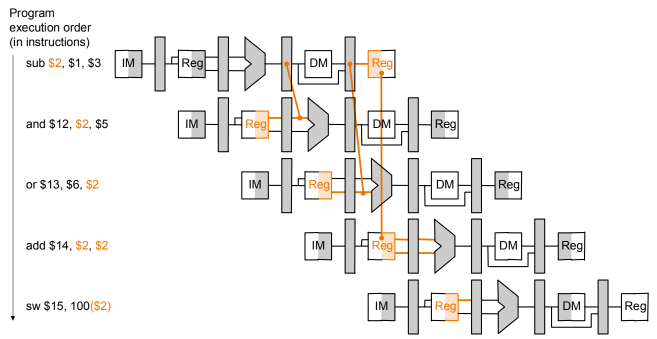
Review Forwarding

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

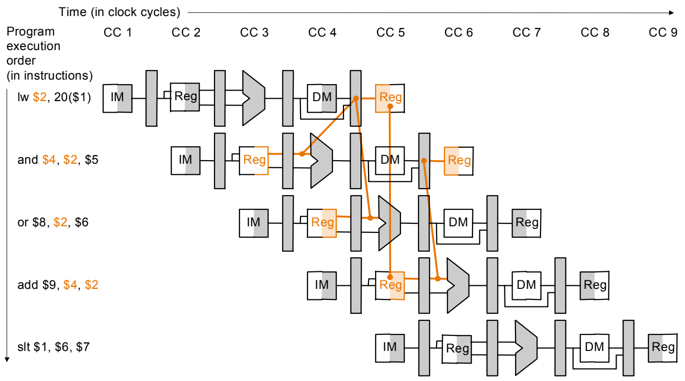
Time (in clock cycles) →

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2 :	10	10	10	10	10/– 20	– 20	– 20	– 20	– 20
Value of EX/MEM :	X	X	X	– 20	X	X	X	X	X
Value of MEM/WB :	X	X	X	X	– 20	X	X	X	X



Can't always forward

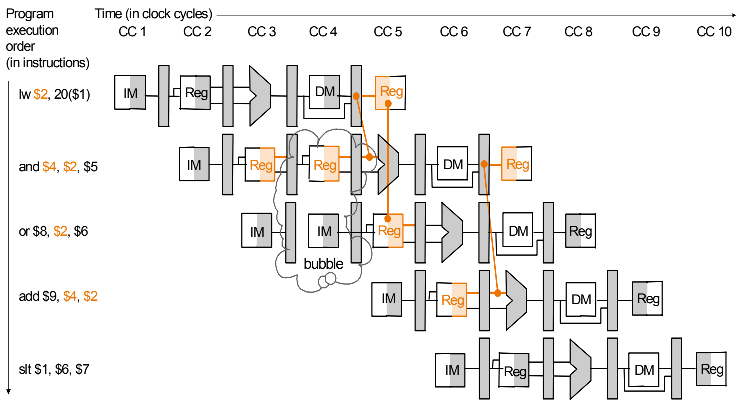
- Load word can still cause a hazard:
  - an instruction tries to read a register following a load instruction that writes to the same register.



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

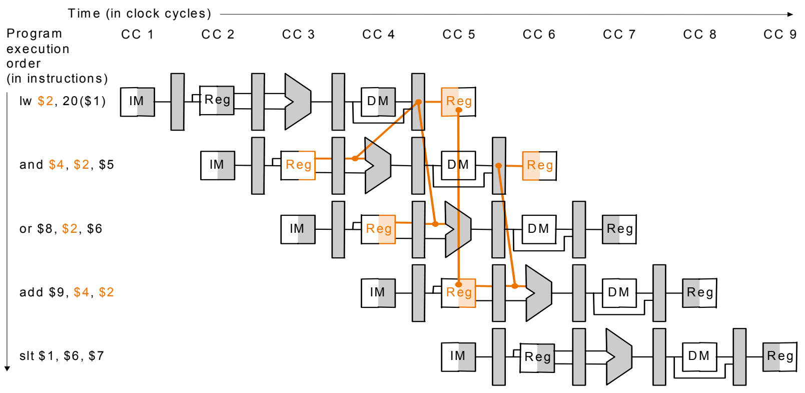
Stalling

- Hardware detection and no-op insertion is called stalling
- We stall the pipeline by keeping an instruction in the same stage



