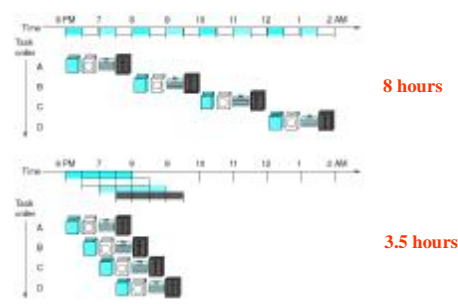


## Chapter Six

- What is pipelining
  - Implementation technique that exploits parallelism among instructions in a sequential instruction stream
  - Multiple instructions overlapped in execution
  - Invisible to programmer
- Make processor faster
  - Same latency
  - Improve throughput
  - e.g. Laundry

### Pipelining



### Pipeline instruction execution

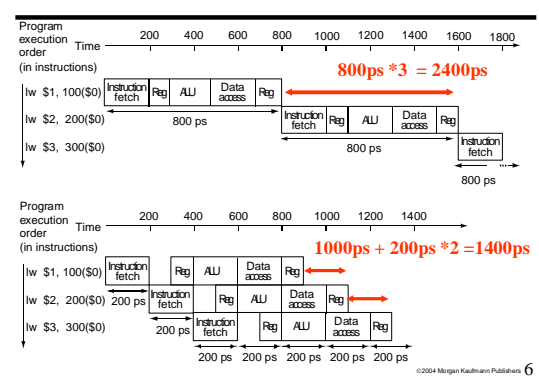
- MIPS instruction
  - Take five steps
    - Fetch instruction from memory
    - Read registers; decode instruction
    - Execute operation; calculate address
    - Access an operand in memory
    - Write results into a register

### Pipeline instruction execution

- Single cycle design
  - Clock cycle decided by worst-case clock cycle
  - e.g. 800ps
- Pipeline
  - Execution clock cycle decided by worst-case clock cycle
  - e.g. 200ps

Instruction Class	Instruction Fetch	Register Read	ALU Operation	Operand Access	Register Write	Latency
Load word	200ps	100ps	200ps	200ps	100ps	800ps
Store word	200ps	100ps	200ps	200ps	0ps	700ps
R-Format branch	200ps	100ps	200ps	200ps	100ps	800ps

### Improve performance by increasing instruction throughput



## Pipelining

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

©2004 Morgan Kaufmann Publishers 7

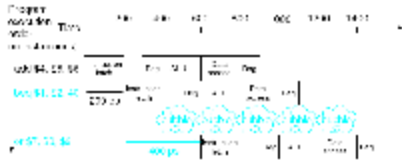
## Pipelining

- What makes it hard?
  - structural hazards
    - Hardware cannot support combination of instructions
      - suppose we had only one memory
  - data hazards
    - an instruction depends on a previous instruction
      - add \$s0, \$t0, \$t1
      - sub \$t2, \$s0, \$t3
  - control hazards
    - Need to make a decision based on results of one instruction
      - need to worry about branch instructions
- We'll build a simple pipeline and look at these issues

©2004 Morgan Kaufmann Publishers 8

## Pipelining

- Control hazard
  - Stall

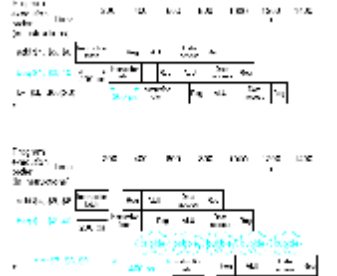


- Cost is too high

©2004 Morgan Kaufmann Publishers 9

## Pipelining

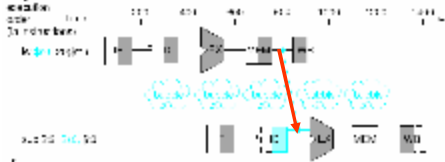
- Control hazard
  - Prediction – assume branch is not taken



