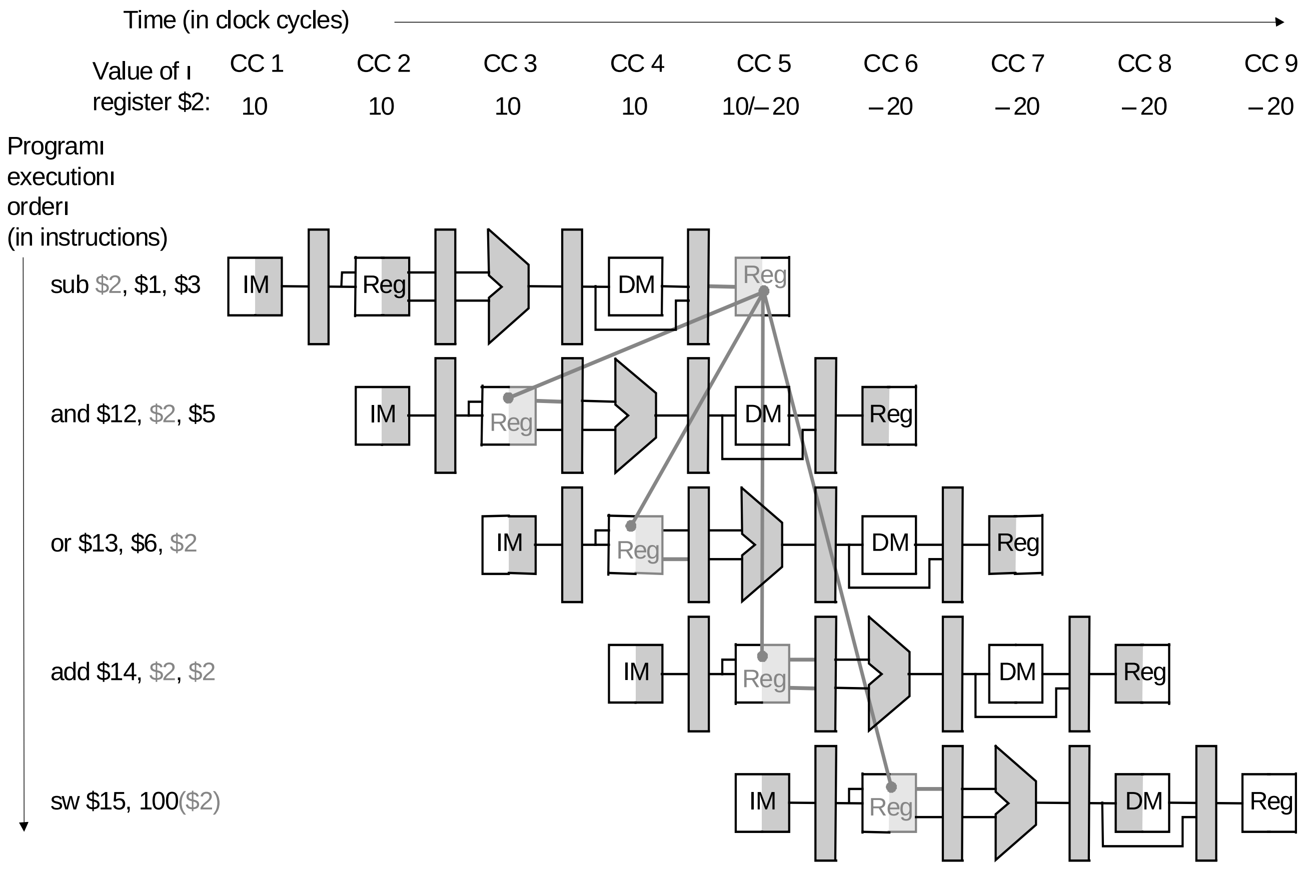
Data and control hazards

- Data hazards:
- Detecting dependencies
- Forwarding
- Stalls
- Detecting branch hazards
- Reducing the delay of branches

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)
```

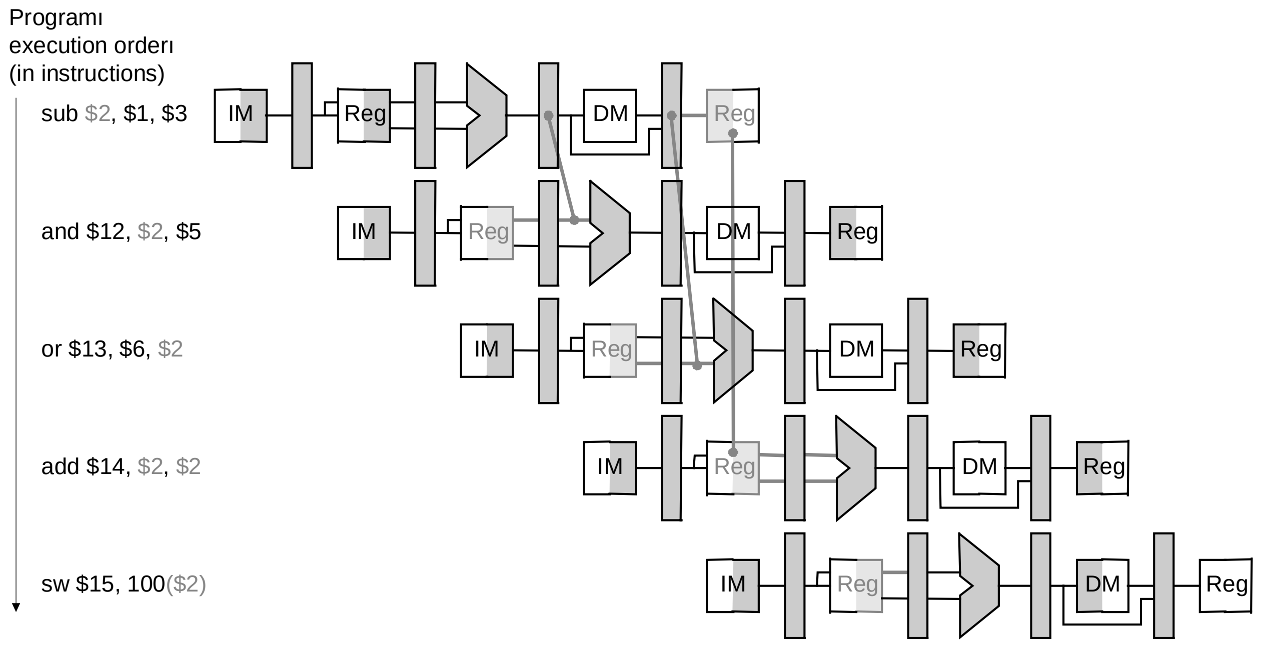
- Problem: this really slows us down!

