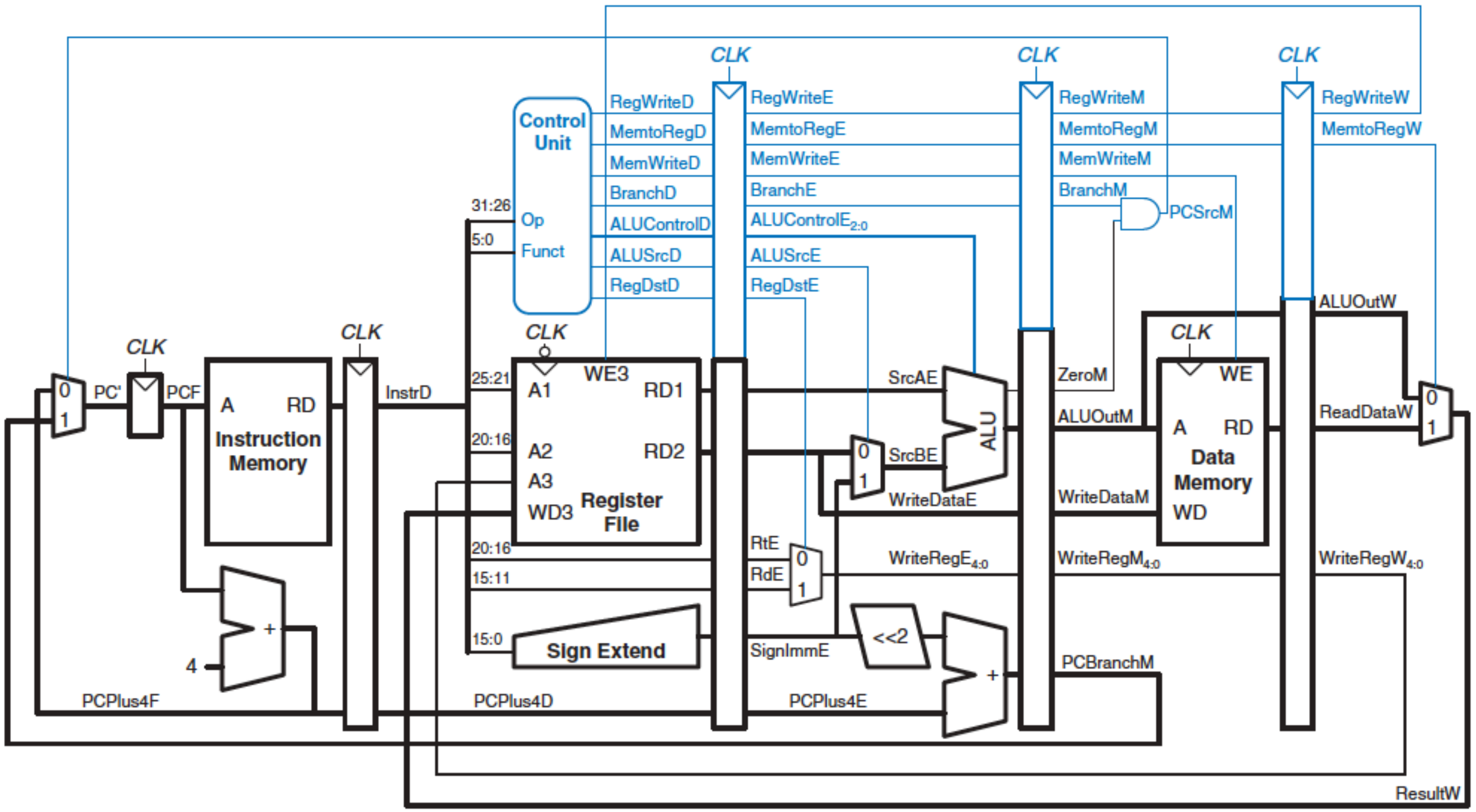
