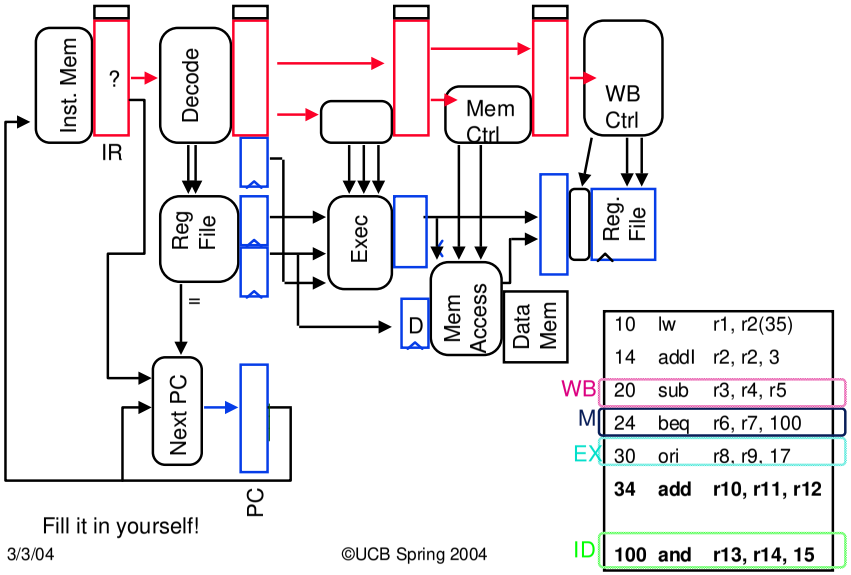
©UCB Spring 2004

Fetch 100, Dcd 30, Ex 24, Mem 20, WB 14

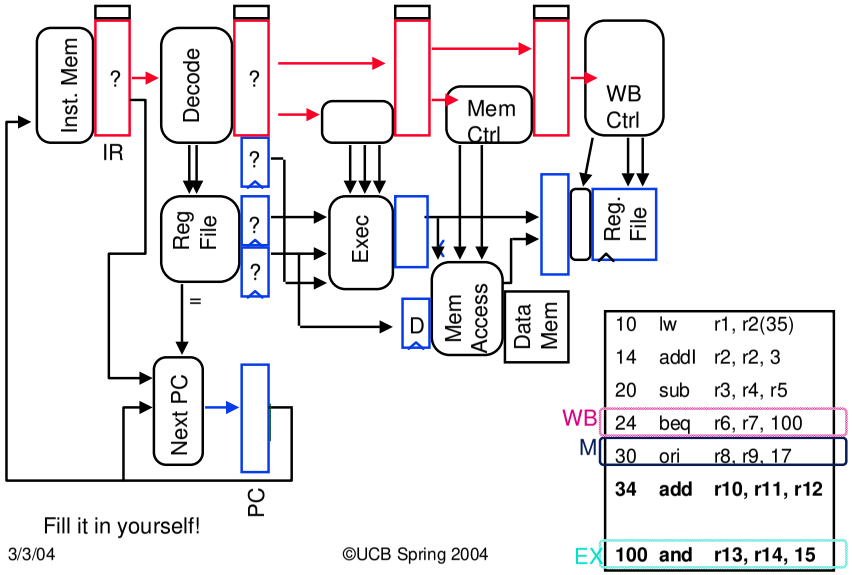


©UCB Spring 2004

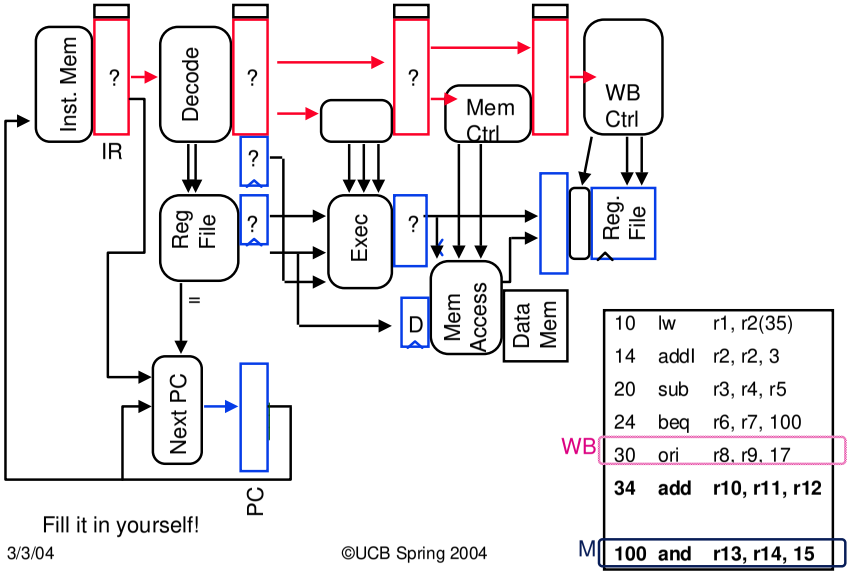
Fetch 104, Dcd 100, Ex 30, Mem 24, WB 20