Forwarding

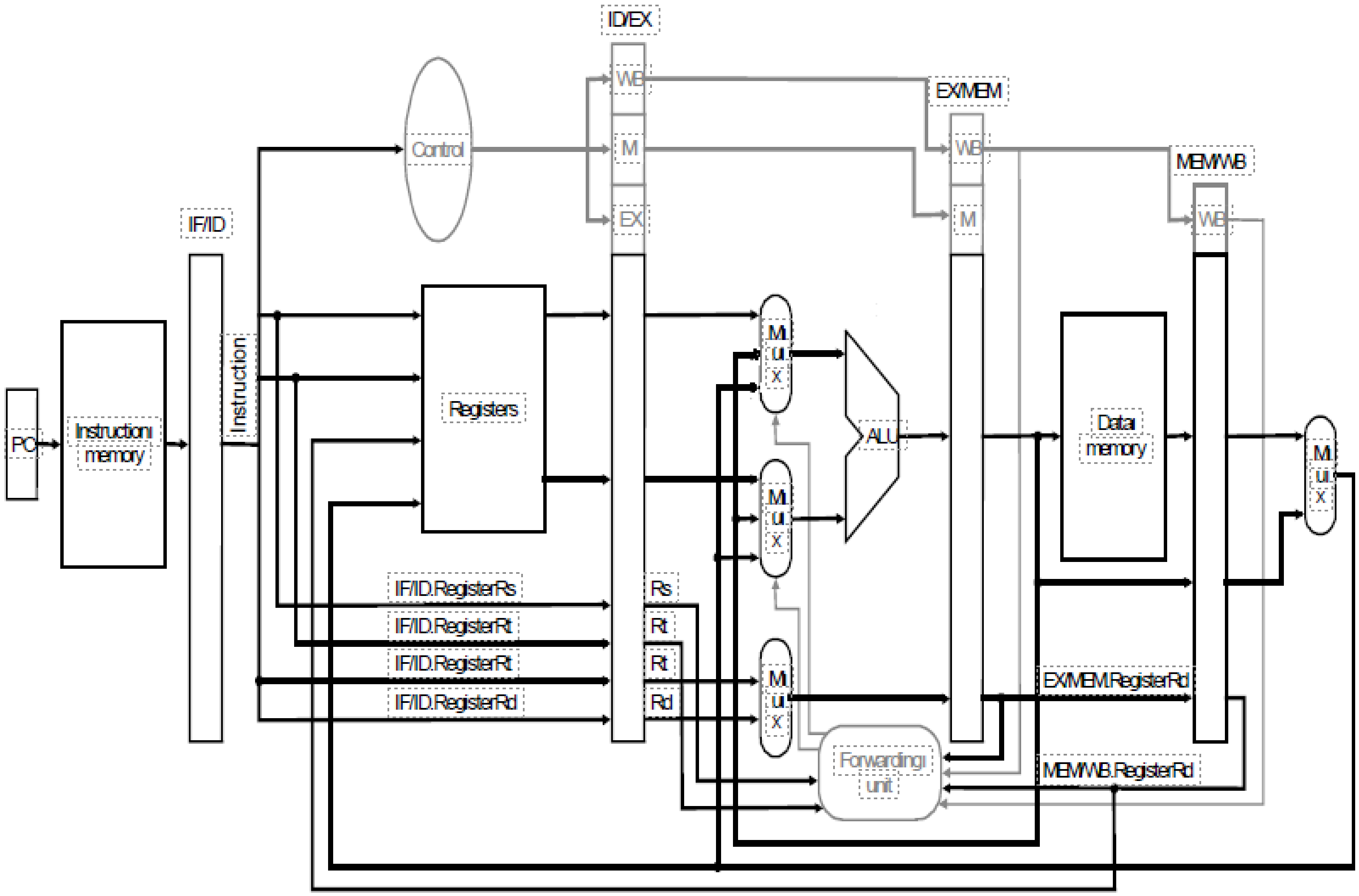
- Use temporary results, don't wait for them to be written
  - register file forwarding to handle read/write to same register
  - ALU forwarding

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

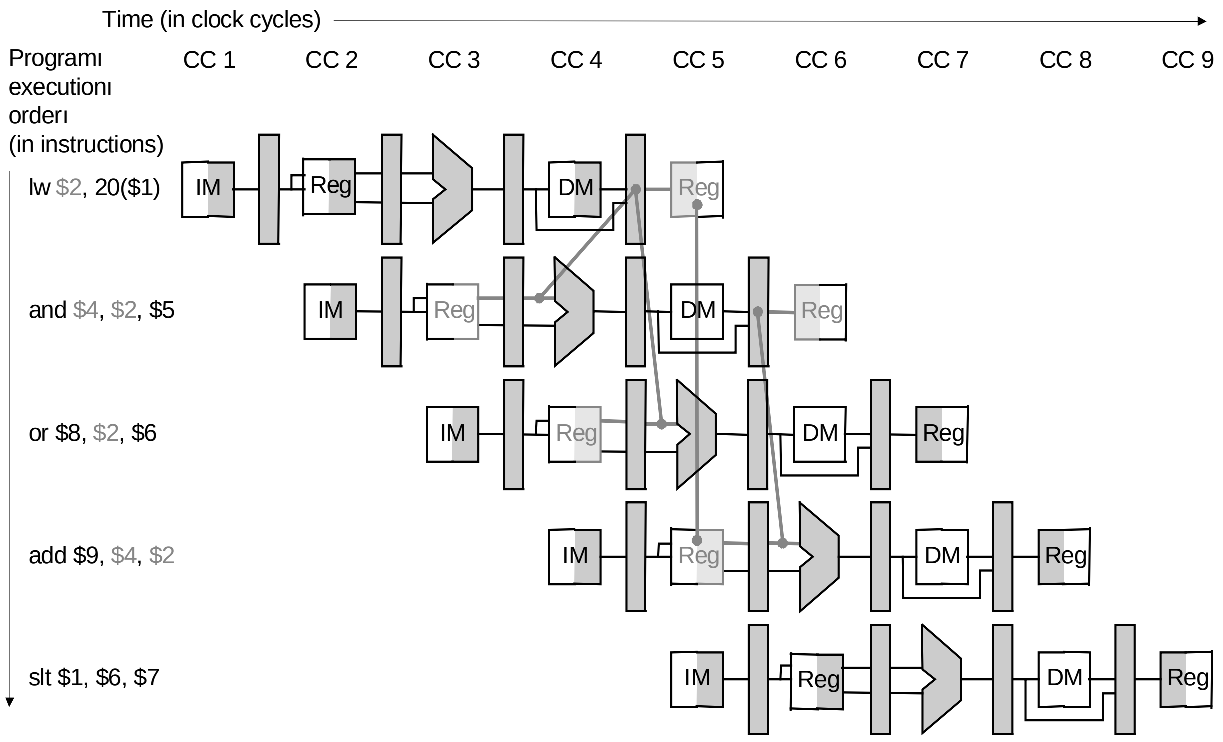


Forwarding


ID/EX
WB
M
EX
Control
EX/MEM
WB
M
MEM/WB
WB
IF/ID
PC
