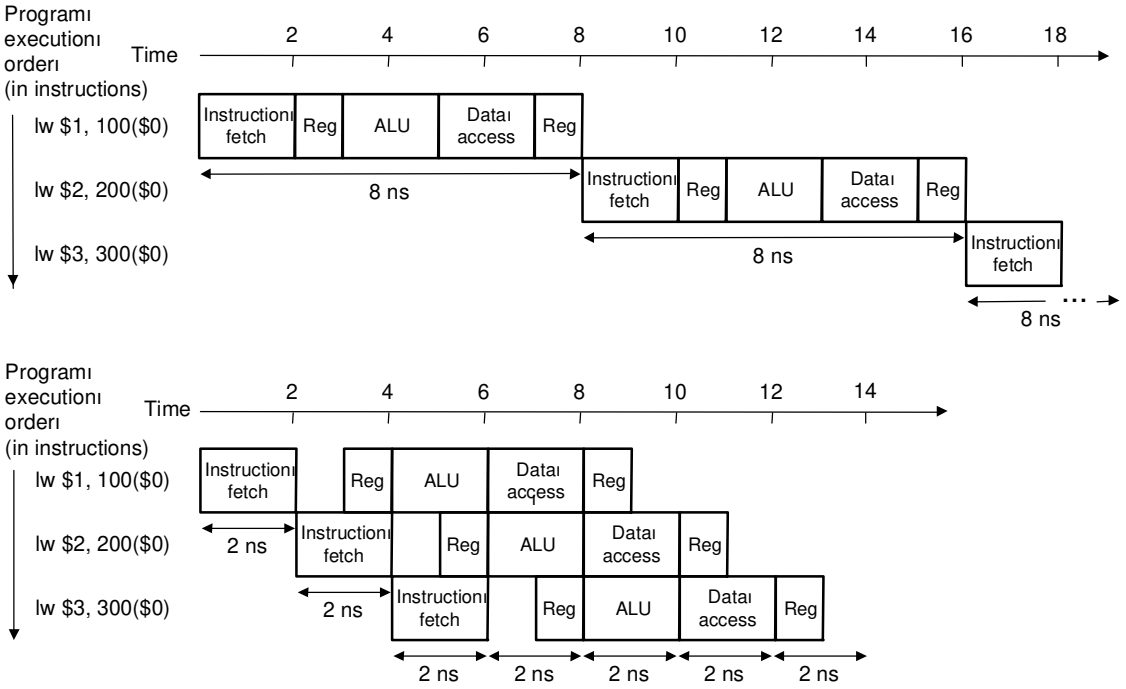
Chapter Six

©1998 Morgan Kaufmann Publishers

Pipelining

- **Improve perfomance by increasing instruction throughput**



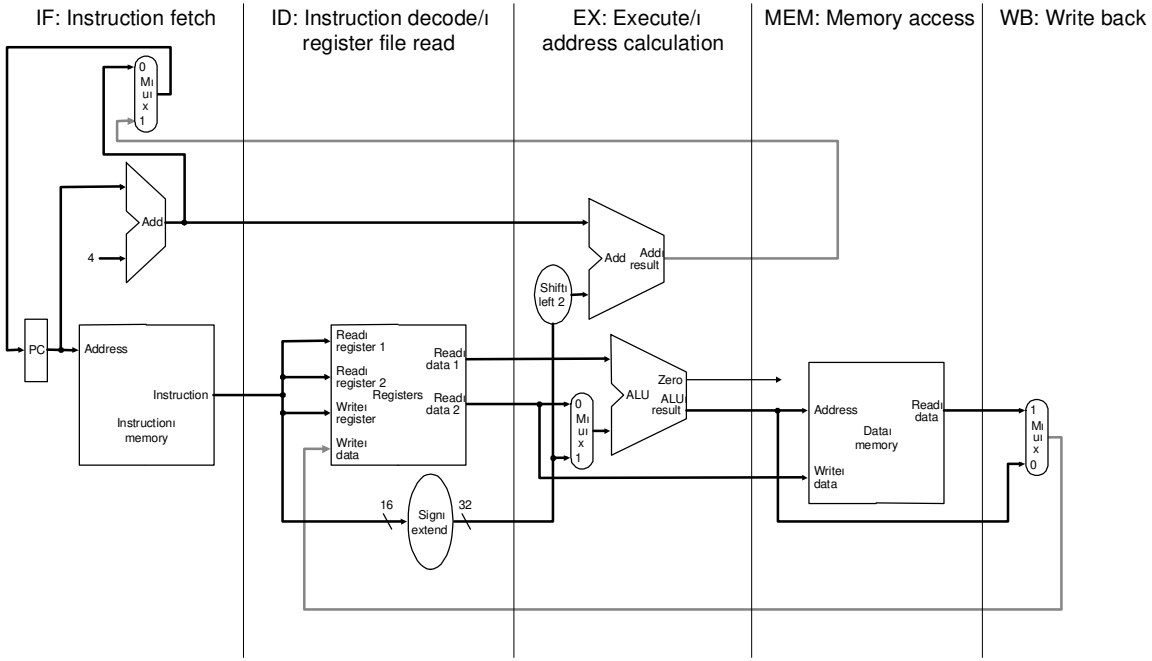
Ideal speedup is number of stages in the pipeline. Do we achieve this?

Pipelining

- What makes it easy
  - all instructions are the same length
  - just a few instruction formats
  - memory operands appear only in loads and stores
- What makes it hard?
  - structural hazards: suppose we had only one memory
  - control hazards: need to worry about branch instructions
  - data hazards: an instruction depends on a previous instruction
- We'll build a simple pipeline and look at these issues
- We'll talk about modern processors and what really makes it hard:
  - exception handling
  - trying to improve performance with out-of-order execution, etc.

