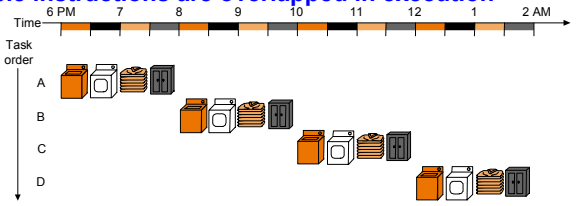
Chapter Six

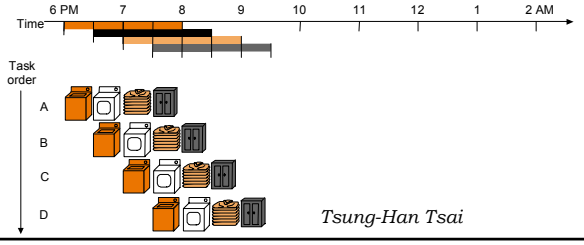
Pipelining

- **Improve performance by increasing instruction throughput**
  - **Multiple instructions are overlapped in execution**

Multicycle version

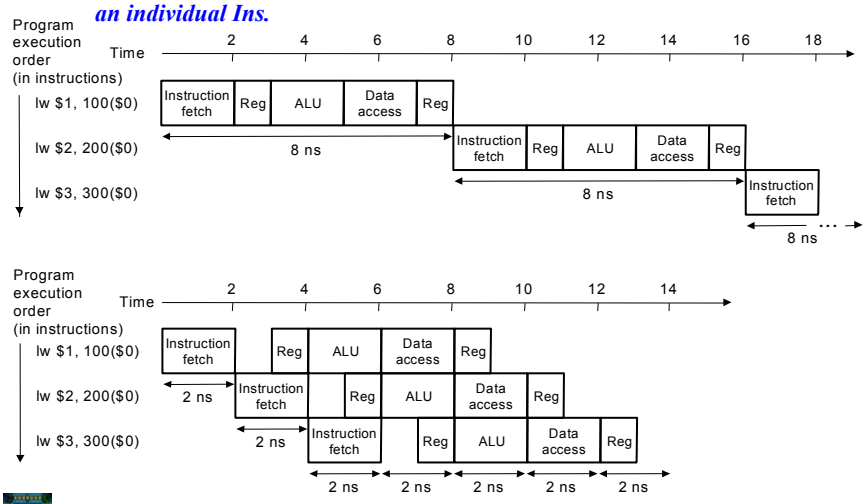


Pipeline Version



Pipelining

- *Ideal speedup is number of stages in the pipeline. Do we achieve this?*
  - *Example on Page 438. Pipelining improve throughput not execution time of an individual Ins.*

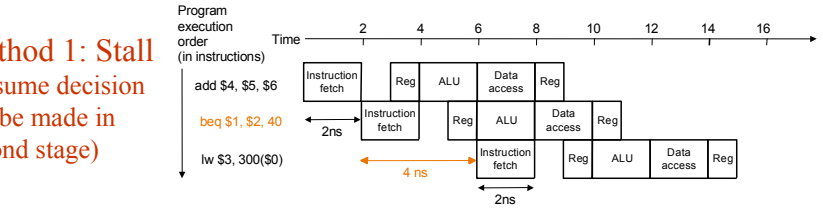


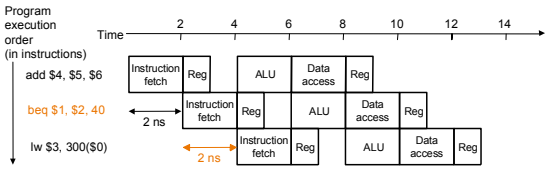
Pipelining

- **What makes it easy (like MIPS)**
  - **all instructions are the same length (some computers have Ins with different length(bits))**
  - **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.**

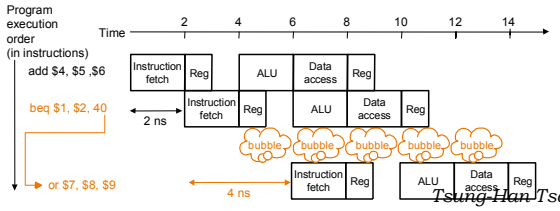
Control Hazards in Pipelining :Branch case

Method 1: Stall
(Assume decision can be made in second stage)