Instruction memory
Instruction
Registers
Mux
Mux
Mux
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

Data hazards and stalls

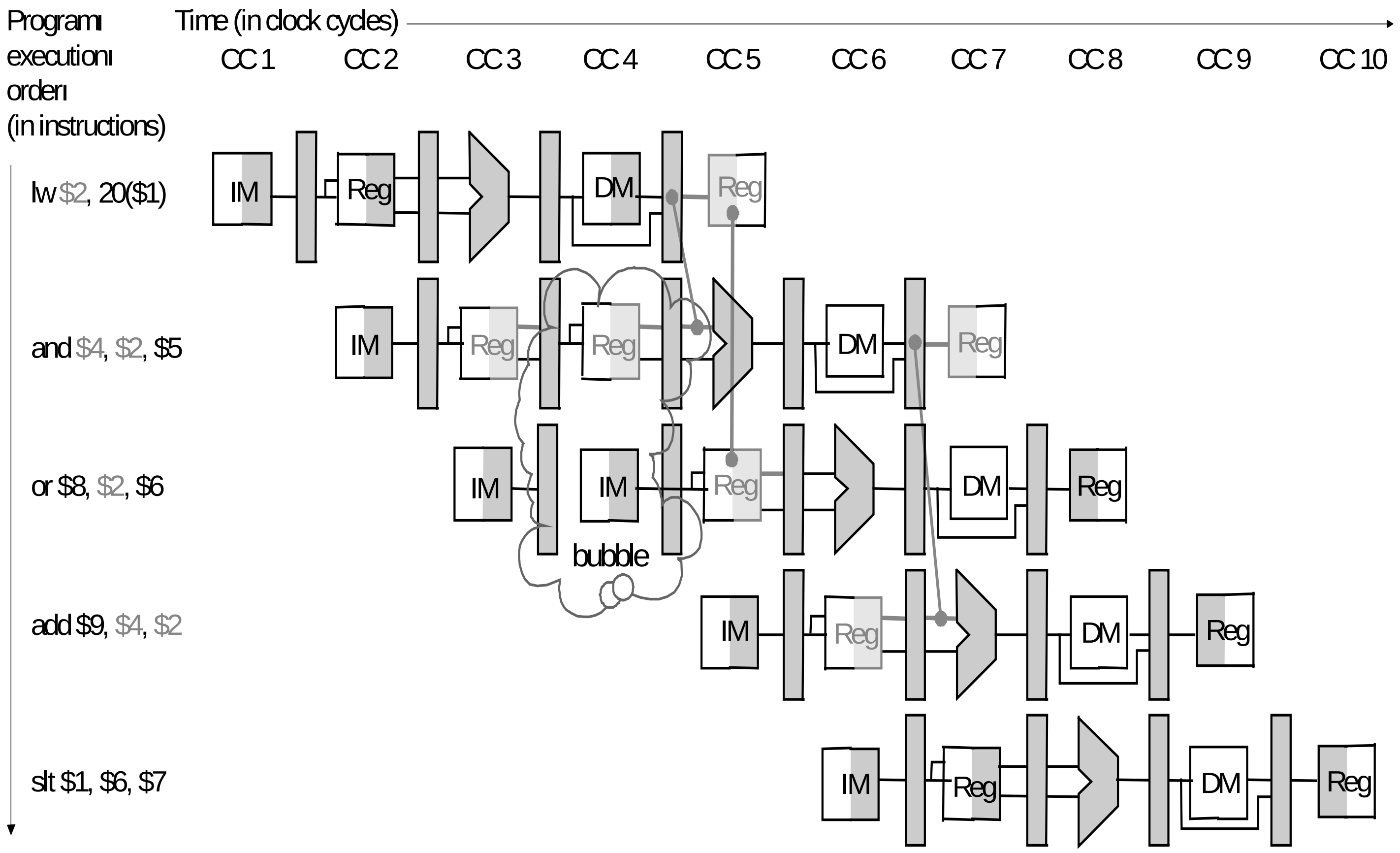
- **Load word can still cause a hazard:**
  - **an instruction tries to read a register following a load instruction that writes to the same register.**
  -



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