©2004 Morgan Kaufmann Publishers 10

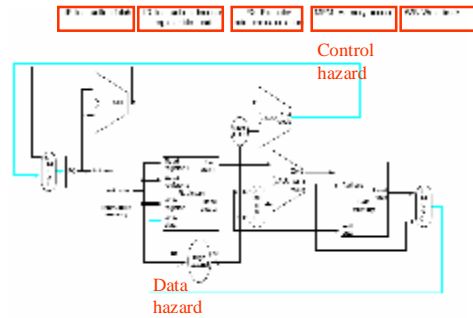
## Pipelining

- Data hazard
  - Forwarding
    - e.g. R-type instructions follow load
      - Need stall

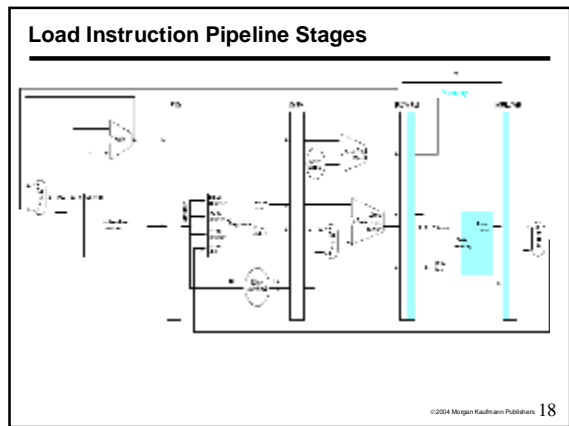
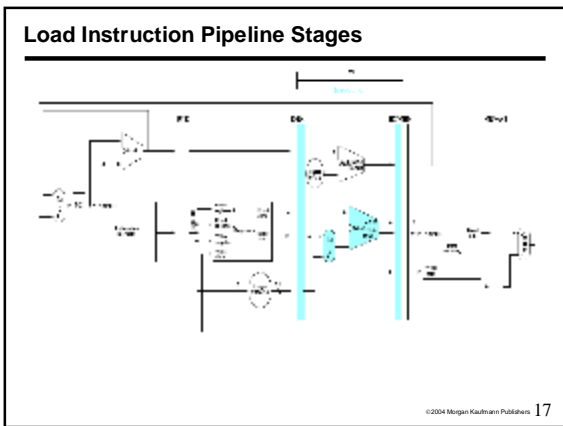
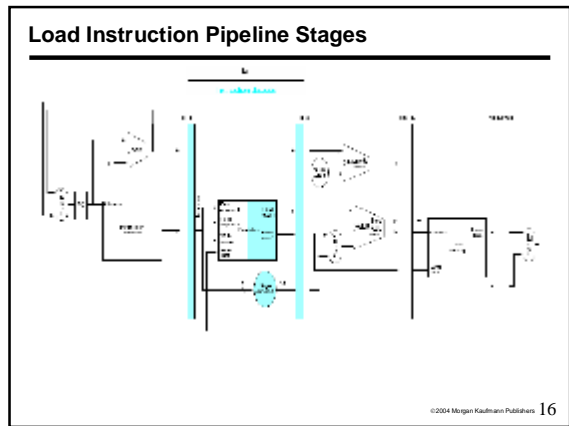
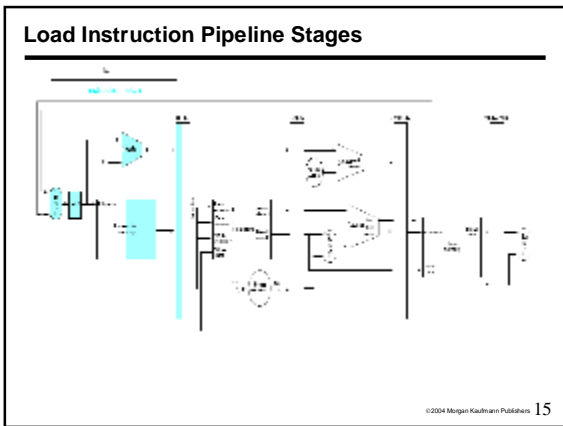
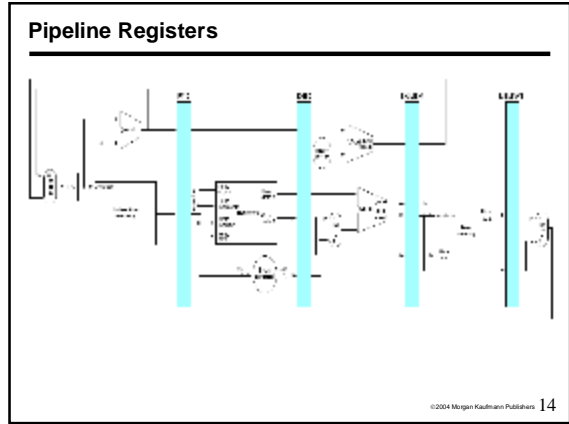
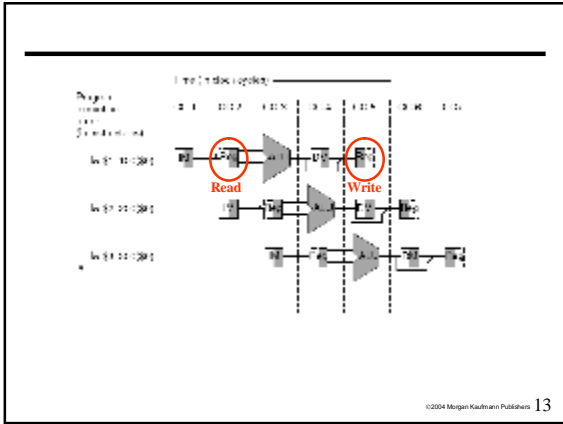