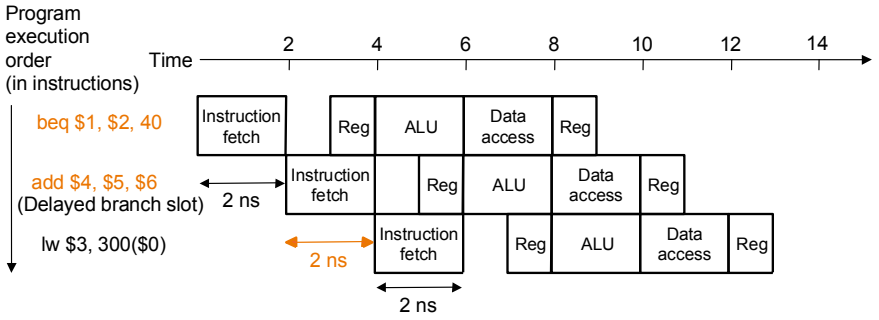


Method 2: Predicting



Control Hazards in Pipelining :Branch case

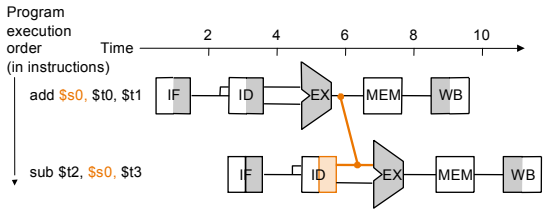
Method 3: Delayed decision (used in MIPS)



Data Hazards in Pipelining:Forwarding

Example: Add $s0, $t0, $t1

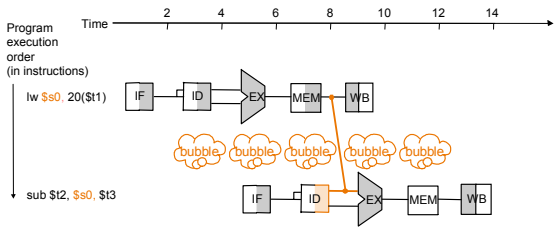
Sub $t2, $s0, $t3

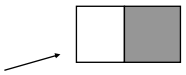


Example: lw $s0 20($t1)