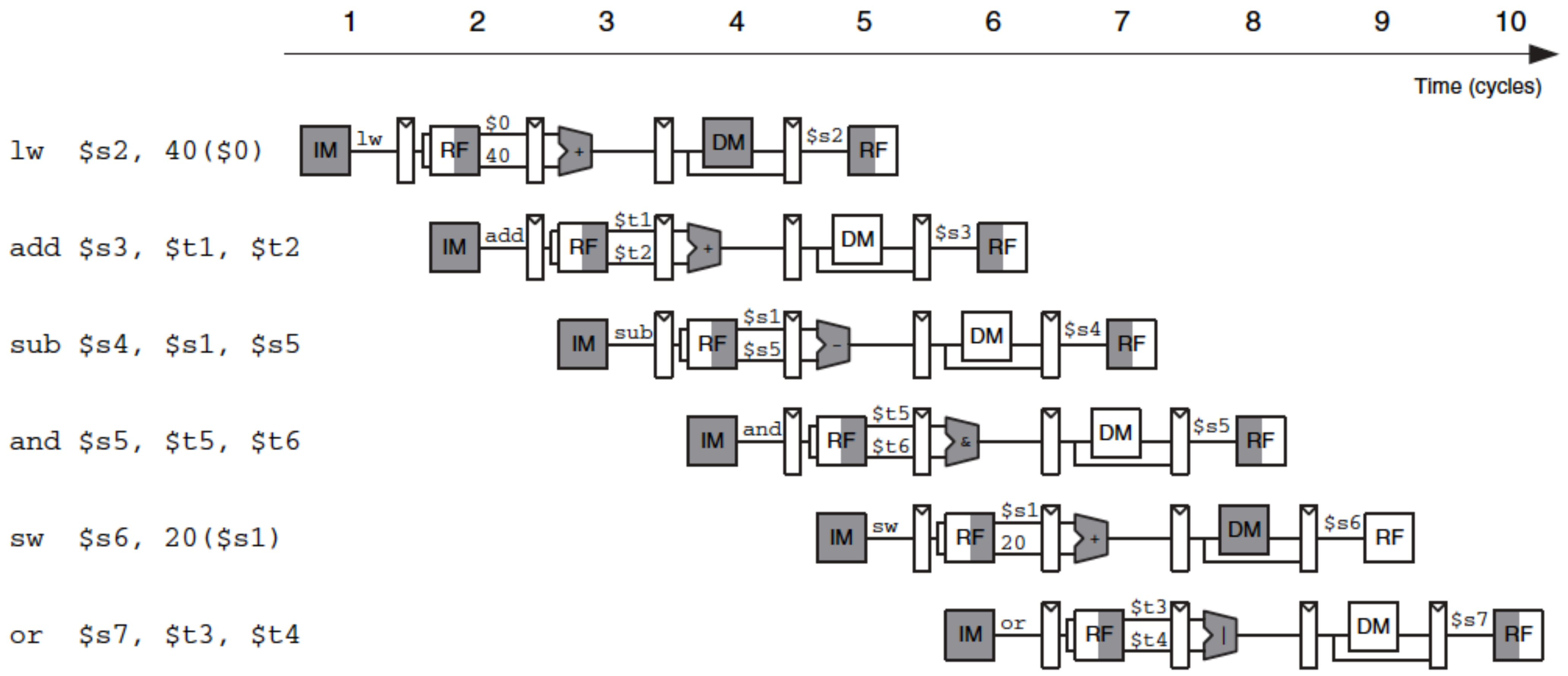
Pipeline Data Hazards