©2004 Morgan Kaufmann Publishers 11

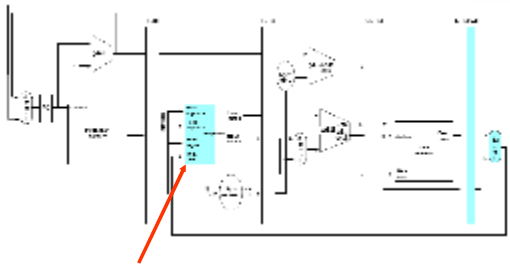
## Five-stage pipeline



©2004 Morgan Kaufmann Publishers 12

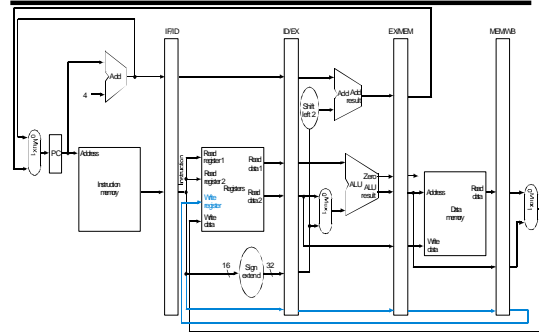


### Load Instruction Pipeline Stages

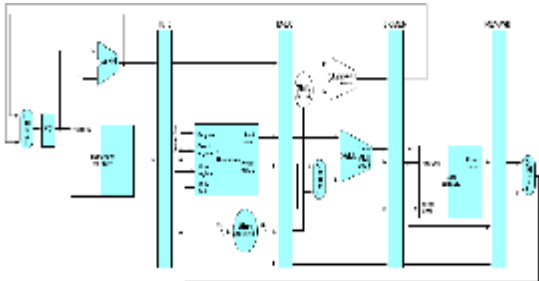


How does computer know which register to write?