Can't always forward

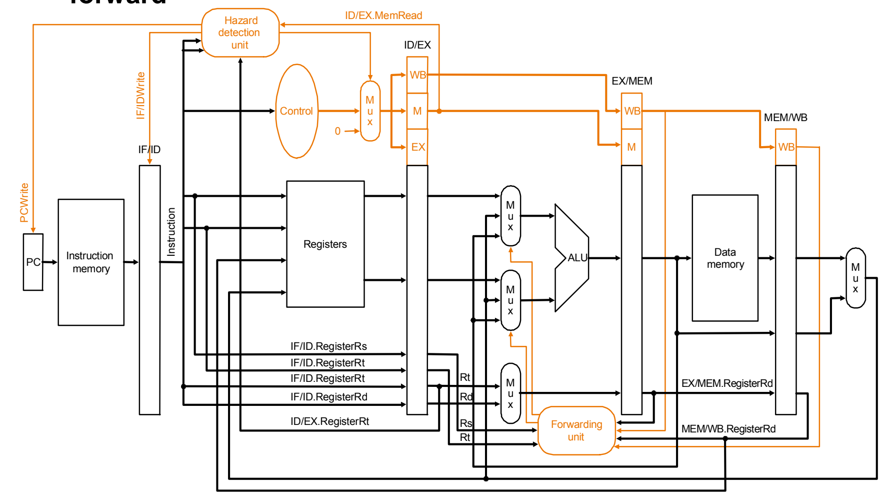
- Load word can still cause a hazard:
  - an instruction tries to read a register following a load instruction that writes to the same register.