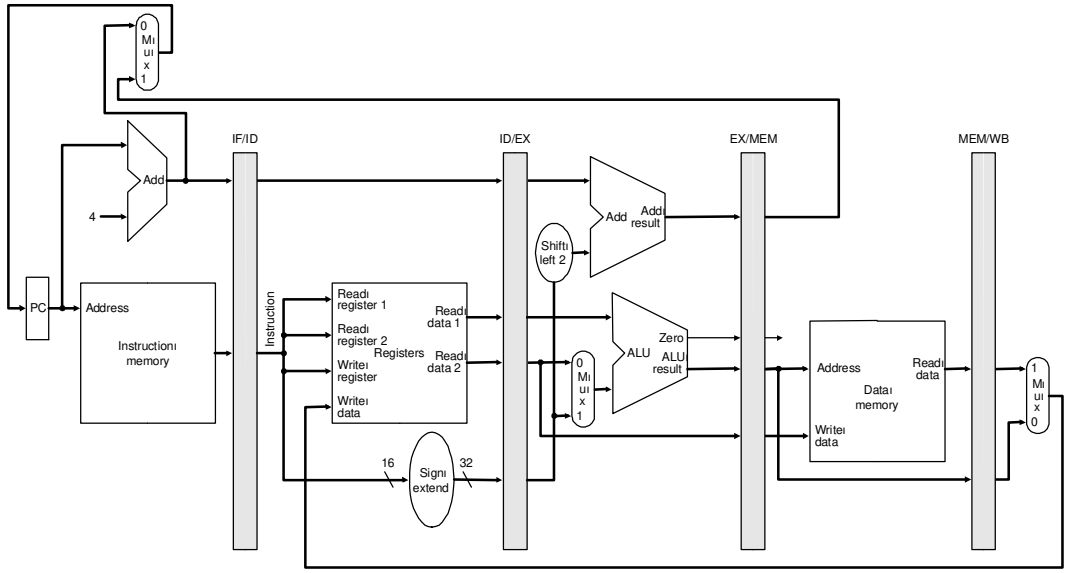
©1998 Morgan Kaufmann Publishers

Basic Idea



- ***What do we need to add to actually split the datapath into stages?***

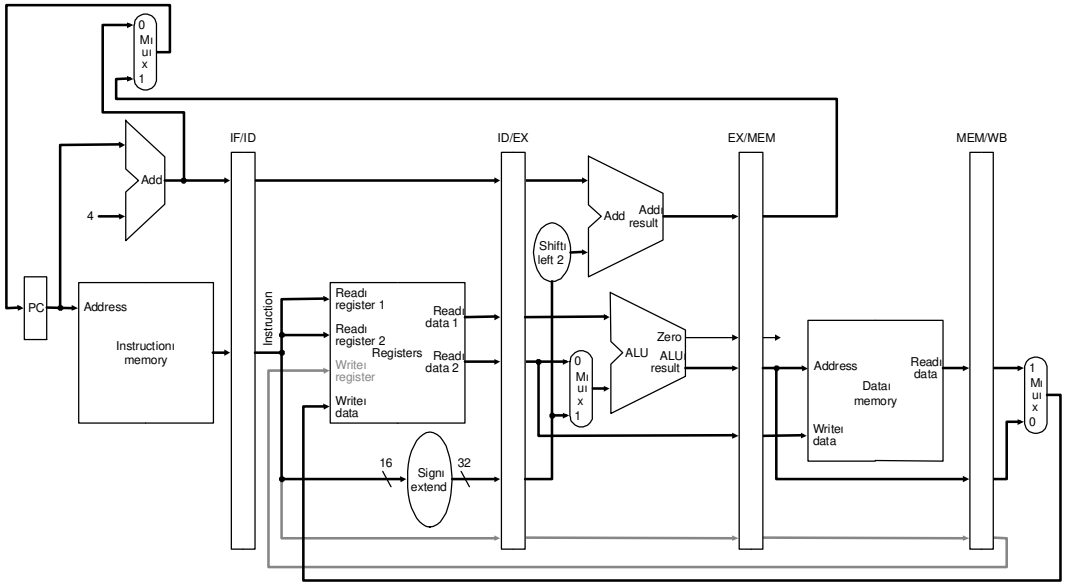
Pipelined Datapath



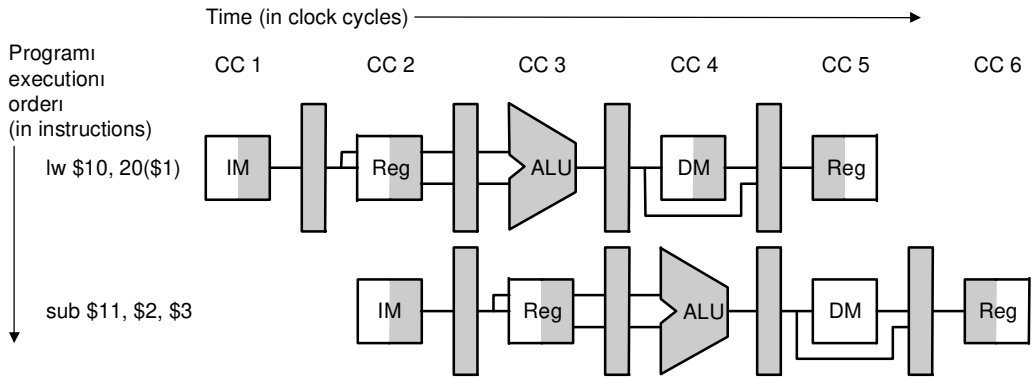
Can you find a problem even if there are no dependencies? What instructions can we execute to manifest the problem?