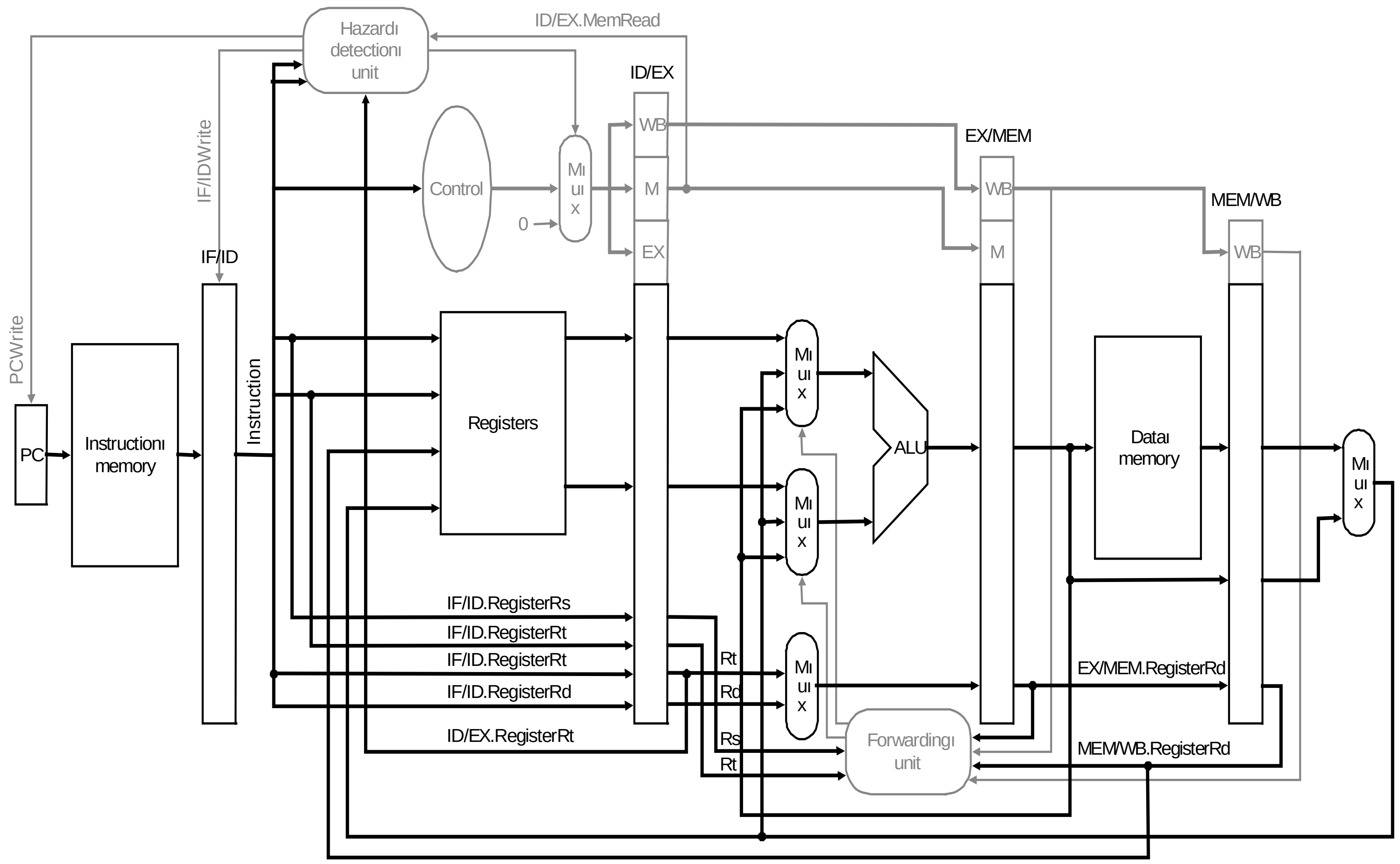
Stalling

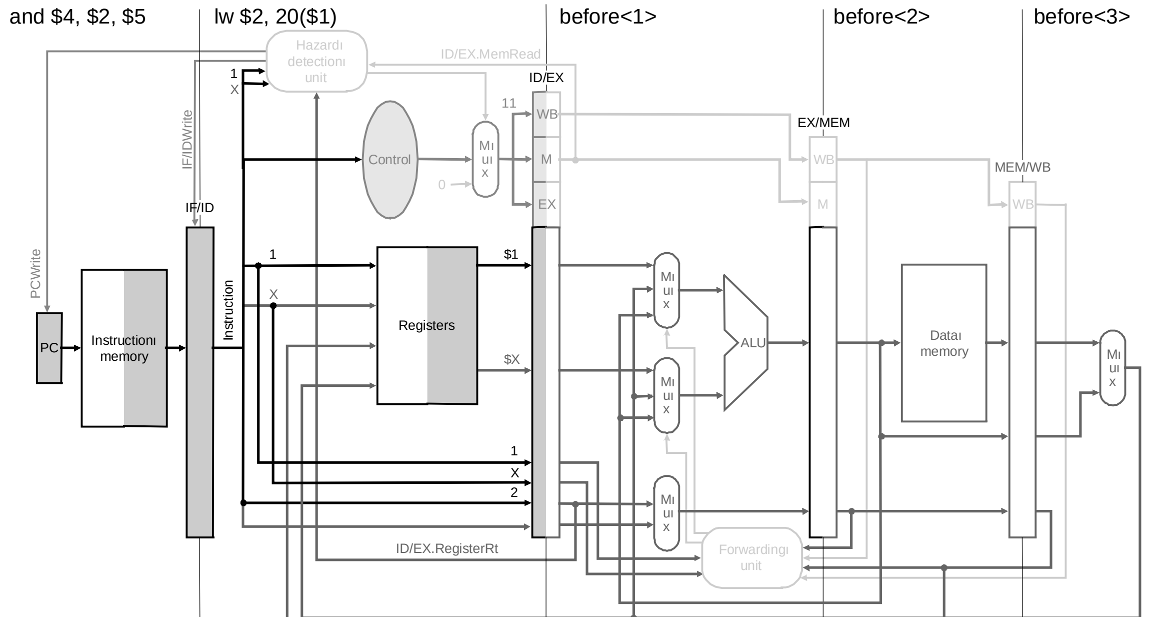
- **We can stall the pipeline by keeping an instruction in the same stage**



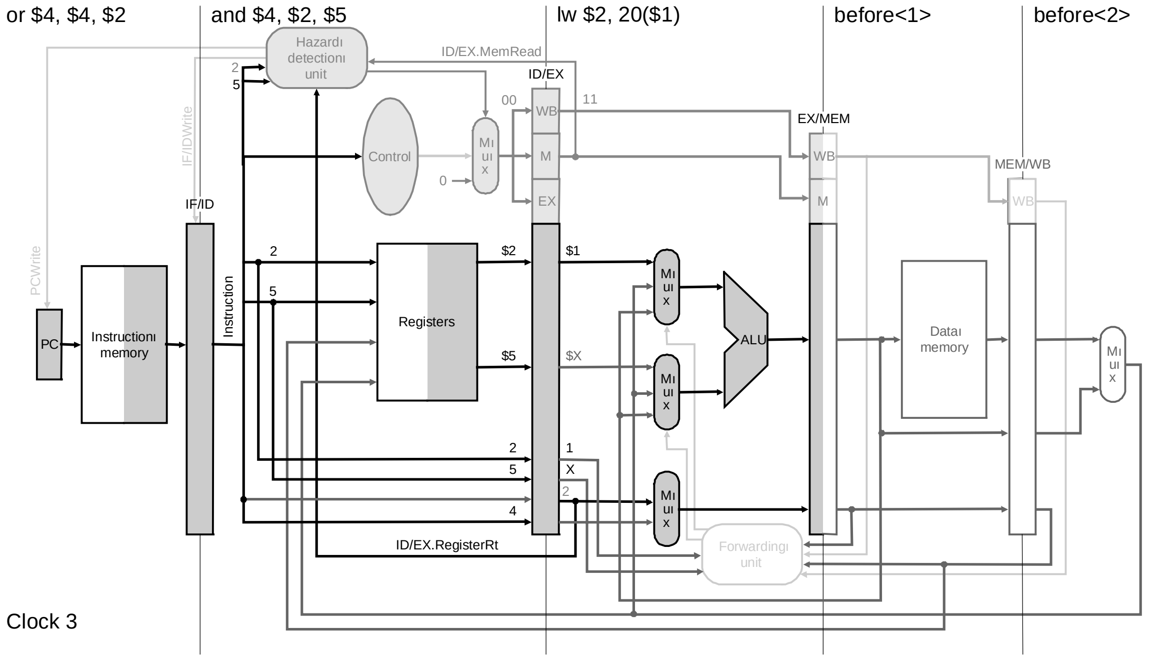
Hazard Detection Unit

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

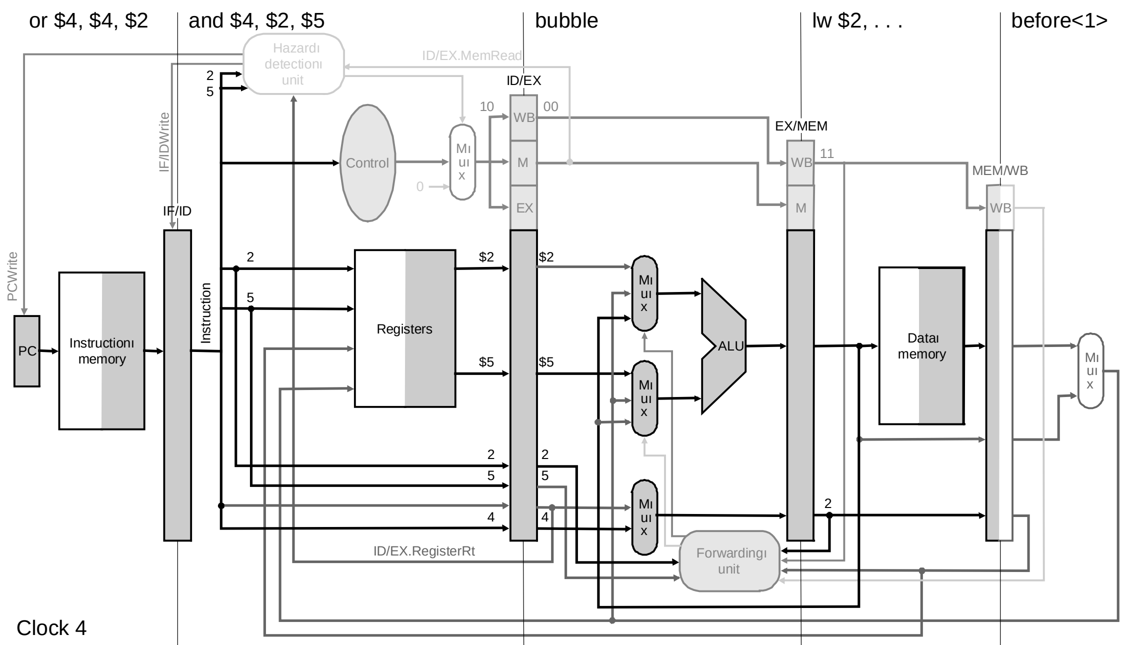


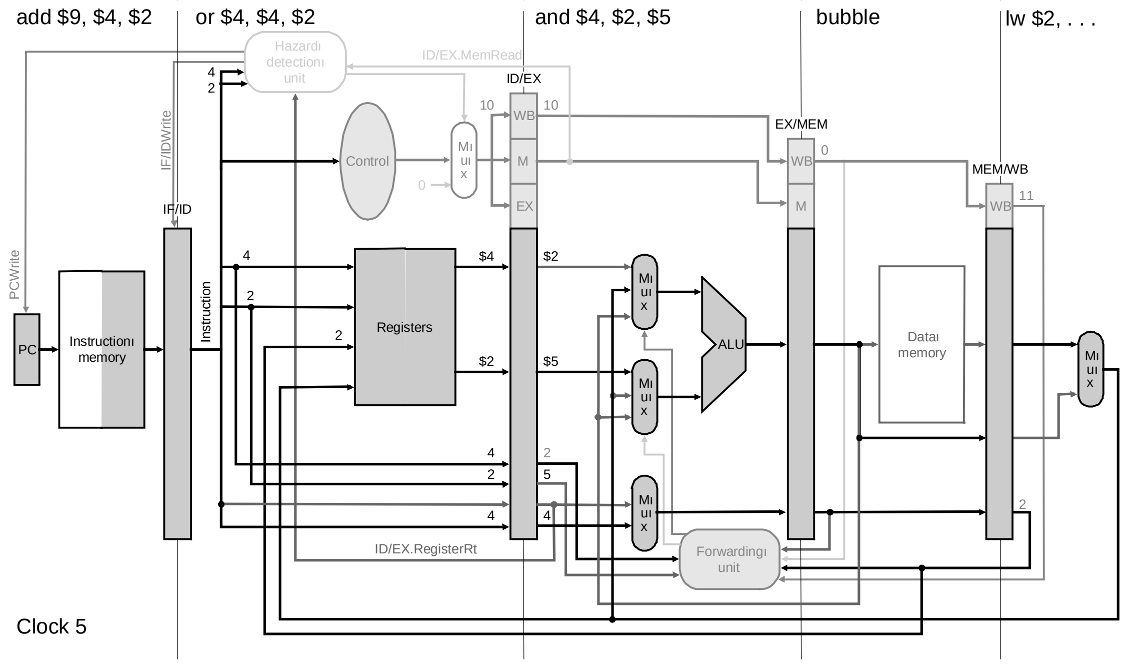