CLK
CLK
CLK
CLK
CLK
CLK
CLK
Control Unit
Op
Funct
31:26
5:0
RegWriteD
MemtoRegD
MemWriteD
BranchD
ALUControlD
ALUSrcD
RegDstD
RegWriteE
MemtoRegE
MemWriteE
BranchE
ALUControlE2:0
ALUSrcE
RegDstE
RegWriteM
MemtoRegM
MemWriteM
BranchM
PCSrcM
RegWriteW
MemtoRegW
PC'
PCF
A
RD
Instruction Memory
InstrD
25:21
20:16
20:16
15:11
15:0
WE3
A1
A2
A3
WD3
RD1
RD2
Register File
Sign Extend
SrcAE
SrcBE
ALU
WriteDataE
RtE
RdE
WriteRegE4:0
SignImmE
<<2
PCPlus4F
PCPlus4D
PCPlus4E
4
ZeroM
ALUOutM
WriteDataM
WriteRegM4:0
PCBranchM
WE
A
RD
Data Memory
WD
ALUOutW
ReadDataW
WriteRegW4:0
ResultW

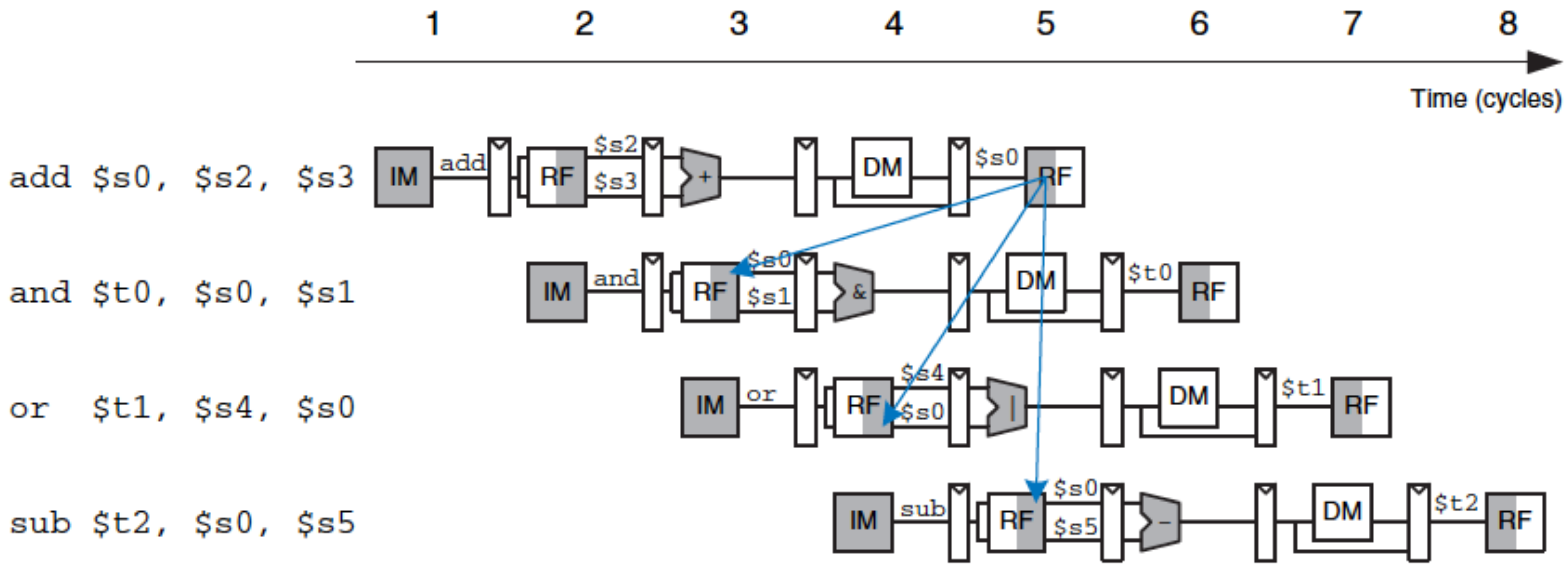

1
2
3
4
5
6
7
8
9
10
Time (cycles)
lw $s2, 40($0)
add $s3, $t1, $t2
sub $s4, $s1, $s5
and $s5, $t5, $t6
sw $s6, 20($s1)
or $s7, $t3, $t4
IM
RF
DM
lw
$0
40
$s2
add
$t1
$t2
$s3
sub
$s1
$s5
$s4
and
$t5
$t6
$s5
sw
$s1
20
$s6
or
$t3
$t4
$s7

Hazards

When one instruction is dependent on another that has not yet completed a hazard occurs.

Can read and write the register file in one cycle.

- Write takes place during the first half of the cycle
- Read during the second half.
- No introduction of a hazard that way.