©1998 Morgan Kaufmann Publishers

Corrected Datapath



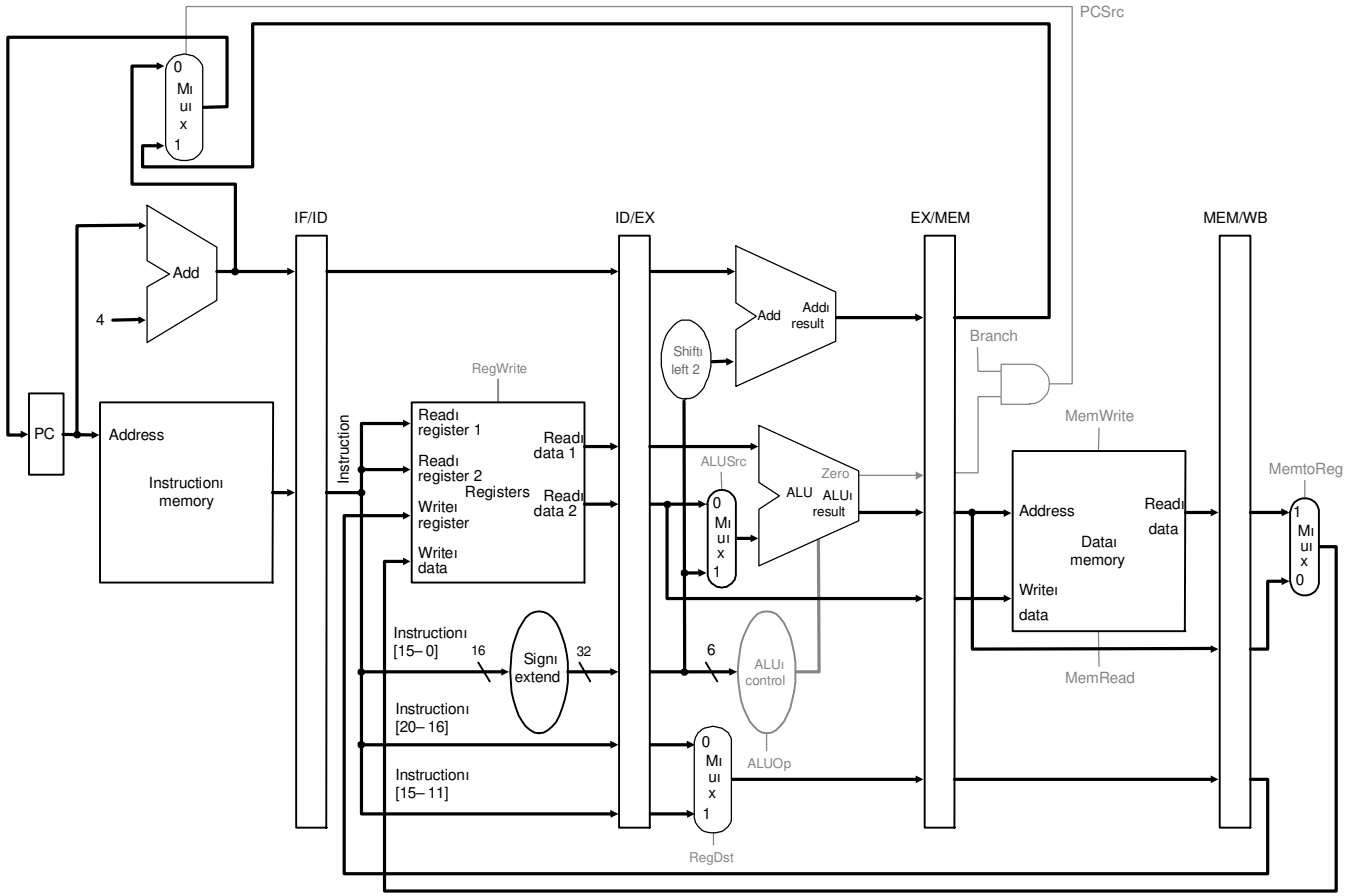
Graphically Representing Pipelines



- **Can help with answering questions like:**
  - **how many cycles does it take to execute this code?**
  - **what is the ALU doing during cycle 4?**
  - **use this representation to help understand datapaths**

©1998 Morgan Kaufmann Publishers

Pipeline Control



Pipeline control

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

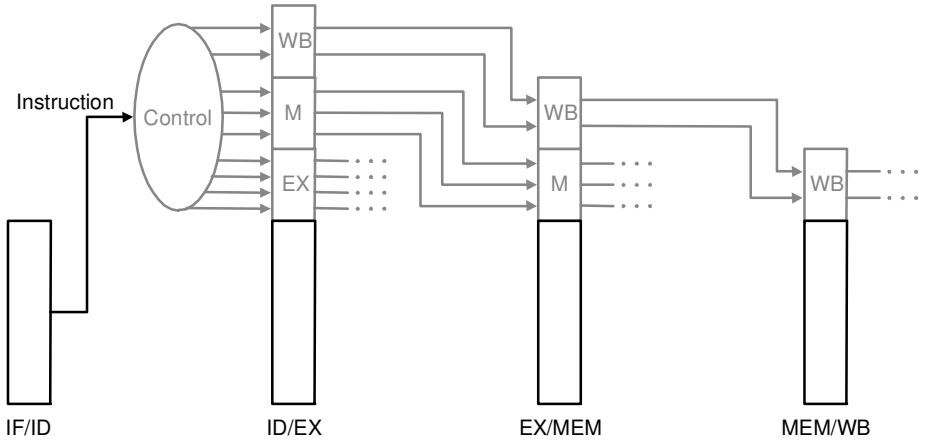
- How would control be handled in an automobile plant?
  - a fancy control center telling everyone what to do?
  - should we use a finite state machine?