Clock 2



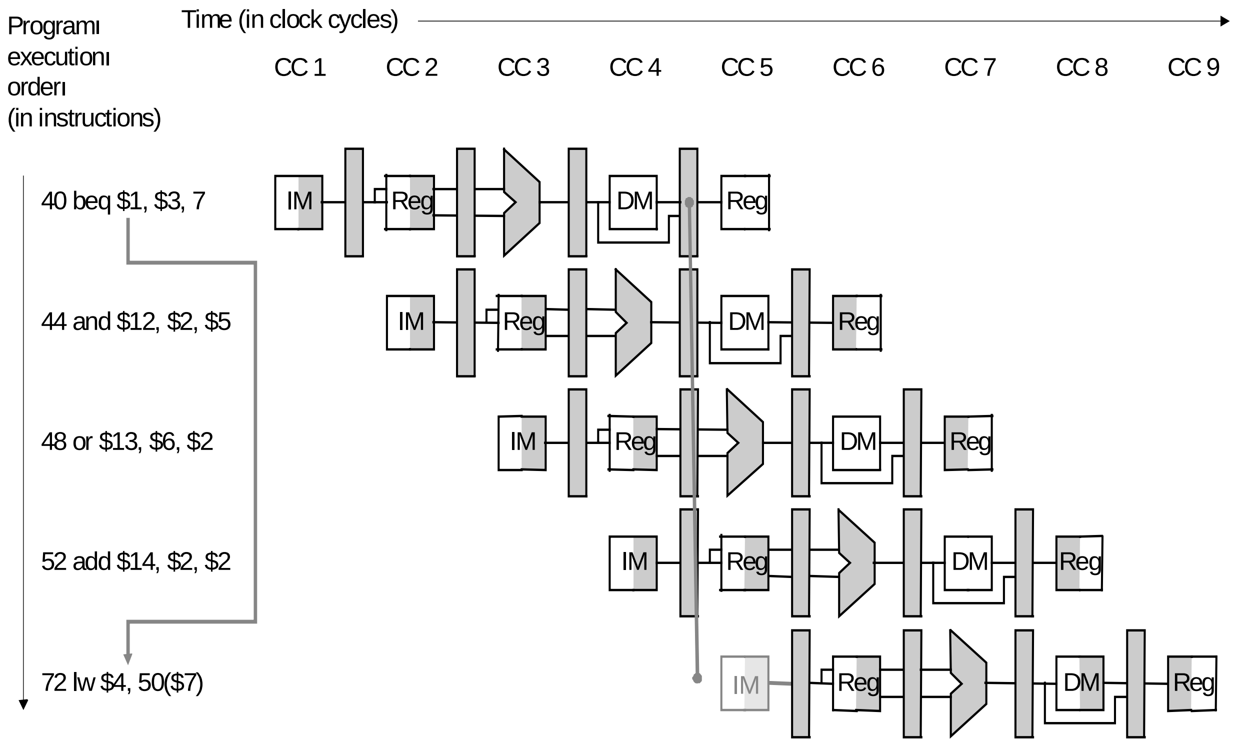
Clock 3


or $4, $4, $2
and $4, $2, $5
bubble
lw $2, . . .
before<1>
Hazard detection unit
ID/EX.MemRead
ID/EX
EX/MEM
MEM/WB
IF/ID
IF/IDWrite
PCWrite
PC
Instruction memory
Instruction
Control
Mux
Registers
ALU
Data memory
Forwarding unit
ID/EX.RegisterRt
WB
M
EX
10
00
11
0
2
5
4
$2
$5
Clock 4