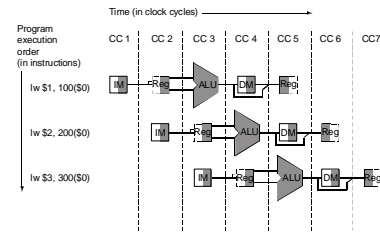
### Corrected Datapath



### All Five Stages for Load Instruction

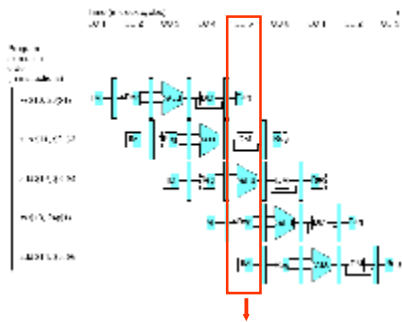


### Graphically Representing Pipelines

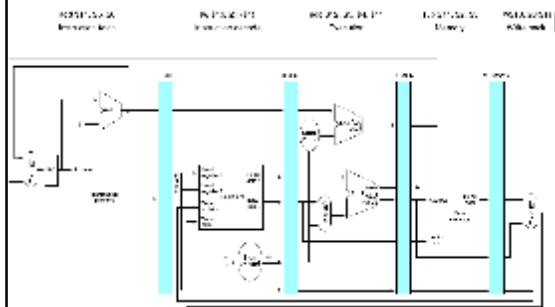


- how many cycles does it take to execute this code?
- what is the ALU doing during cycle 4?

### Multi-Clock-Cycle Pipeline for Five Instruction Sequence



### Multi-Clock-Cycle Pipeline for Five Instruction Sequence



Clock Cycle 5

## Pipeline Control

©2004 Morgan Kaufmann Publishers 25

## Pipeline control

- We have 5 stages. What needs to be controlled in each stage?
  - Instruction Fetch and PC Increment
  - Instruction Decode / Register File Read
  - Execution
  - Memory Stage
  - Write Back

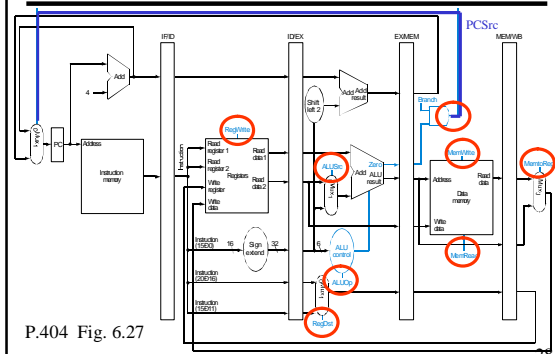
©2004 Morgan Kaufmann Publishers 26

## Control Lines

- Borrow as much as we can from the existing data path
- Five groups
  - Instruction fetch – happens every stage, no optional control lines
    - Read instruction from IM
    - PC asserted
  - Instruction decode/register file read – happens every stages, no optional control lines
    - RegDst, ALUOp, and ALUSrc
  - Memory access
    - Branch, MemRead, and MemWrite
    - Signals set by branch equal, load, and store instructions
  - Write back
    - MemtoReg (ALU result or memory value) and RegWrite

©2004 Morgan Kaufmann Publishers 27

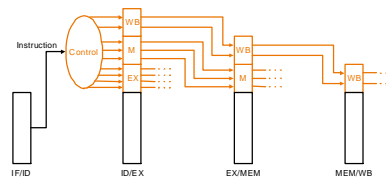
## Pipeline Control



## Pipeline Control

- Pass control signals along just like the data

Instruction	Execution/Address Calculation stage control lines				Memory access stage control lines			stage control lines	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg write	Mem to Reg
R-format	1	1	0	0	0	0	0	1	0
lwr	0	0	0	1	0	1	0	1	1
swr	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

