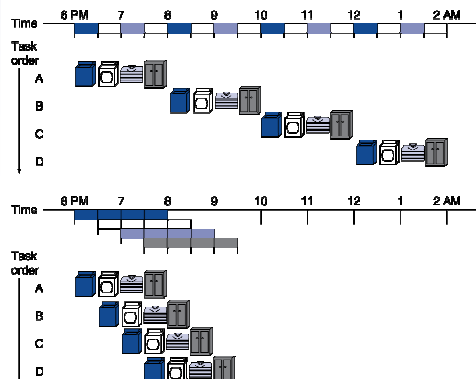


## Review: Pipelining



### Pipelining Analogy

- Pipelined laundry: overlapping execution
  - Parallelism improves performance



- Four loads:
  - Speedup  
 $= 8 / 3.5 = 2.3$

- Non-stop:
  - Speedup  
 $= 2n / 0.5n + 1.5 \approx 4$   
 $= \text{number of stages}$

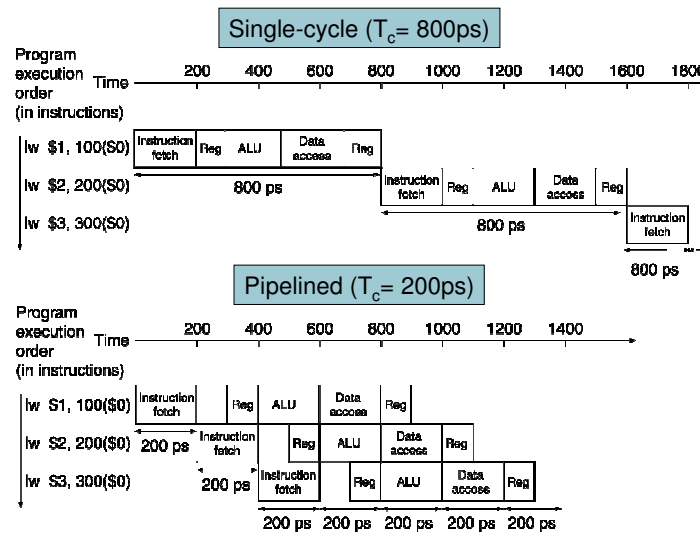


## MIPS Pipeline

- Five stages, one step per stage
  1. IF: Instruction fetch from memory
  2. ID: Instruction decode & register read
  3. EX: Execute operation or calculate address
  4. MEM: Access memory operand
  5. WB: Write result back to register



## Pipeline Performance



## Pipeline Speedup

- If all stages are balanced
  - i.e., all take the same time
  - Time between instructions<sub>pipelined</sub>  
=  $\frac{\text{Time between instructions}_{\text{nonpipelined}}}{\text{Number of stages}}$
- If not balanced, speedup is less
- Speedup due to increased throughput
  - Latency (time for each instruction) does not decrease



Chapter 4 — The Processor — 5

## Hazards

- Situations that prevent starting the next instruction in the next cycle
- Structure hazards
  - A required resource is busy
- Data hazard
  - Need to wait for previous instruction to complete its data read/write
- Control hazard
  - Deciding on control action depends on previous instruction



Chapter 4 — The Processor — 6

## Structure Hazards

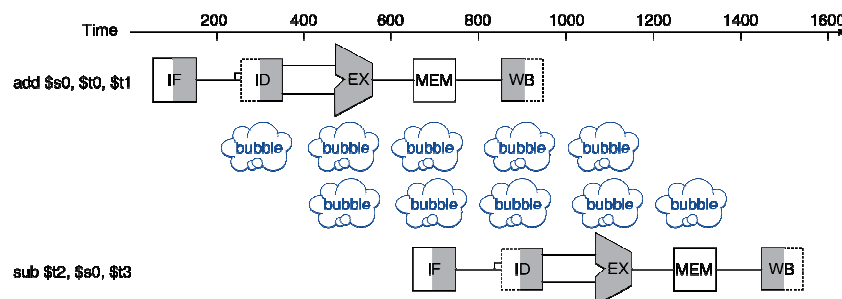
- Conflict for use of a resource
- In MIPS pipeline with a single memory
  - Load/store requires data access
  - Instruction fetch would have to *stall* for that cycle
    - Would cause a pipeline “bubble”
- Hence, pipelined datapaths require separate instruction/data memories
  - Or separate instruction/data caches