©1998 Morgan Kaufmann Publishers

Pipeline Control

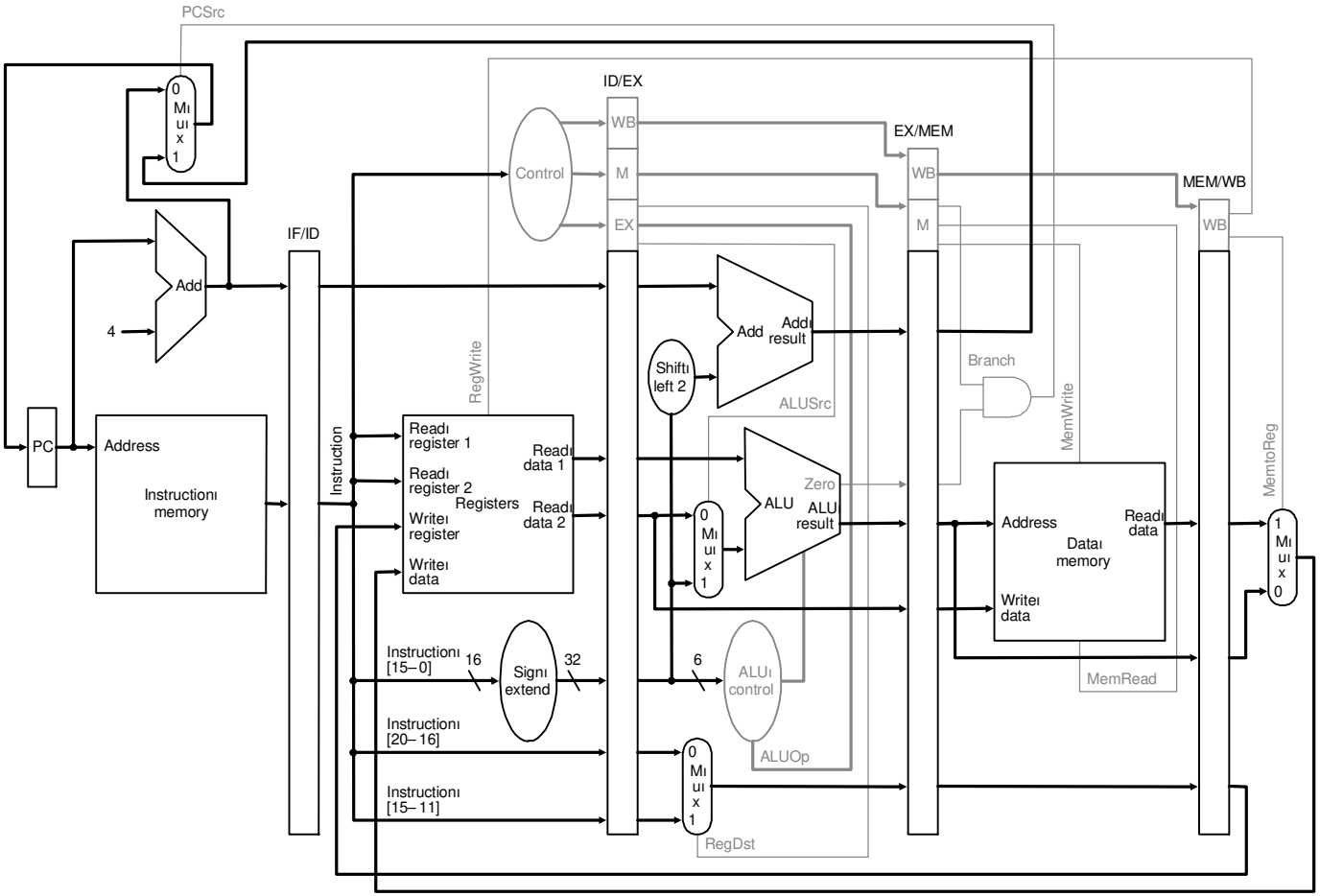
- **Pass control signals along just like the data**

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
`lw`	0	0	0	1	0	1	0	1	1
`sw`	X	0	0	1	0	0	1	0	X
`beq`	X	0	1	0	1	0	0	0	X



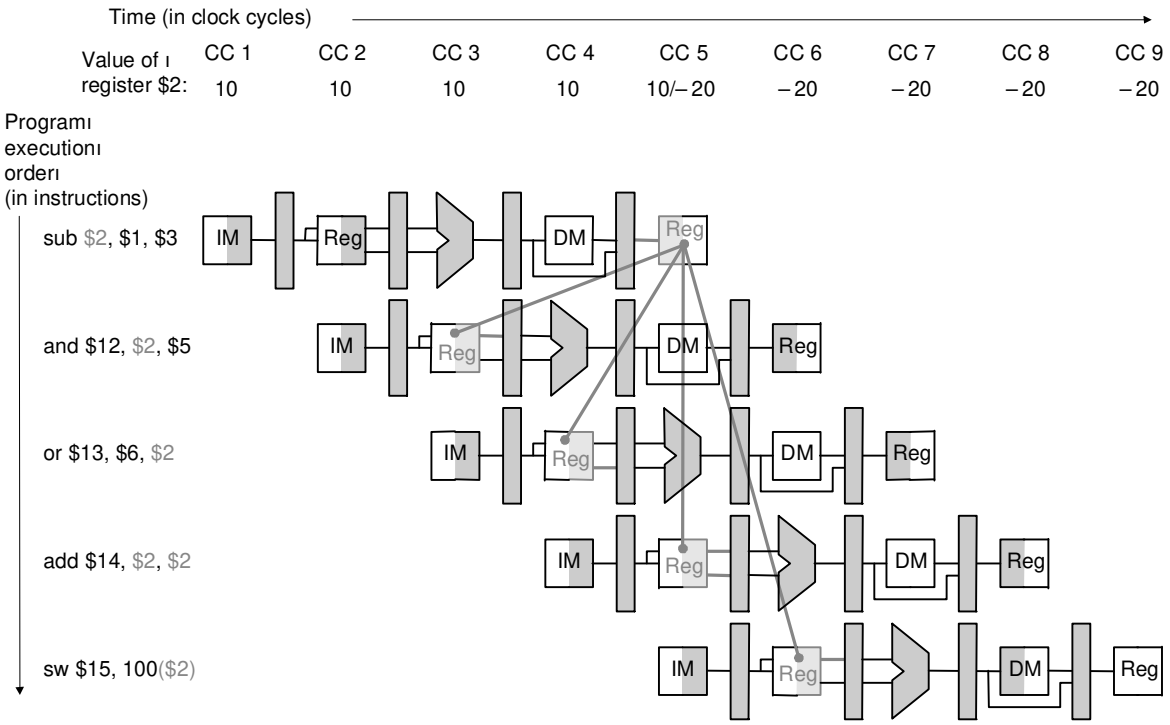
©1998 Morgan Kaufmann Publishers

Datapath with Control



Dependencies

- **Problem with starting next instruction before first is finished**
  - **dependencies that "go backward in time" are data hazards**



©1998 Morgan Kaufmann Publishers

Software Solution

- Have compiler guarantee no hazards
- Where do we insert the “nops” ?