1
2
3
4
5
6
7
8
Time (cycles)
add $s0, $s2, $s3
and $t0, $s0, $s1
or $t1, $s4, $s0
sub $t2, $s0, $s5
IM
add
RF
$s2
$s3
+
DM
$s0
RF
and
$s0
$s1
&
$t0
or
$s4
$s0
|
$t1
sub
$s0
$s5
-
$t2

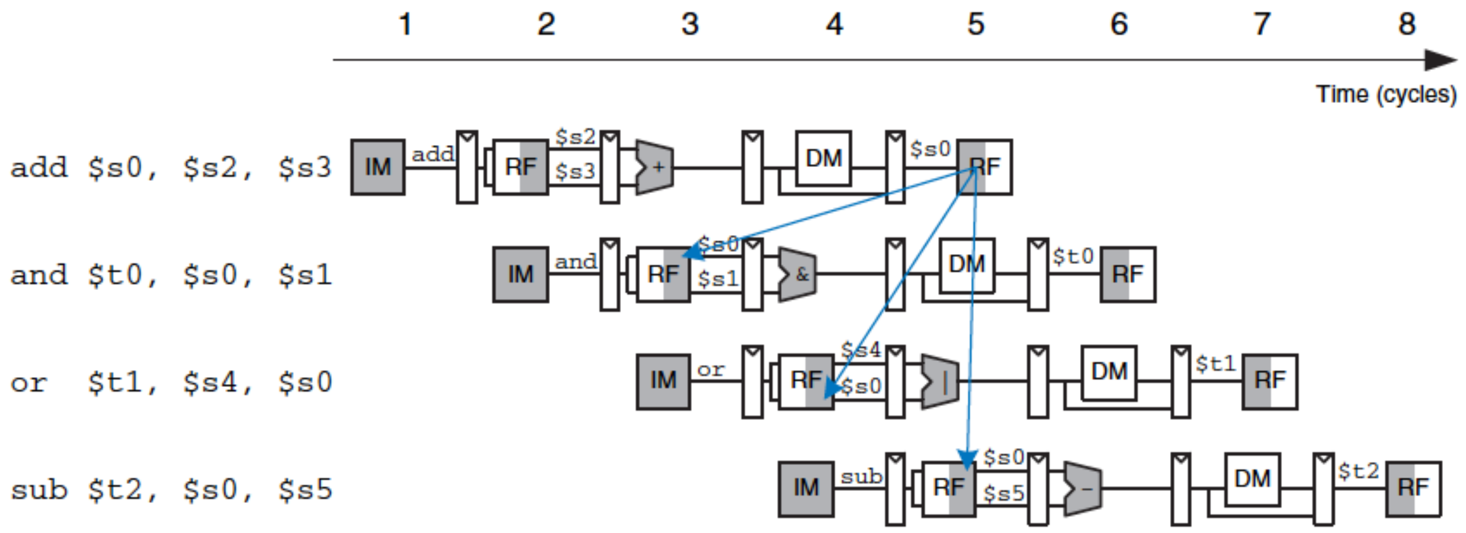
Hazard

One instruction writes to $s0 and subsequent instructions read the register

- Read after Write hazard (RAW)

Hazard when instruction writes on register and either of the two subsequent instructions read on that register.