Chapter 4 — The Processor — 7

## Data Hazards

- An instruction depends on completion of data access by a previous instruction
  - add \$s0, \$t0, \$t1
  - sub \$t2, \$s0, \$t3



Chapter 4 — The Processor — 8

## Control Hazards

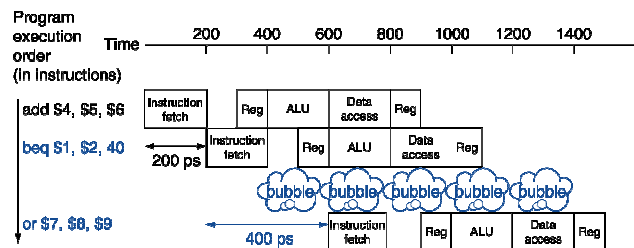
- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipeline can't always fetch correct instruction
    - Still working on ID stage of branch
- In MIPS pipeline
  - Need to compare registers and compute target early in the pipeline
  - Add hardware to do it in ID stage



Chapter 4 — The Processor — 9

## Stall on Branch

- Wait until branch outcome determined before fetching next instruction



Chapter 4 — The Processor — 10

## Branch Prediction

- Longer pipelines can't readily determine branch outcome early
  - Stall penalty becomes unacceptable
- Predict outcome of branch
  - Only stall if prediction is wrong
- In MIPS pipeline
  - Can predict branches not taken
  - Fetch instruction after branch, with no delay



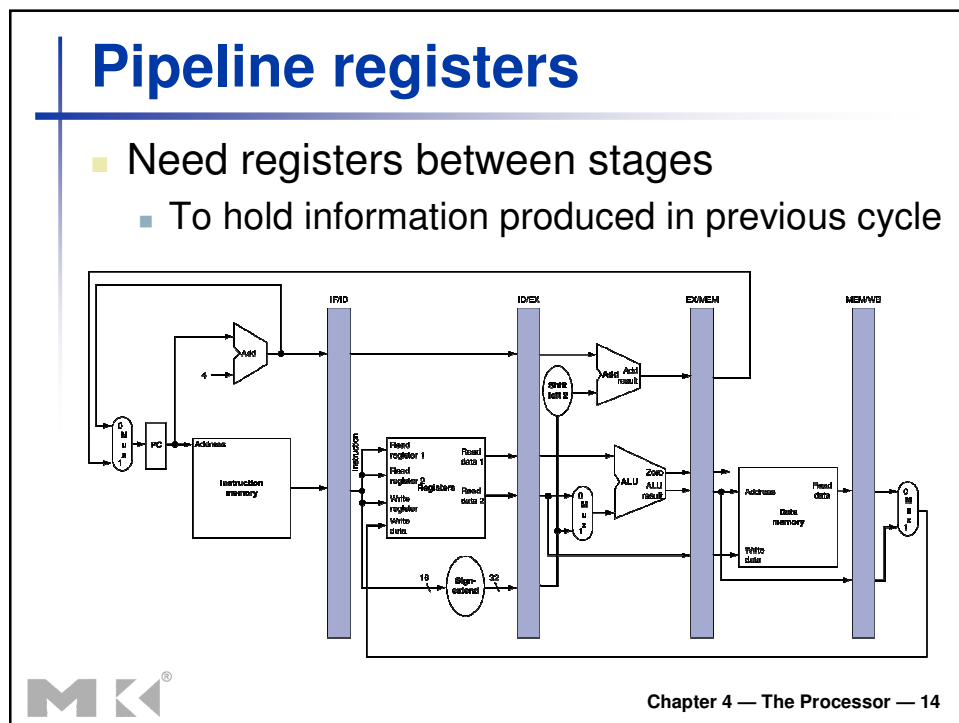
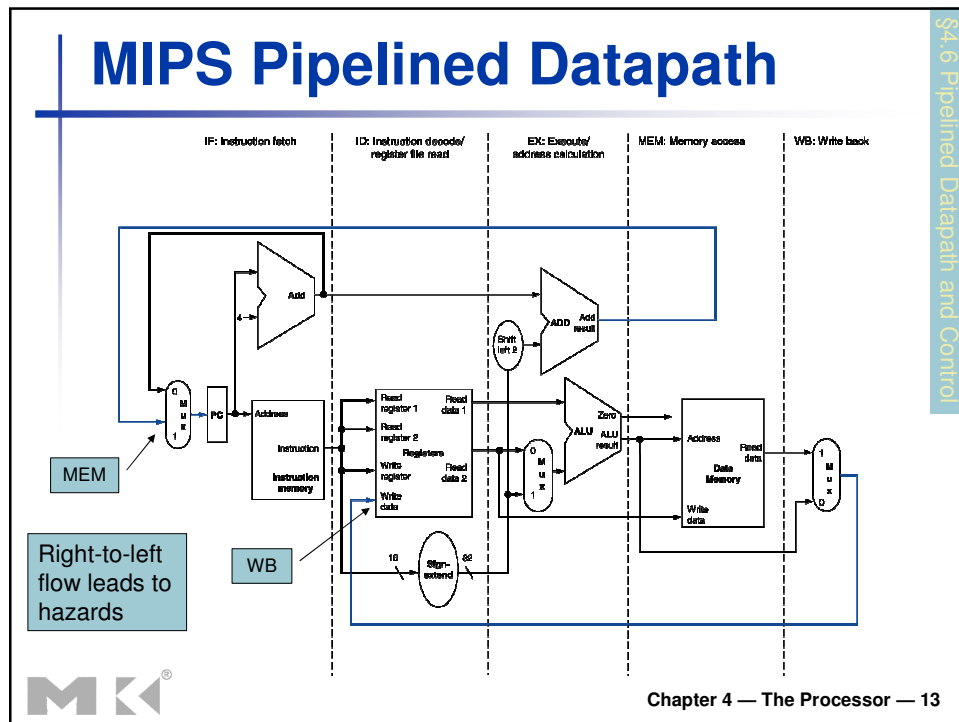
Chapter 4 — The Processor — 11