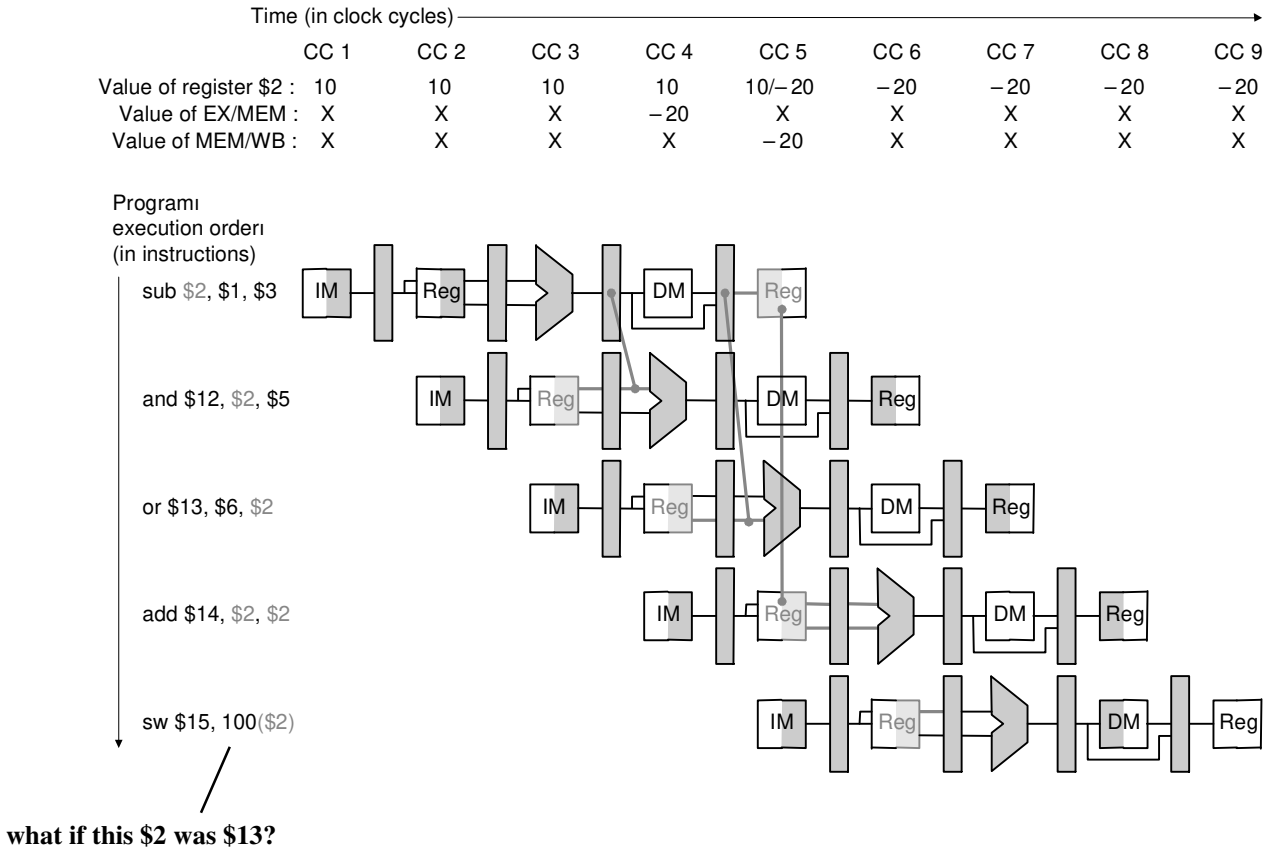
```
sub    $2, $1, $3
and    $12, $2, $5
or     $13, $6, $2
add    $14, $2, $2
sw     $15, 100($2)
```

- Problem: this really slows us down!