sub $t2, $s0, $t3

Example on P.447

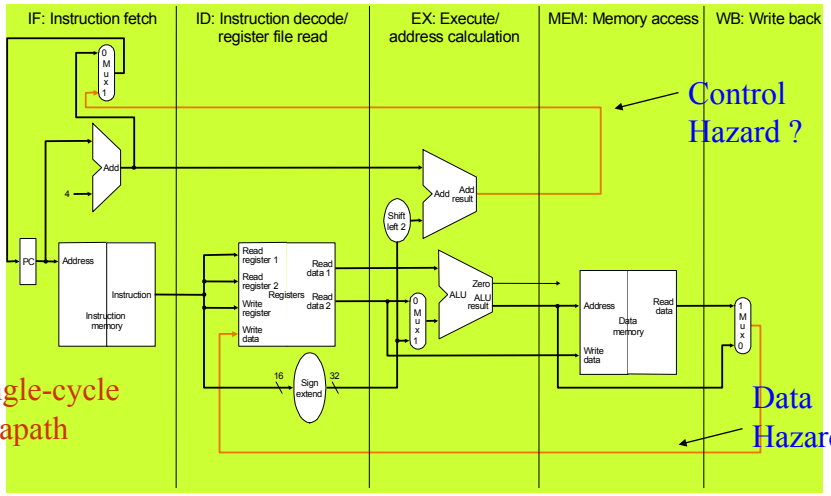




Write

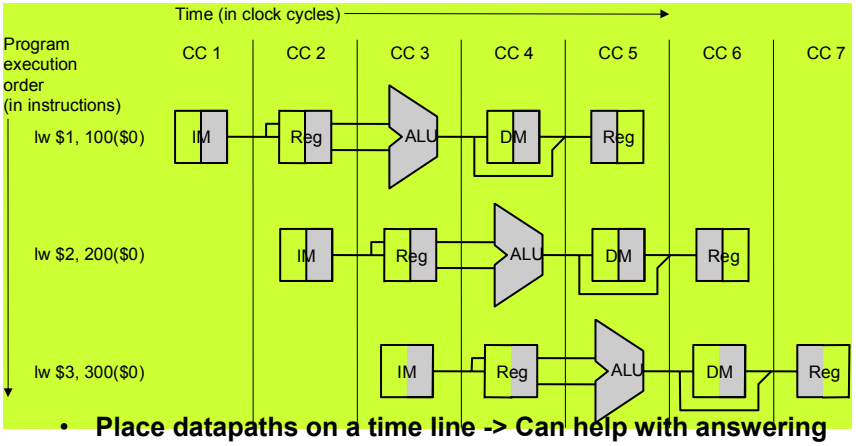
Read

6.2 A pipelined Datapath



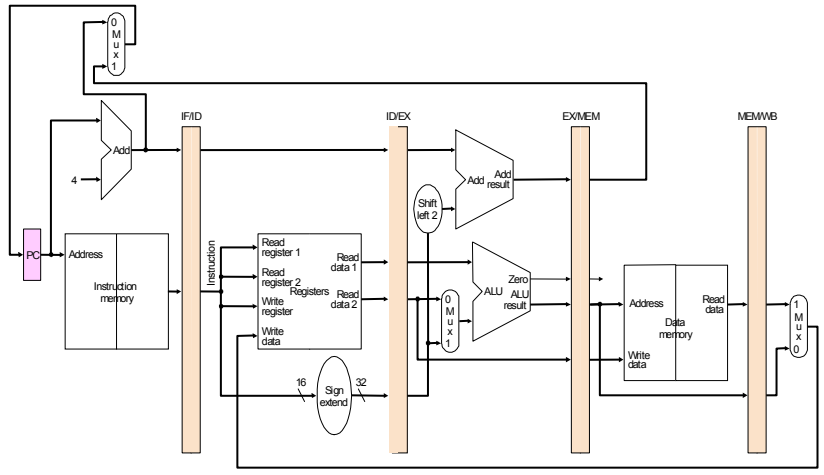
- ***Why do we need to add to actually split the datapath into stages?***

Graphically Representing Single-Cycle Instruction Execution



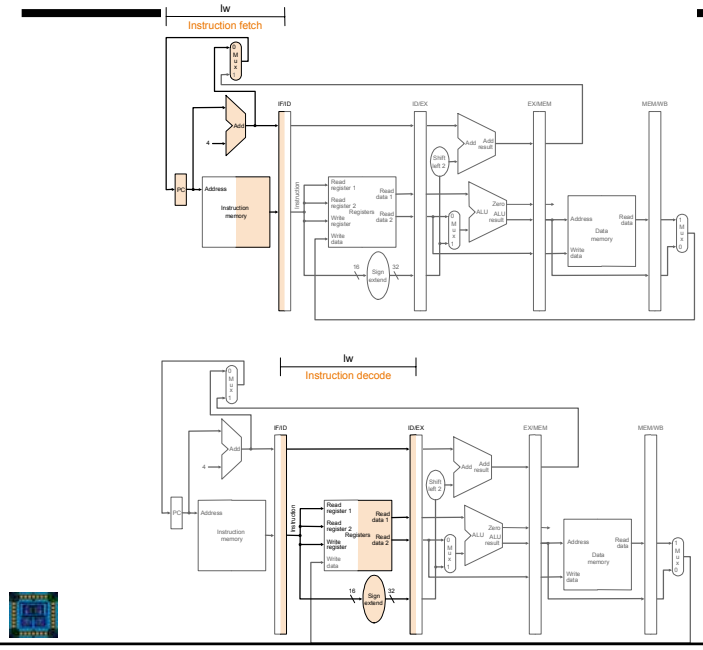
- **Place datapaths on a time line -> Can help with answering questions like:**
  - **How many cycles does it take to execute this instruction?**
  - **What is the ALU doing during cycle 4?**
  - **The relationship among different instructions that are executed**

Pipelined Datapath

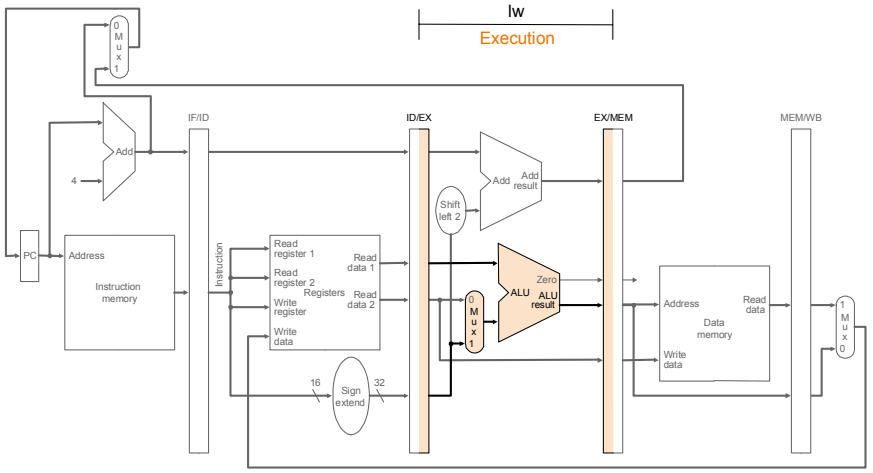


- **Pipeline registers are used to separate pipeline stage**
  - **Each bus with n lines (n bits) use n D-FF as the pipeline registers**
  - **All the registers are triggered by a same global clock**