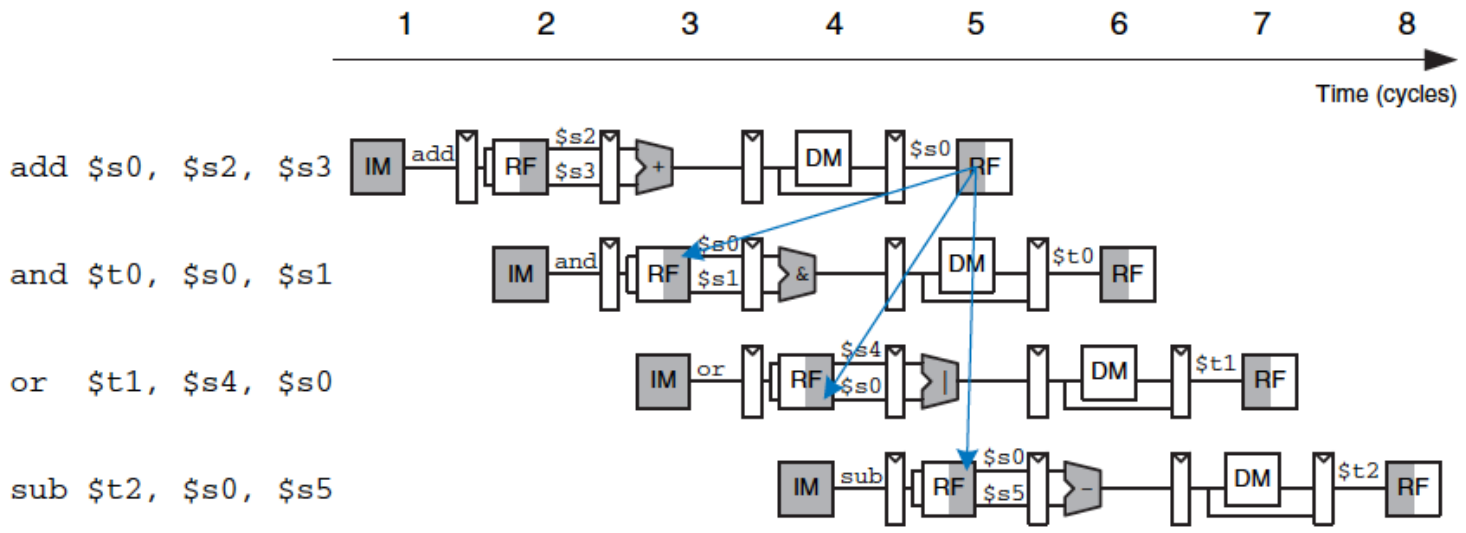
- Will give the wrong result unless we are able to mitigate.



Hazard

Hazard when instruction writes on register and either of the two subsequent instructions read on that register.

When is the value actually computed? When is it used by the next instruction?



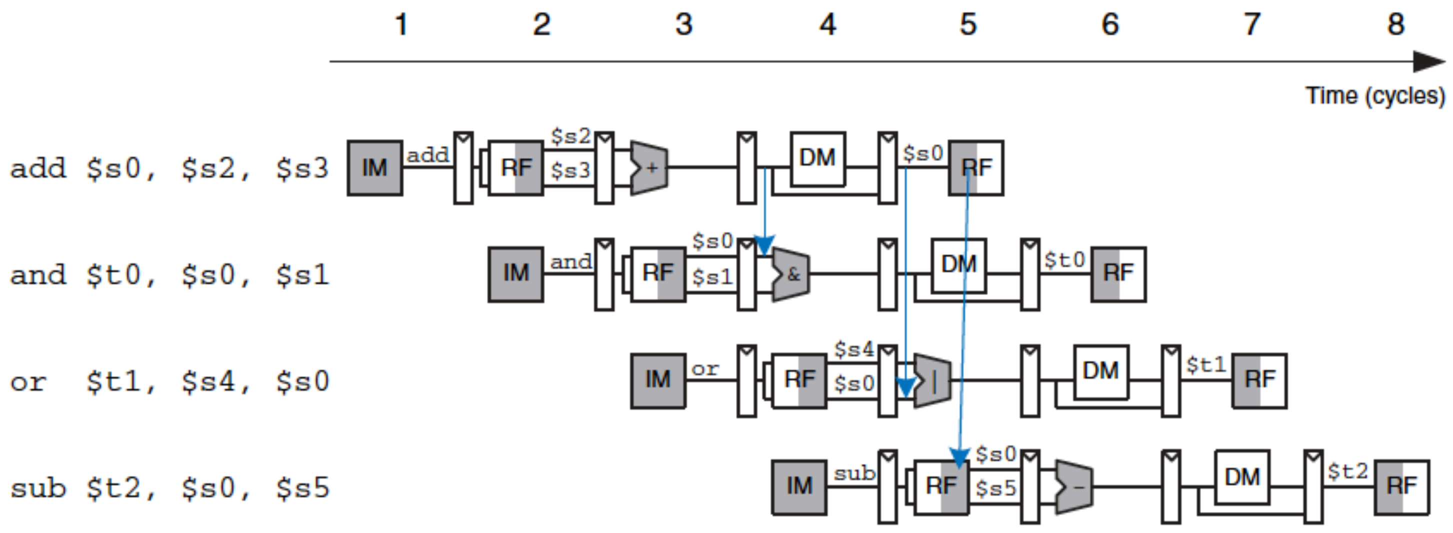
```
add $s0, $s2, $s3
and $t0, $s0, $s1
or  $t1, $s4, $s0
sub $t2, $s0, $s5
```