- Thus, we need a hazard detection unit to “stall” the load instruction

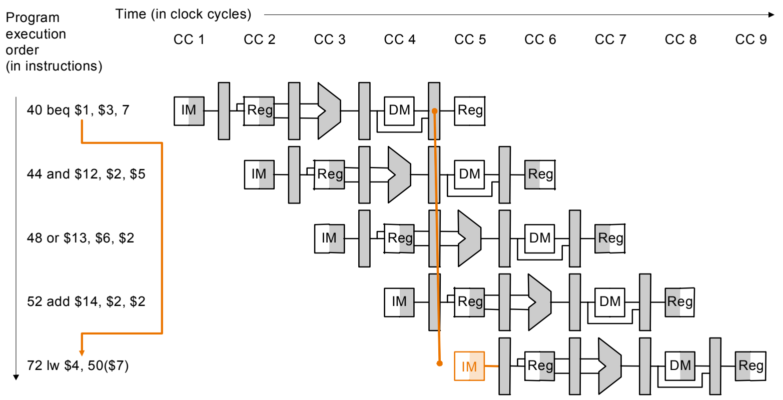
Hazard Detection Unit

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



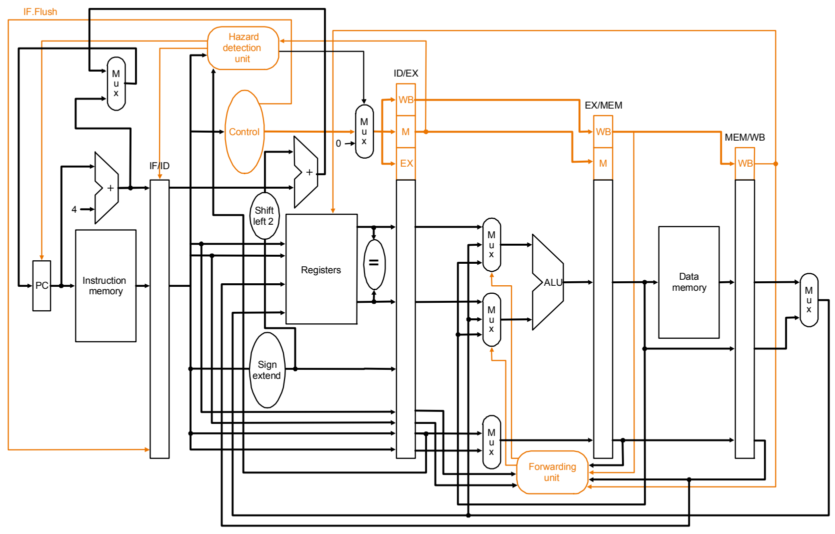
Branch Hazards

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



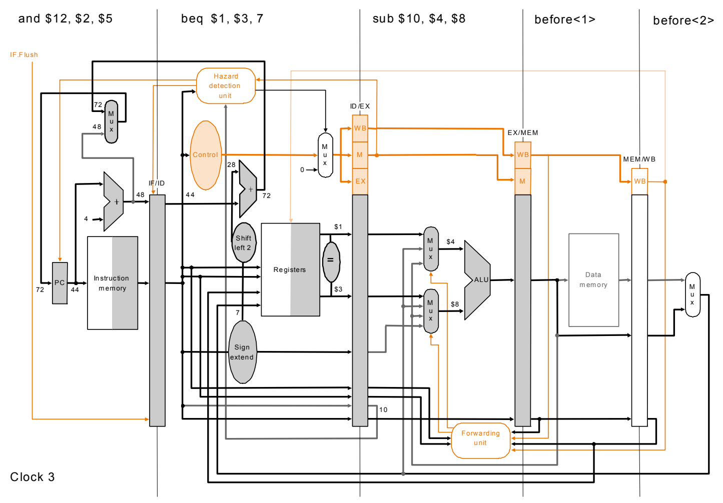
- **We are predicting "branch not taken"**
  - **need to add hardware for flushing instructions if we are wrong**

Flushing Instructions



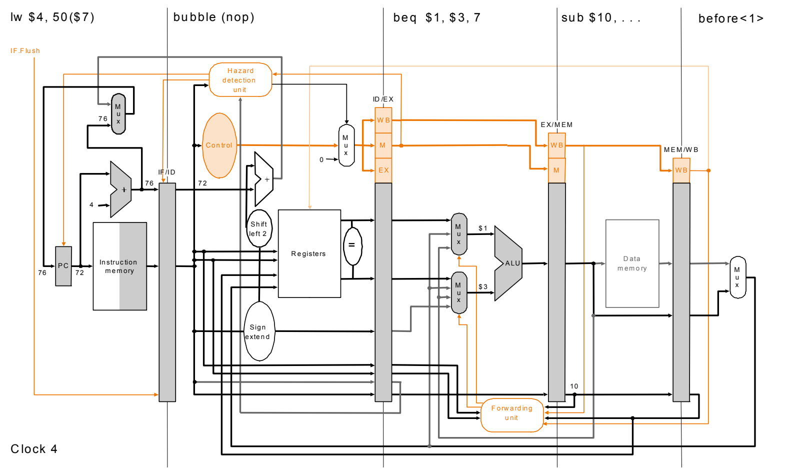
Where is the equality test? When and where instructions are flushed?

Flushing Example



Assume $1 == $3; which instruction will be affected?

Flushing Example (Cont)



Where is the `and` *instruction? Does it make a difference to the program execution?*