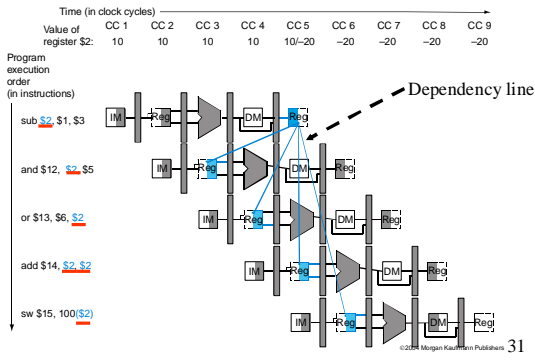


©2004 Morgan Kaufmann Publishers 29

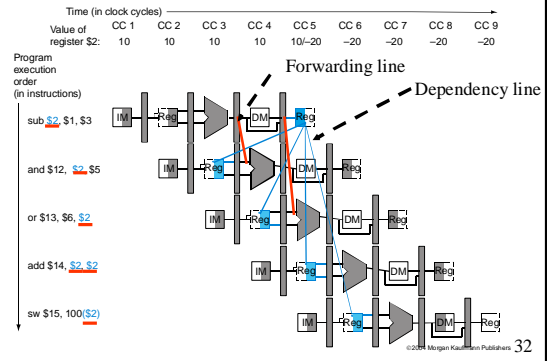
## Dependency and Forwarding

©2004 Morgan Kaufmann Publishers 30

## Dependencies – Data Hazards

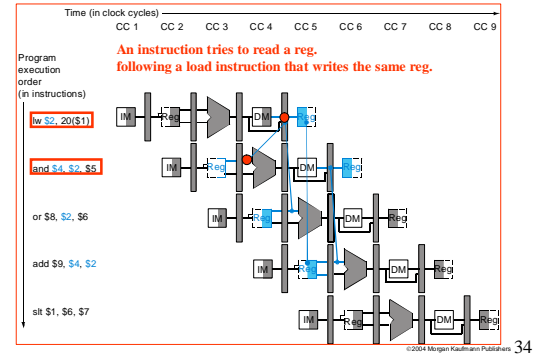


## Forwarding – consider EX stage only



Can't always forward!

## Can't always forward – Load word

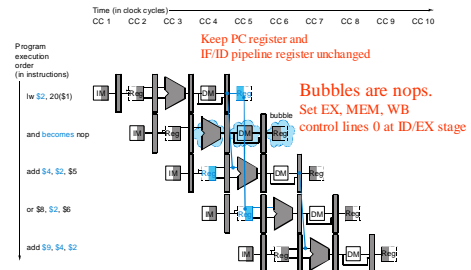


## Stalling

- We can stall the pipeline by keeping an instruction in the same stage
  - Nop**
    - An instruction that does no operation to change state
    - All control signals are set to 0
- Prevent instruction from making progress
  - Prevent PC register and pipeline register from changing

## Stalling

- We can stall the pipeline by keeping an instruction in the same stage

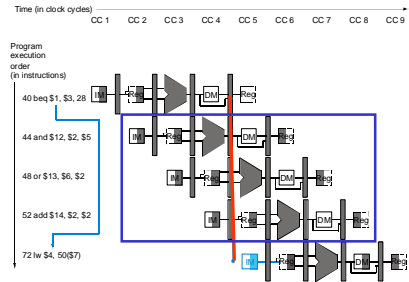


## Branch Hazard

©2004 Morgan Kaufmann Publishers 37

## Branch Hazards

- When we decide to branch, other instructions are in the pipeline!



- We are predicting "branch not taken"
  - need to add hardware for flushing instructions if we are wrong

©2004 Morgan Kaufmann Publishers 38

## Branch Prediction

- Sophisticated Techniques:
  - A "branch target buffer" to help us look up the destination
  - Correlating predictors that base prediction on global behavior and recently executed branches (e.g., prediction for a specific branch instruction based on what happened in previous branches)
  - Tournament predictors that use different types of prediction strategies and keep track of which one is performing best.
  - A "branch delay slot" which the compiler tries to fill with a useful instruction (make the one cycle delay part of the ISA)
- Branch prediction is especially important because it enables other more advanced pipelining techniques to be effective!
- Modern processors predict correctly 95% of the time!

©2004 Morgan Kaufmann Publishers 39

## Instruction Level Parallelism

- Pipelining
  - Increase depth of pipeline to overlap more instructions
- Multiple issue
  - Replicate internal components
  - Issue multiple instruction per stage
    - Allow instruction execution rate to exceed clock rate
- SuperScalar and VLIW

©2004 Morgan Kaufmann Publishers 40