Solving Data Hazards with Forwarding

Can solve some by forwarding (bypassing) a result from Memory or Writeback stages to a dependent instruction in the Execute stage.

Forward the result from one instruction to the next to resolve a RAW hazard

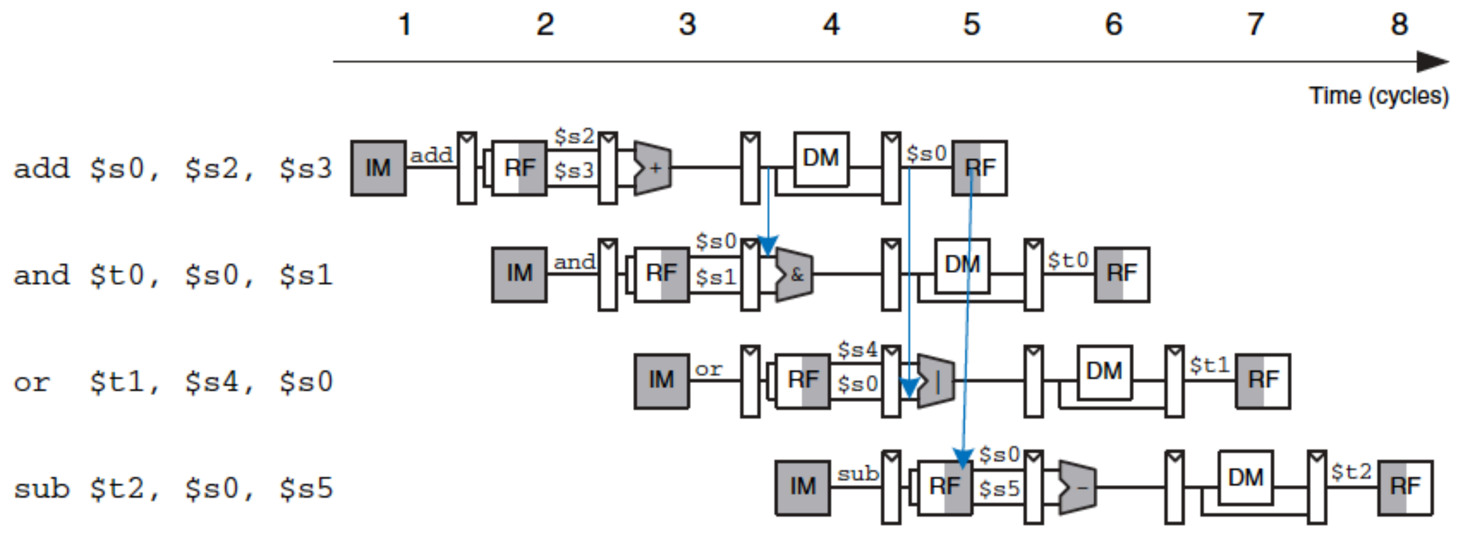
- Doesn't slow down the pipeline!