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)
```

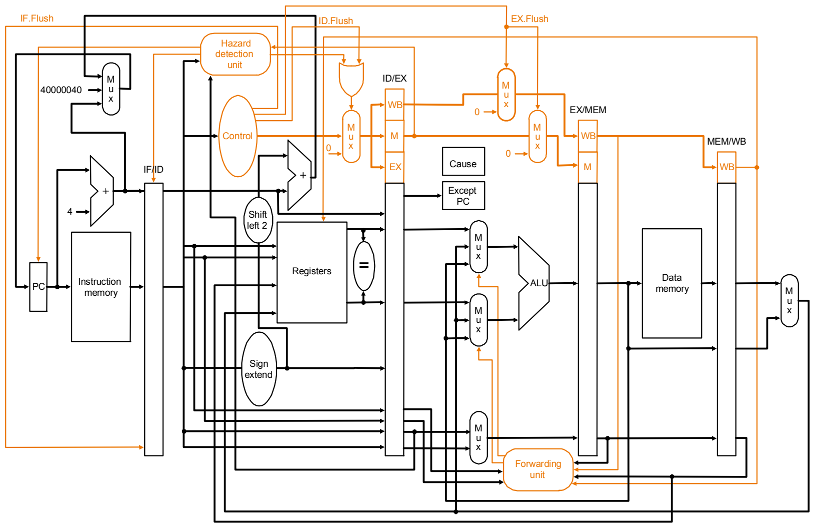
- **Add a "branch delay slot"**
  - **the next instruction after a branch is always executed**
  - **rely on compiler to "fill" the slot with something useful**

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

Other Issues in Pipelines

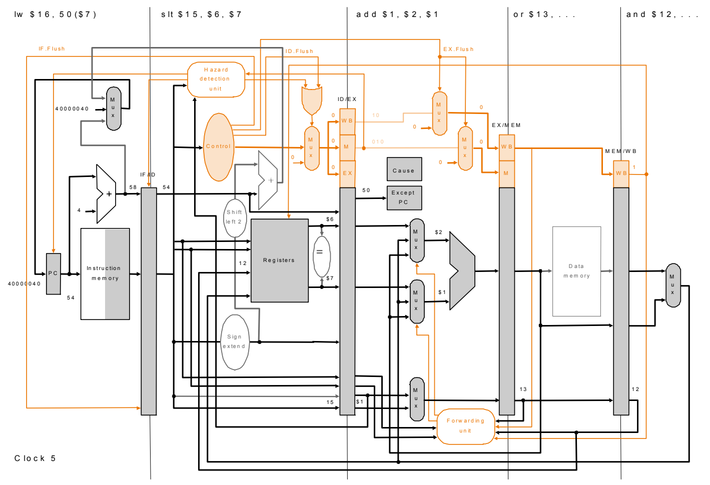
- Exceptions
  - Errors in ALU for arithmetic instructions
  - Memory non-availability
- Exceptions lead to a jump in a program
- However, the current PC value must be saved so that the program can return to it back for recoverable errors
- Multiple exception can occur in a pipeline
- Preciseness of exception location is important in some cases
- I/O exceptions are handled in the same manner

Datapath/Controls to Handle Exceptions



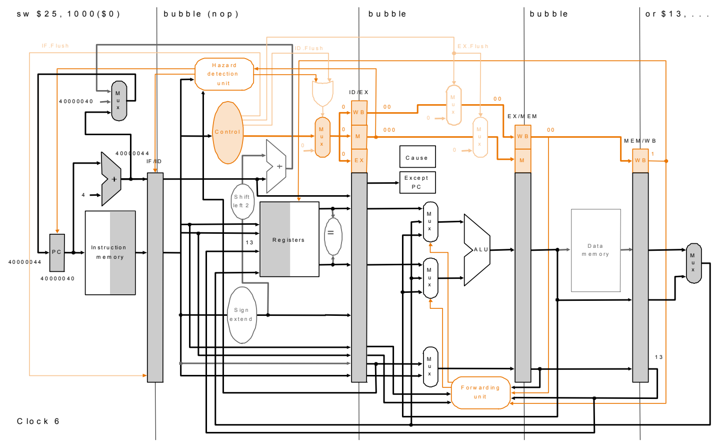
What is address 0x40000040? Where should be the ALU overflow signal?

Exception Example



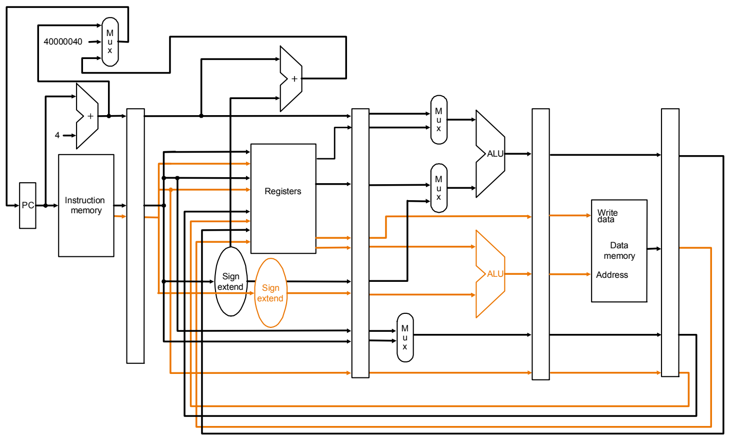
Assume "add $1, $2, $1" overflows; which instructions to flush?

Exception Example (Cont)



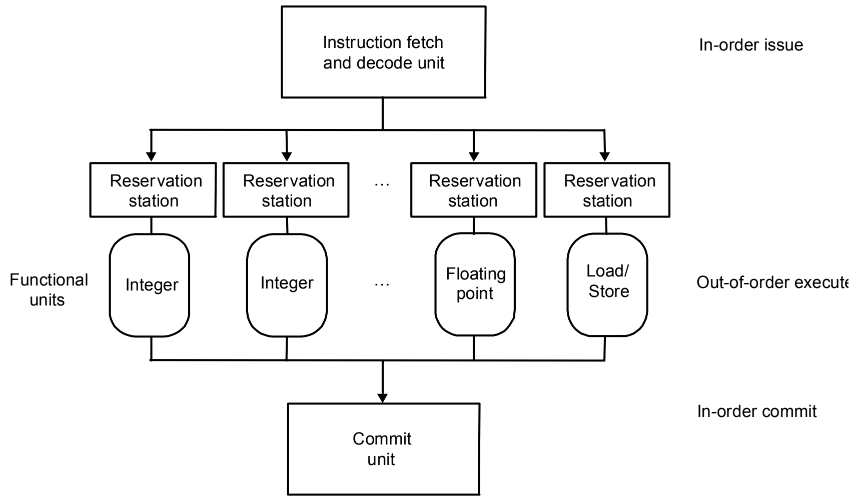
Where is "sw $25, 1000($0)" from?

Superscalar Architecture



Where are the complexities?

A Modern Pipelined Microprocessor



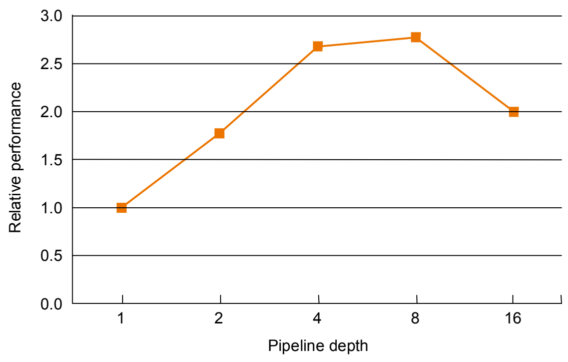
Big issue: complexity! How about dependence detection, forwarding, and exception handling?