add $9, $4, $2
or $4, $4, $2
and $4, $2, $5
bubble
lw $2, . . .
Hazard detection unit
ID/EX.MemRead
ID/EX
EX/MEM
MEM/WB
IF/ID
IF/IDWrite
PCWrite
PC
Instruction memory
Instruction
Control
Mux
Registers
ALU
Data memory
Forwarding unit
ID/EX.RegisterRt
WB
M
EX
10
0
11
4
2
5
$4
$2
$5
Clock 5

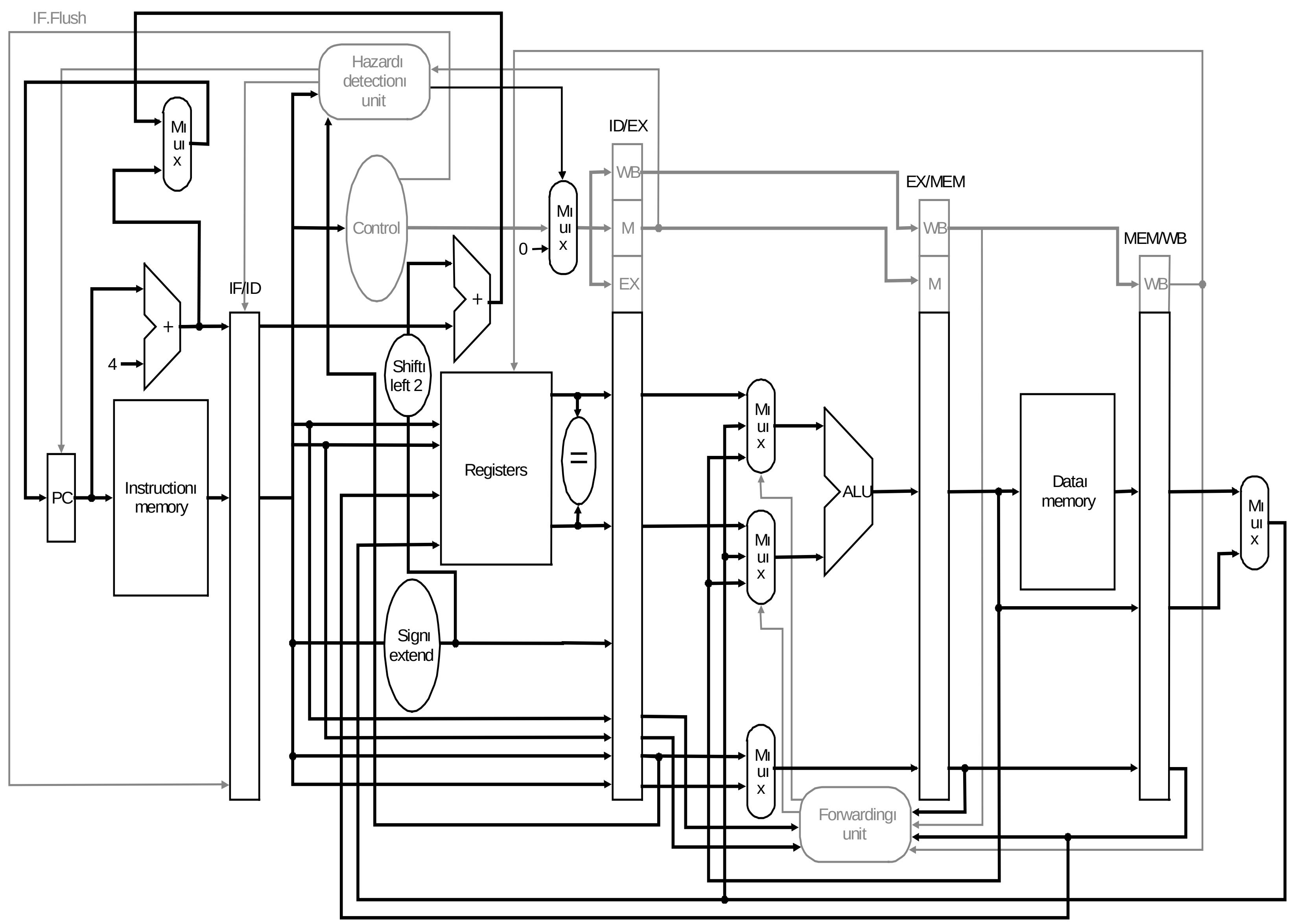
Branch Hazards

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