©1998 Morgan Kaufmann Publishers

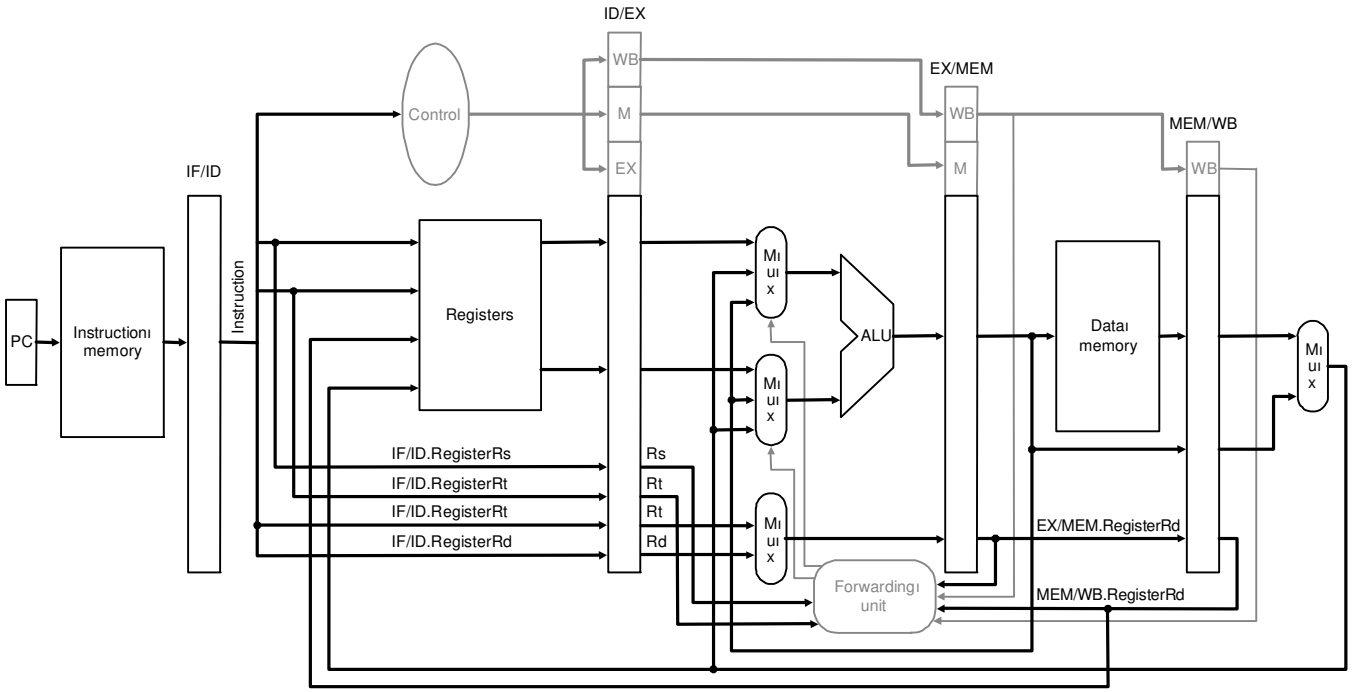
Forwarding

- **Use temporary results, don't wait for them to be written**
  - **register file forwarding to handle read/write to same register**
  - **ALU forwarding**



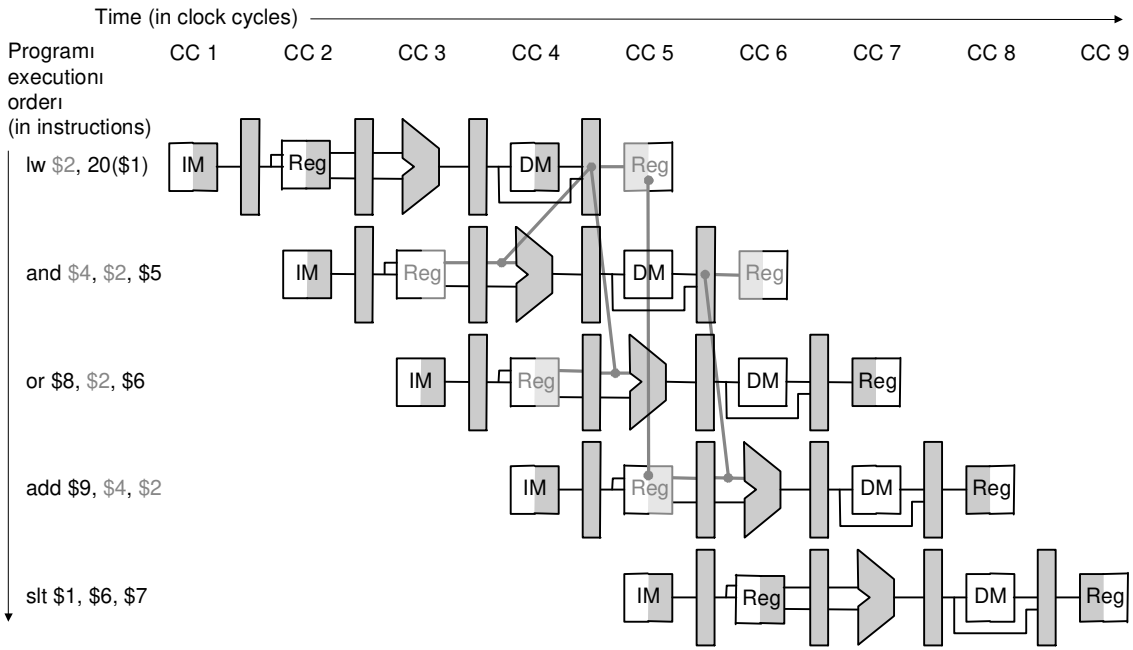
©1998 Morgan Kaufmann Publishers

Forwarding


ID/EX
EX/MEM
MEM/WB
WB
M
EX
Control
IF/ID
Instruction
PC
Instruction memory
Registers
ALU
Data memory
Mux
IF/ID.RegisterRs
IF/ID.RegisterRt
IF/ID.RegisterRt
IF/ID.RegisterRd
Rs
Rt
Rt
Rd
EX/MEM.RegisterRd
MEM/WB.RegisterRd
Forwarding unit

Can't always forward

- **Load word can still cause a hazard:**
  - **an instruction tries to read a register following a load instruction that writes to the same register.**