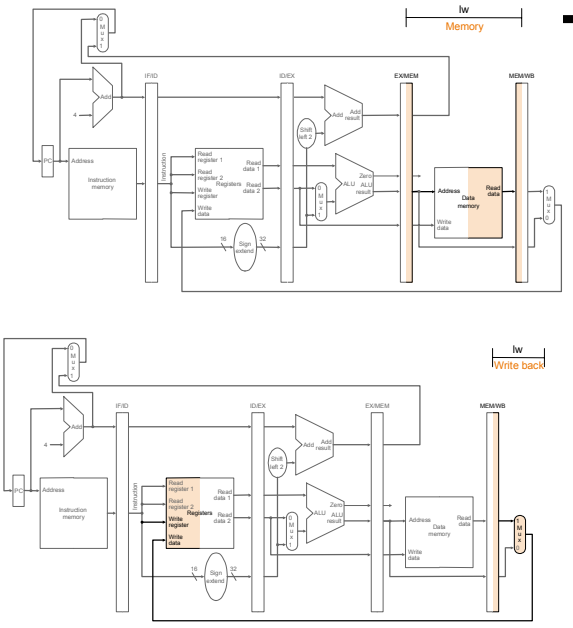
IF and ID of lw



EX of lw

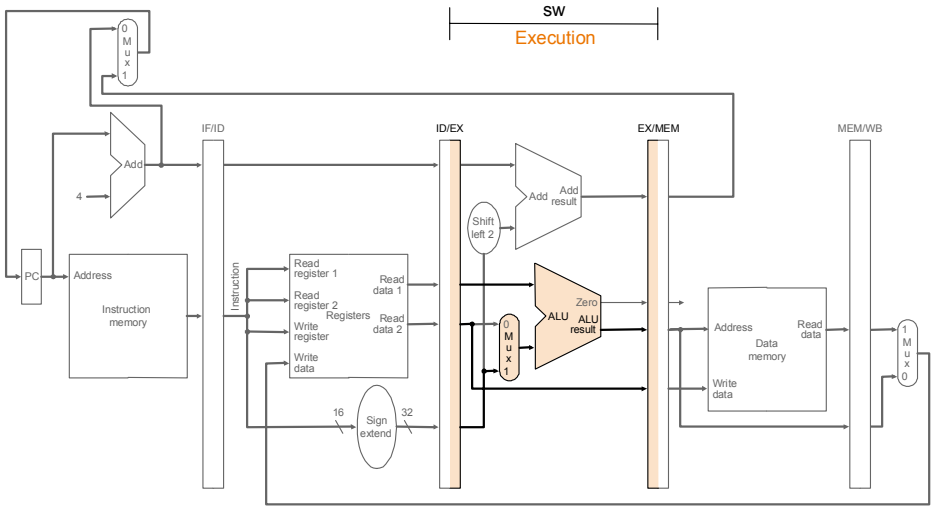


MEM and WB of lw

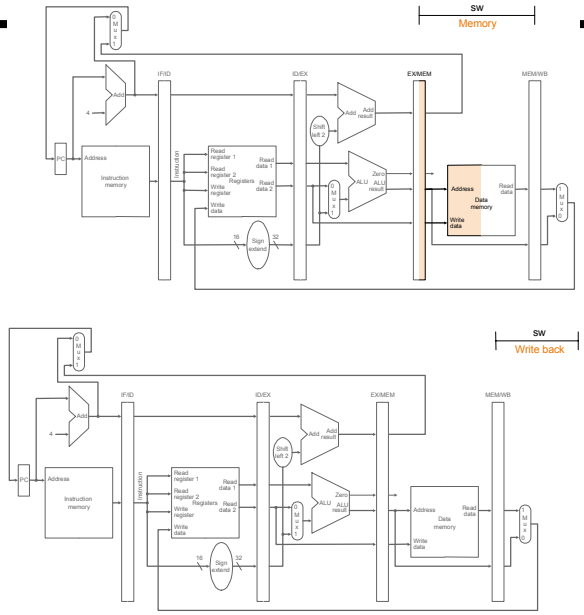


13

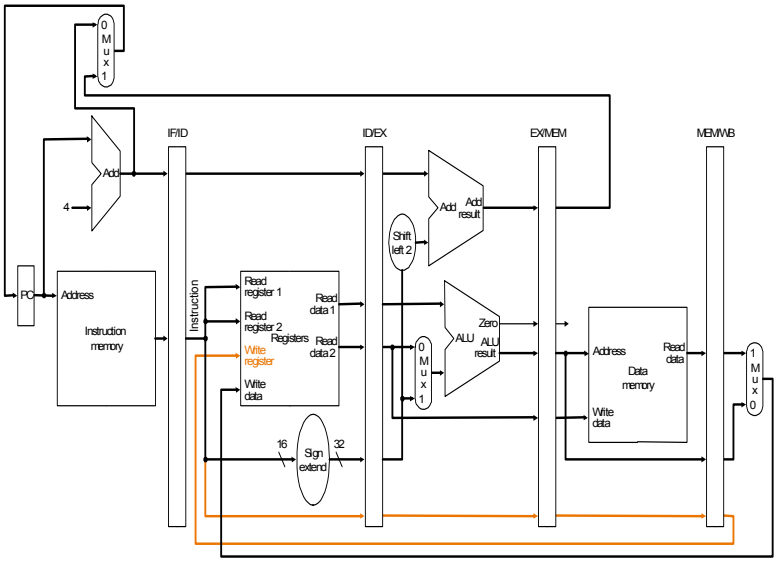
EX of sw



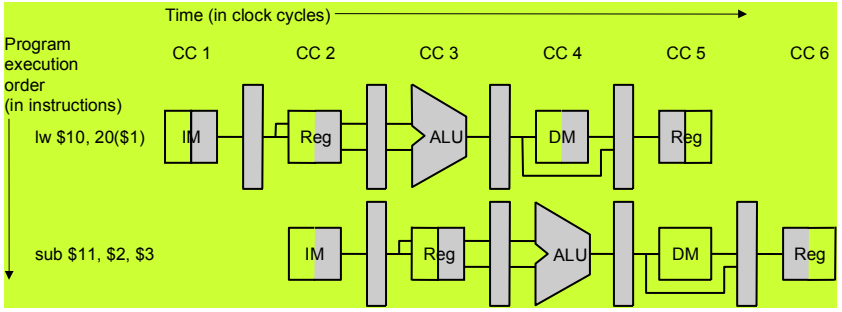
MEM and WB of sw

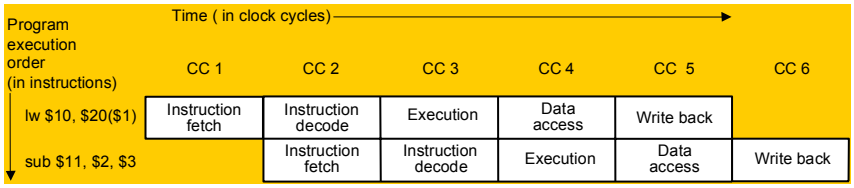


Corrected Pipelined Datapath



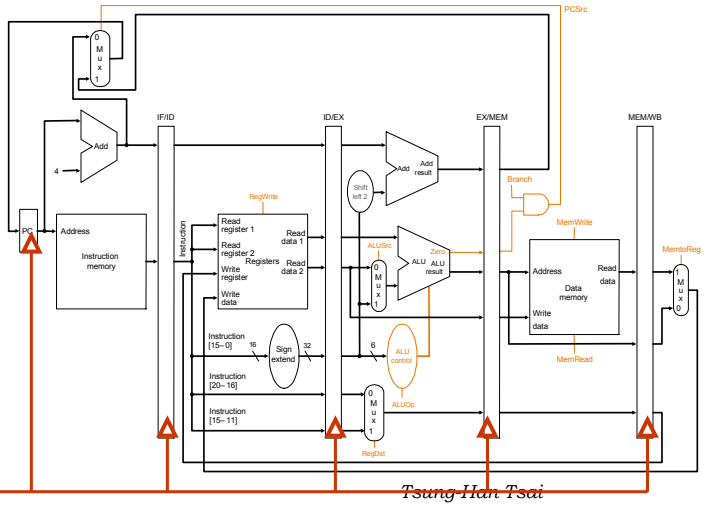