Important Facts to Remember

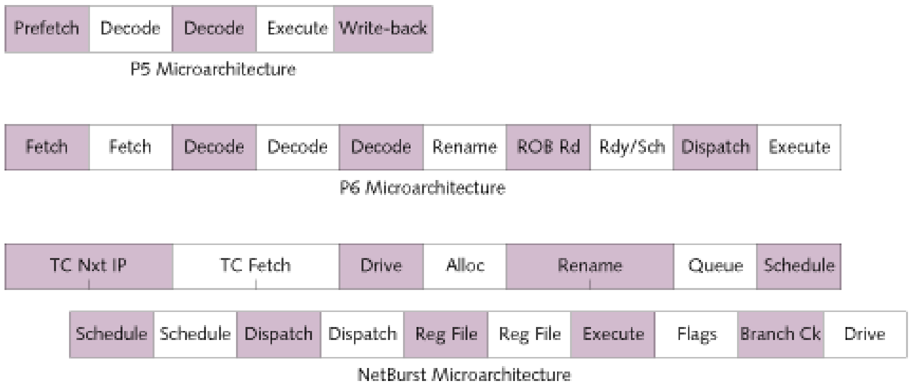
- Pipelined processors divide the execution in multiple steps
- However pipeline hazards reduce performance
  - Structural, data, and control hazard
- Data forwarding helps resolve data hazards
  - But all hazards cannot be resolved
  - Some data hazards require bubble or noop insertion
- Effects of control hazard reduced by branch prediction
  - Predict always taken, delayed slots, branch prediction table
  - Structural hazards are resolved by duplicating resources
- Time of n instructions depends on n, # of stages k, # of control hazard and penalty of each step, # of data hazards and penalty for each
- Time = n + k - 1 + load hazard penalty + branch penalty
- Load hazard penalty is 1 or 0 cycle depending on data use with forwarding
- branch penalty is 3, 2, 1, or zero cycles depending on scheme

Design and Performance Issues With Pipelining

- Pipelined processors are not EASY to design
- Technology affect implementation
- Instruction set design affect the performance, i.e., beq, bne
- More stages do not lead to higher performance



Real Stuff: Pentium, Pentium Pro, Pentium 4 Pipeline



- **Pentium (P5) = 5 stages**
  Pentium Pro, II, III (P6) = 10 stages
- **Pentium 4 (NetBurst) = 20 stages**
- ***What are critical to performance?***

From "Pentium 4 (Partially) Previewed," Microprocessor Report, 8/28/00