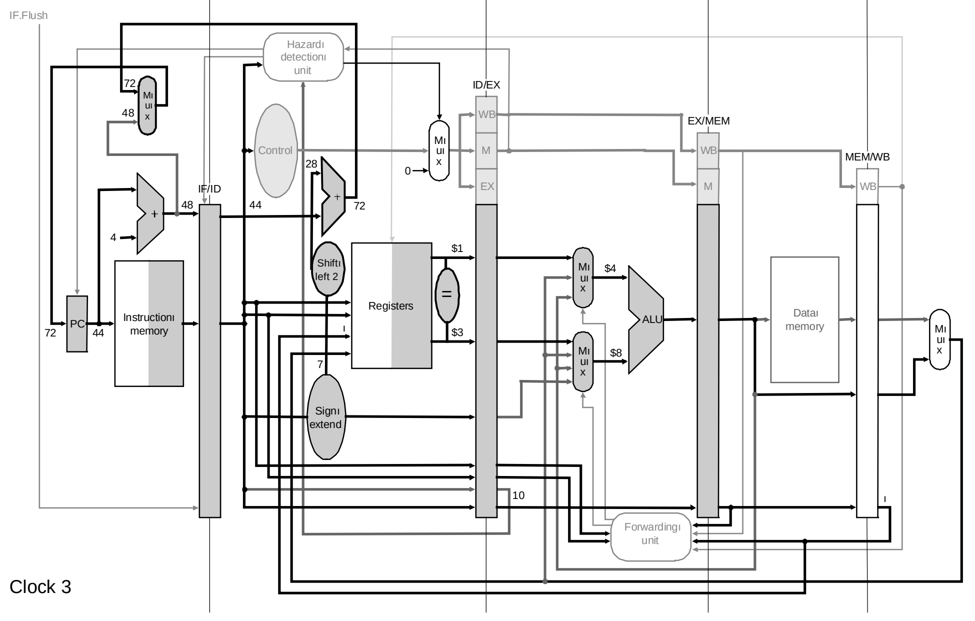


- **We are predicting “branch not taken”**
  - **need to add hardware for flushing instructions if we are wrong**

Flushing Instructions

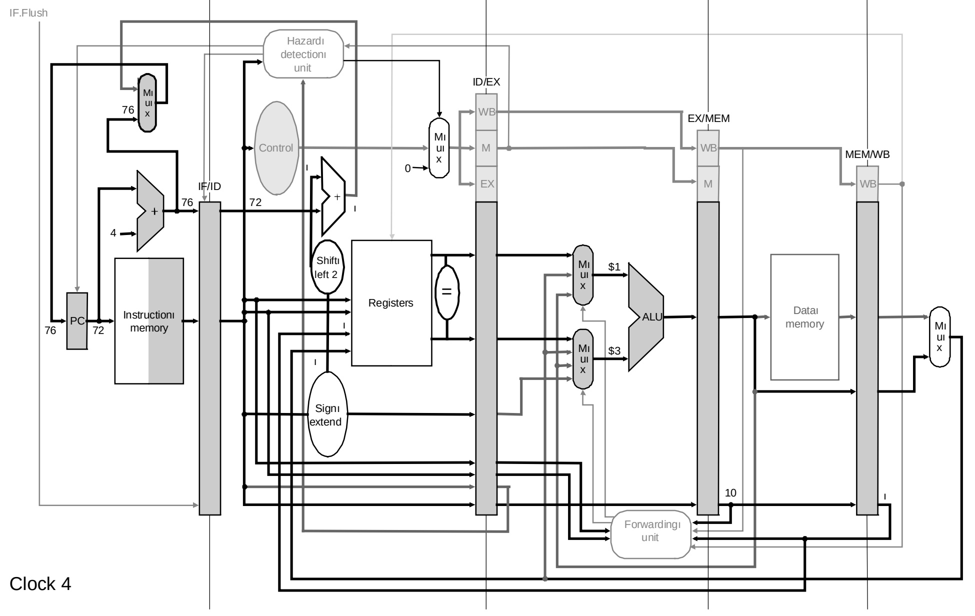


and $12, $2, $5 beq $1, $3, 7 sub $10, $4, $8 before<1> before<2>



Clock 3

lw $4, 50($7) bubble (nop) beq $1, $3, 7 sub $10, . . . before<1>



Clock 4

Advanced Pipelining

- **Longer pipelines - Superpipelining**
- **Replicating components of the datapath - Multiple instruction per cycle (superscalar)**
- **Dynamic pipeline shceduling - avoid stalls**

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
  - longer pipeline
  - branch history table
  - compiler technology important