Multiple-Clock-Cycle Pipeline Diagram





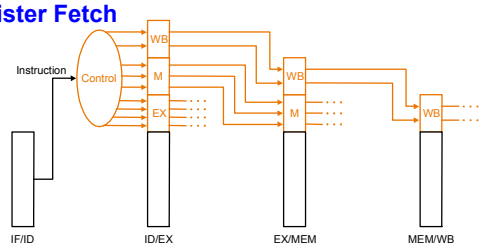
6.3 Pipeline Control

Add control to the pipelined datapath: Need to set the control values during each pipeline stage



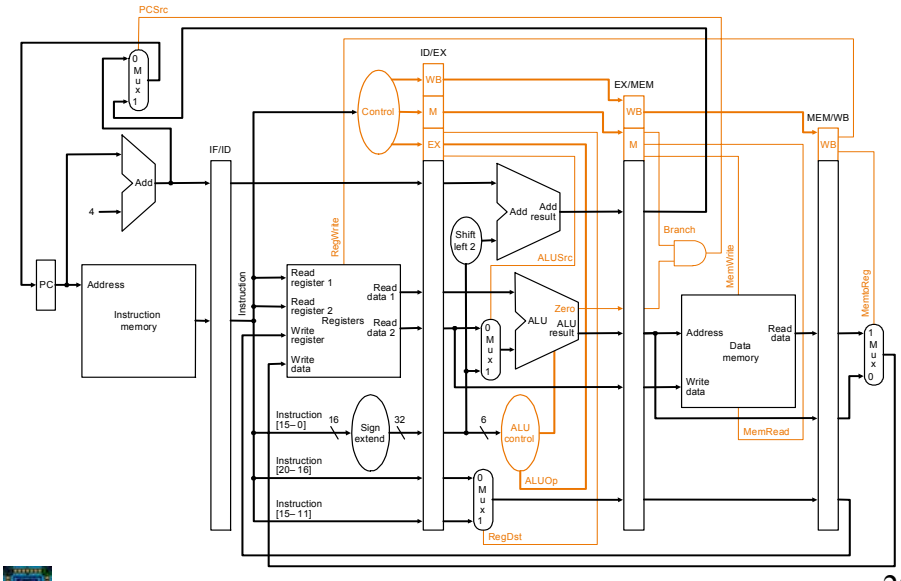
Pipeline Control

- Pass control signals along just like the data
- 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