## More-Realistic Branch Prediction

- Static branch prediction
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - Predict backward branches taken
    - Predict forward branches not taken
- Dynamic branch prediction
  - Hardware measures actual branch behavior
    - e.g., record recent history of each branch
  - Assume future behavior will continue the trend
    - When wrong, stall while re-fetching, and update history

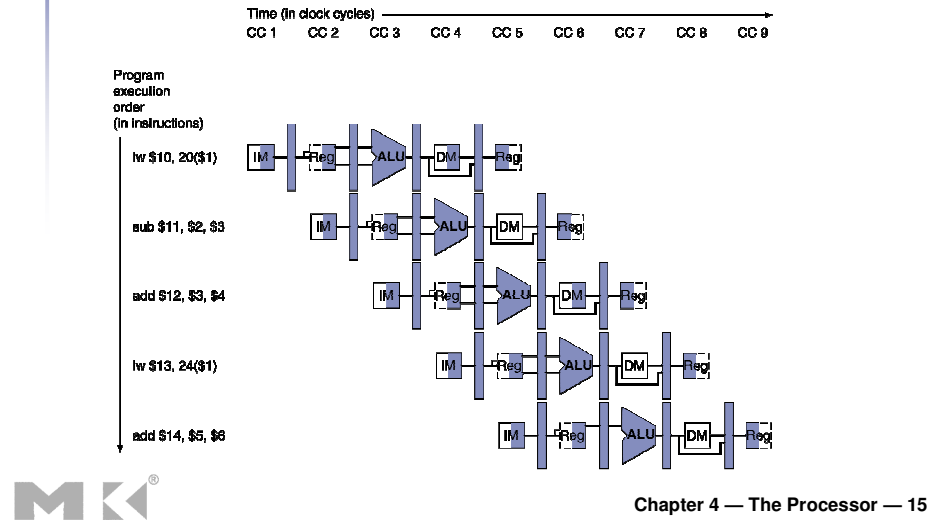


Chapter 4 — The Processor — 12



## Multi-Cycle Pipeline Diagram

### ■ Form showing resource usage



## Multi-Cycle Pipeline Diagram

### ■ Traditional form