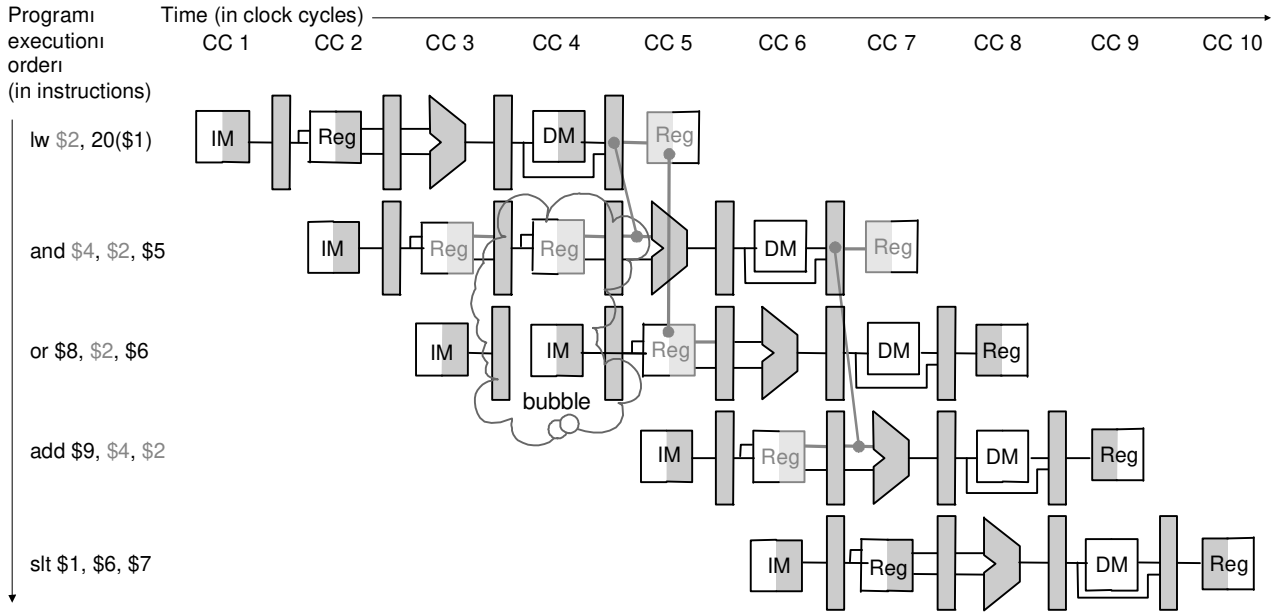


- **Thus, we need a hazard detection unit to "stall" the load instruction**

©1998 Morgan Kaufmann Publishers

Stalling

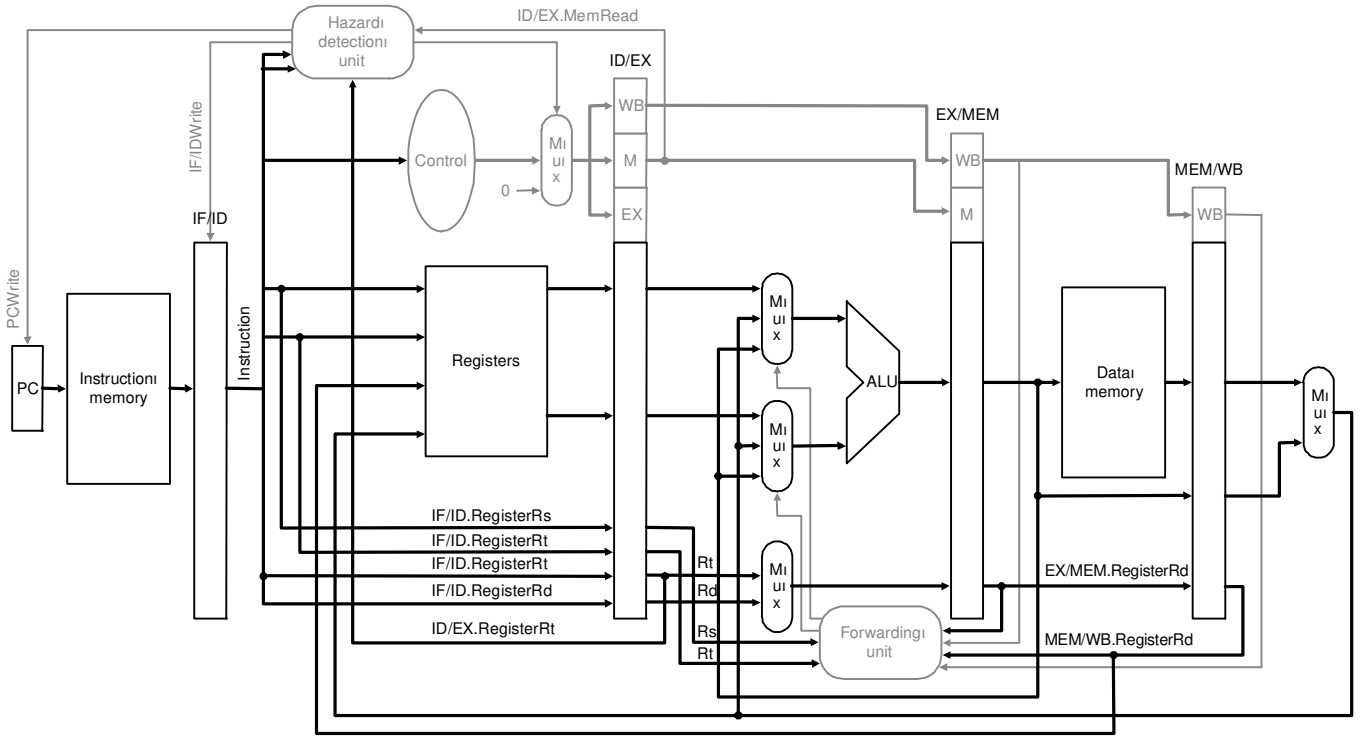
- **We can stall the pipeline by keeping an instruction in the same stage**



©1998 Morgan Kaufmann Publishers

Hazard Detection Unit

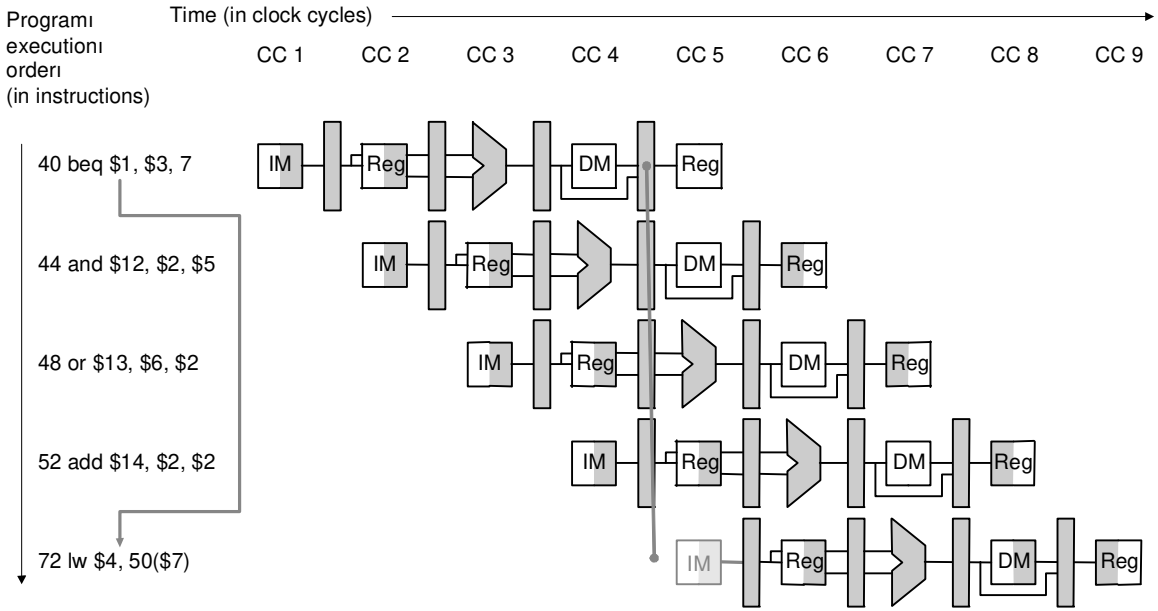
- **Stall by letting an instruction that won't write anything go forward**