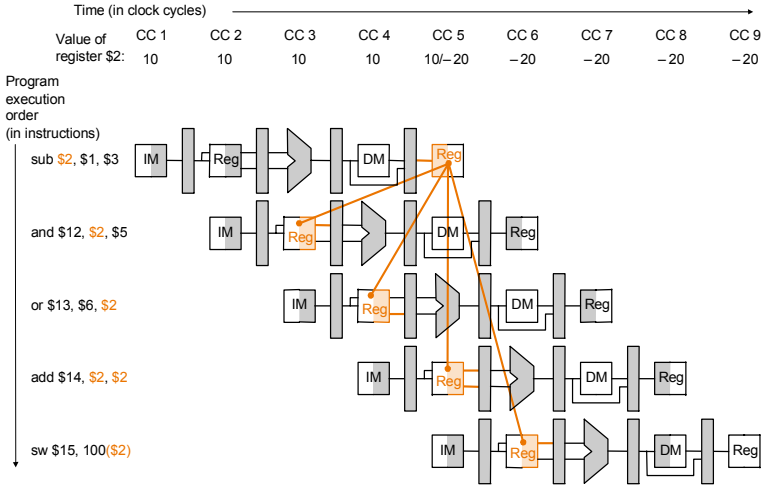
Instruction	Execution/Address Calculation stage control lines				Memory access stage control lines			stage control lines	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg write	Mem to Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

Datapath with Control: Fig.6.30



Dependencies

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



Software Solution

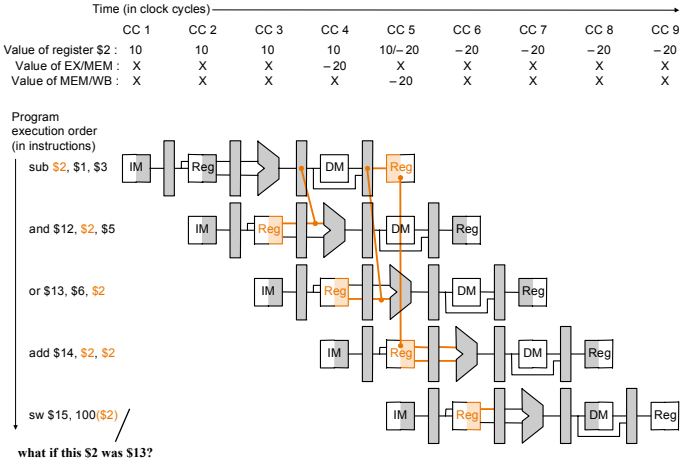
- **Have compiler guarantee no hazards**
- **Where do we insert the "nops" ?**

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

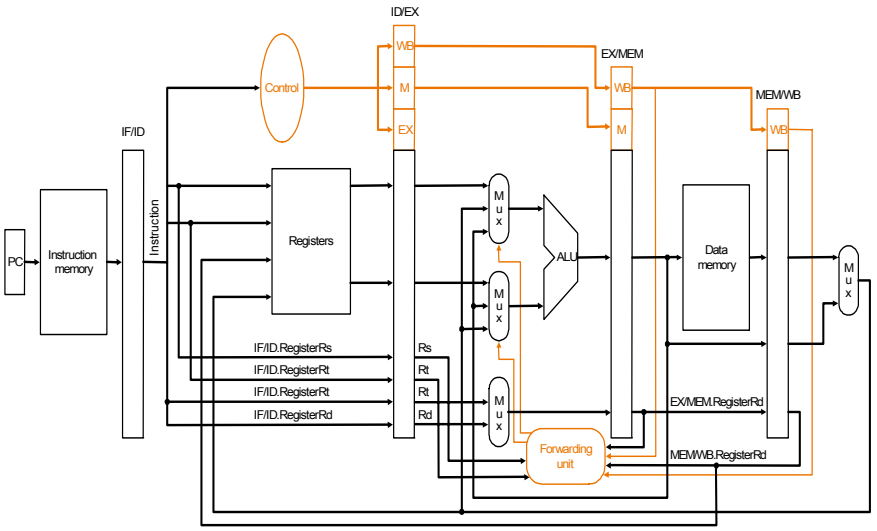
- **Problem: this really slows us down!**