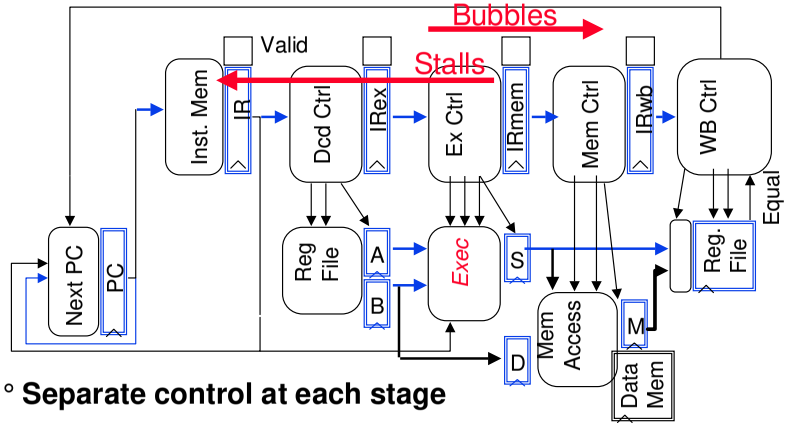
Fetch 110, Dcd 104, Ex 100, Mem 30, WB 24



Fetch 114, Dcd 110, Ex 104, Mem 100, WB 30



Pipelined Processor

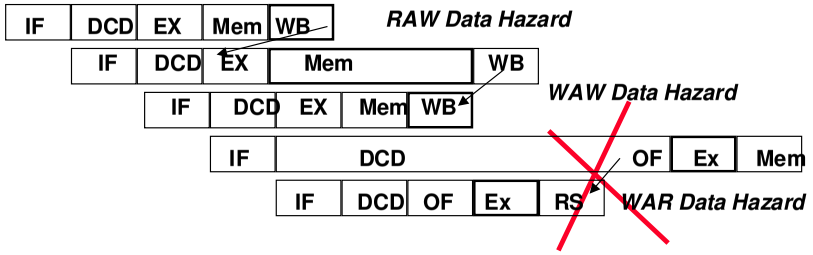


- Separate control at each stage
- Stalls propagate backwards to freeze previous stages
- Bubbles in pipeline introduced by placing “Noops” into local stage, stall previous stages.

Recap: Data Hazards

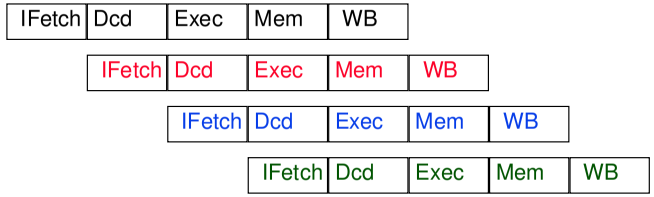
° Avoid some "by design"
- eliminate WAR by always fetching operands early (DCD) in pipe
- eleminate WAW by doing all WBs in order (last stage, static)

° Detect and resolve remaining ones
- stall or forward (if possible)



Is CPI = 1 for our pipeline?

° Remember that CPI is an "Average # cycles/inst



° CPI here is 1, since the average throughput is 1 instruction every cycle.

° What if there are stalls or multi-cycle execution?

° Usually CPI > 1. How close can we get to 1??

Summary

° What makes it easy
- all instructions are the same length
- just a few instruction formats
- memory operands appear only in loads and stores

° Hazards limit performance
- Structural: need more HW resources
- Data: need forwarding, compiler scheduling
- Control: early evaluation & PC, delayed branch, prediction

° Data hazards must be handled carefully:
- RAW data hazards handled by forwarding
- WAW and WAR hazards don't exist in 5-stage pipeline

° MIPS I instruction set architecture made pipeline visible (delayed branch, delayed load)

Summary: Where this class is going

° We'll build a simple pipeline and look at these issues
- Lab 4 ⇒ Pipelined Processor
- Lab 5 ⇒ With caches

° We'll talk about modern processors and what's really hard:
- Branches (control hazards) are really hard!
- Exception handling
- Trying to improve performance with out-of-order execution, etc.
- Trying to get CPI < 1 (Superscalar execution)