©1998 Morgan Kaufmann Publishers

Branch Hazards

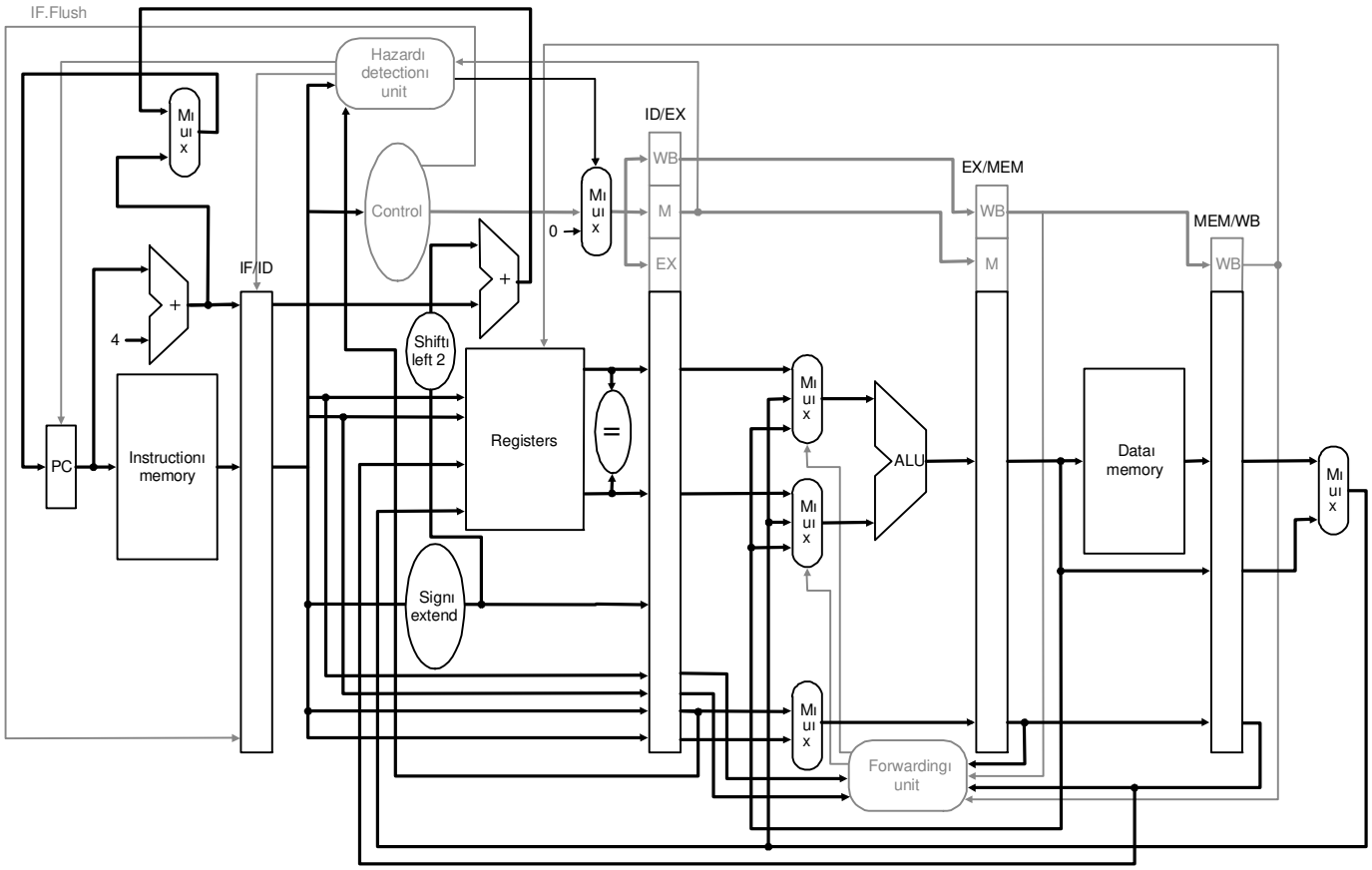
- **When we decide to branch, other instructions are in the pipeline!**



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

©1998 Morgan Kaufmann Publishers

Flushing Instructions



Improving Performance

- **Try and avoid stalls! E.g., reorder these instructions:**

```
lw $t0, 0($t1)
lw $t2, 4($t1)
sw $t2, 0($t1)
sw $t0, 4($t1)
```

- **Add a "branch delay slot"**
  - **the next instruction after a branch is always executed**
  - **rely on compiler to "fill" the slot with something useful**

- **Superscalar: start more than one instruction in the same cycle**

©1998 Morgan Kaufmann Publishers

Dynamic Scheduling

- The hardware performs the "scheduling"
  - hardware tries to find instructions to execute
  - out of order execution is possible
  - speculative execution and dynamic branch prediction
- All modern processors are very complicated
  - DEC Alpha 21264: 9 stage pipeline, 6 instruction issue
  - PowerPC and Pentium: branch history table
  - Compiler technology important

- This class has given you the background you need to learn more

- Video: An Overview of Intel's Pentium Processor

  (available from University Video Communications)

©1998 Morgan Kaufmann Publishers