Forwarding

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

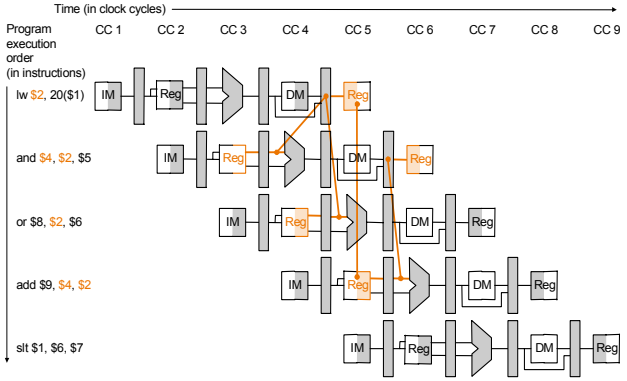


Forwarding



Can't always forward

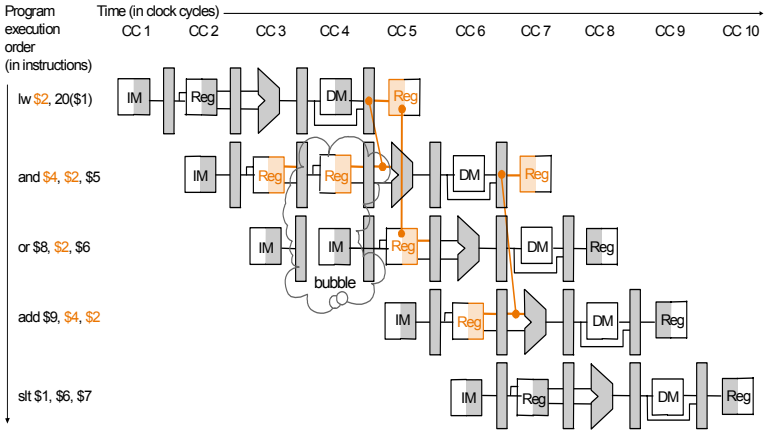
- **Load word can still cause a hazard:**
  - **an instruction tries to read a register following a load instruction that writes to the same register.**



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

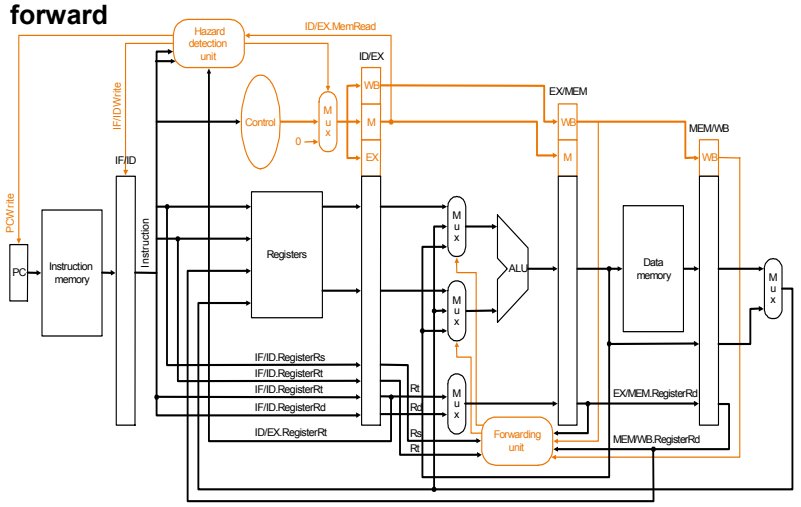
Stalling

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



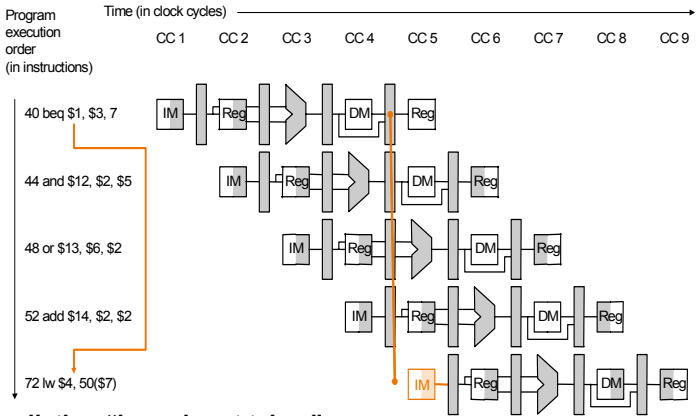
Hazard Detection Unit

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



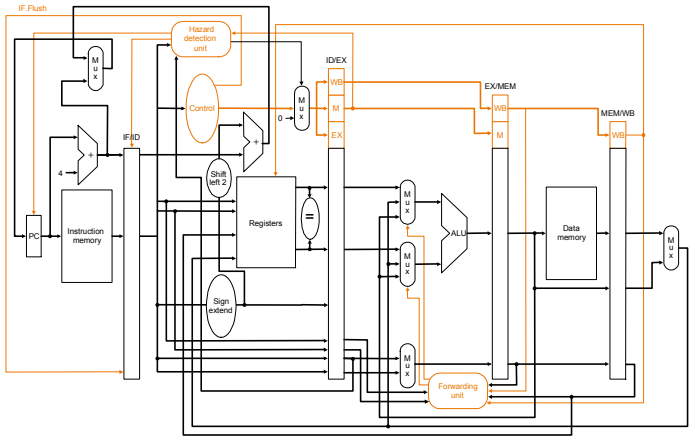
Branch Hazards

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



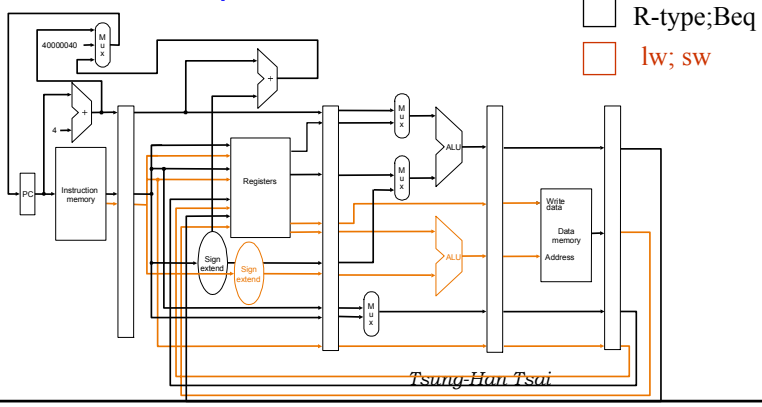
- **We are predicting "branch not taken"**
  - **need to add hardware for flushing instructions if we are wrong**

Flushing Instructions



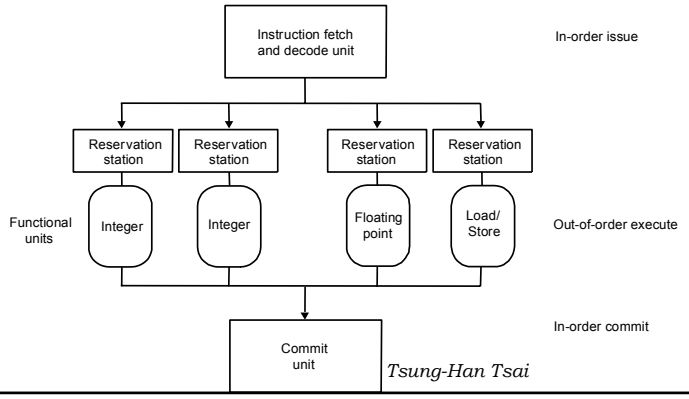
6.8 Improving Performance: Superscalar

- **Superscalar: to replicate the internal components of the computer so that it can launch multiple instructions in every pipeline stage -> decrease CPI or increase IPC (Instructions Per Cycle)**
  - **start more than one instruction in the same cycle; Can we ?**
  - **A 500 MHz four way superscalar CPU can execute a peak rate of two billion instructions per second**

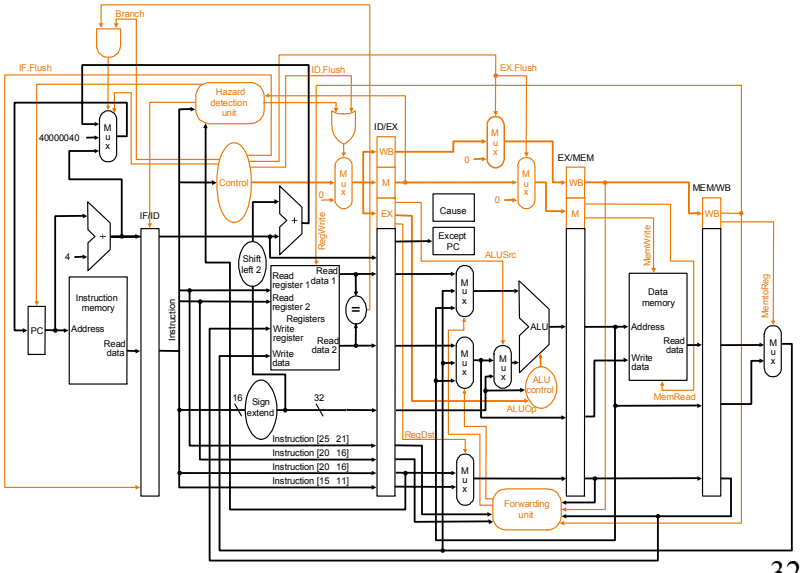


Dynamic Scheduling

- Dynamic pipeline scheduling: dynamic pipelining by the hardware to avoid pipeline hazards
- The hardware performs the "scheduling"
  - hardware tries to find instructions to execute
  - out of order execution is possible
  - speculative execution and dynamic branch prediction



The Final Datapath and Control



The Processor: Datapath & Control

- **All modern processors are very complicated**
  - **DEC Alpha 21264: 9 stage pipeline, 6 instruction issue**
  - **PowerPC and Pentium: branch history table**