Forwarding

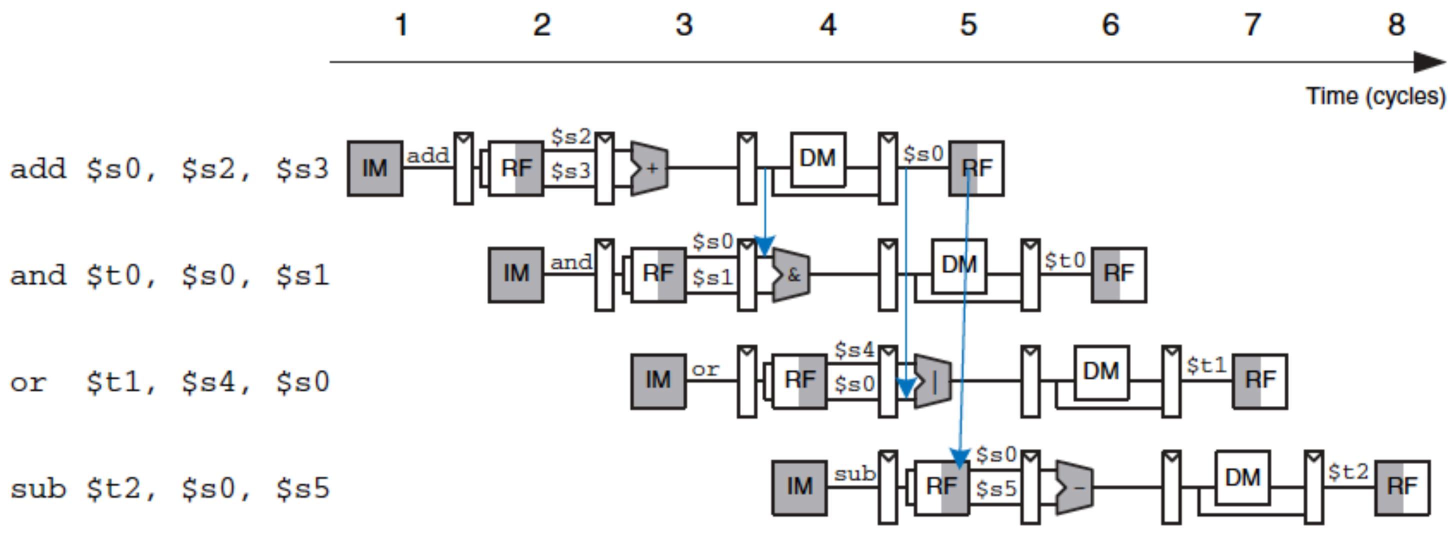
Cycle 4: $s0 is forwarded from Memory stage of add to Execute stage of the or instruction

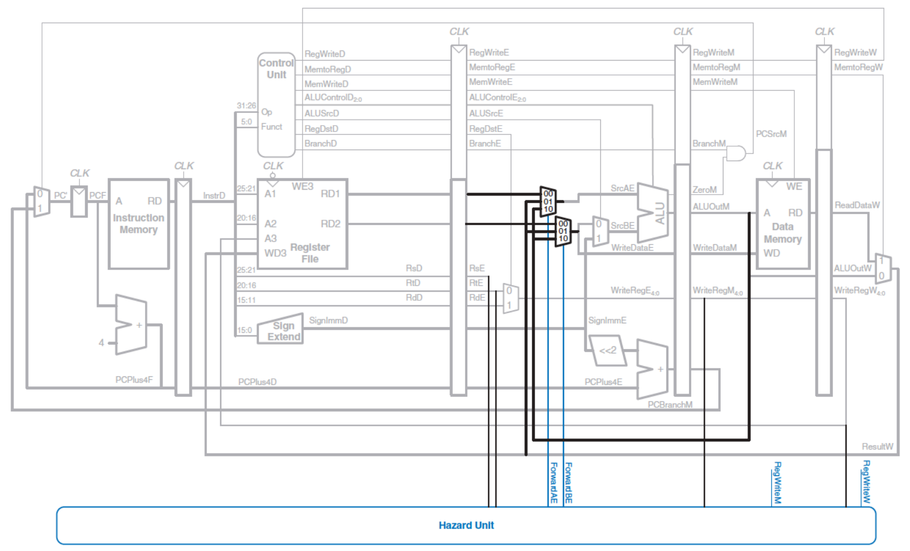
Cycle 5: $s0 is forwarded from Writeback stage of add to Execute of the or instruction

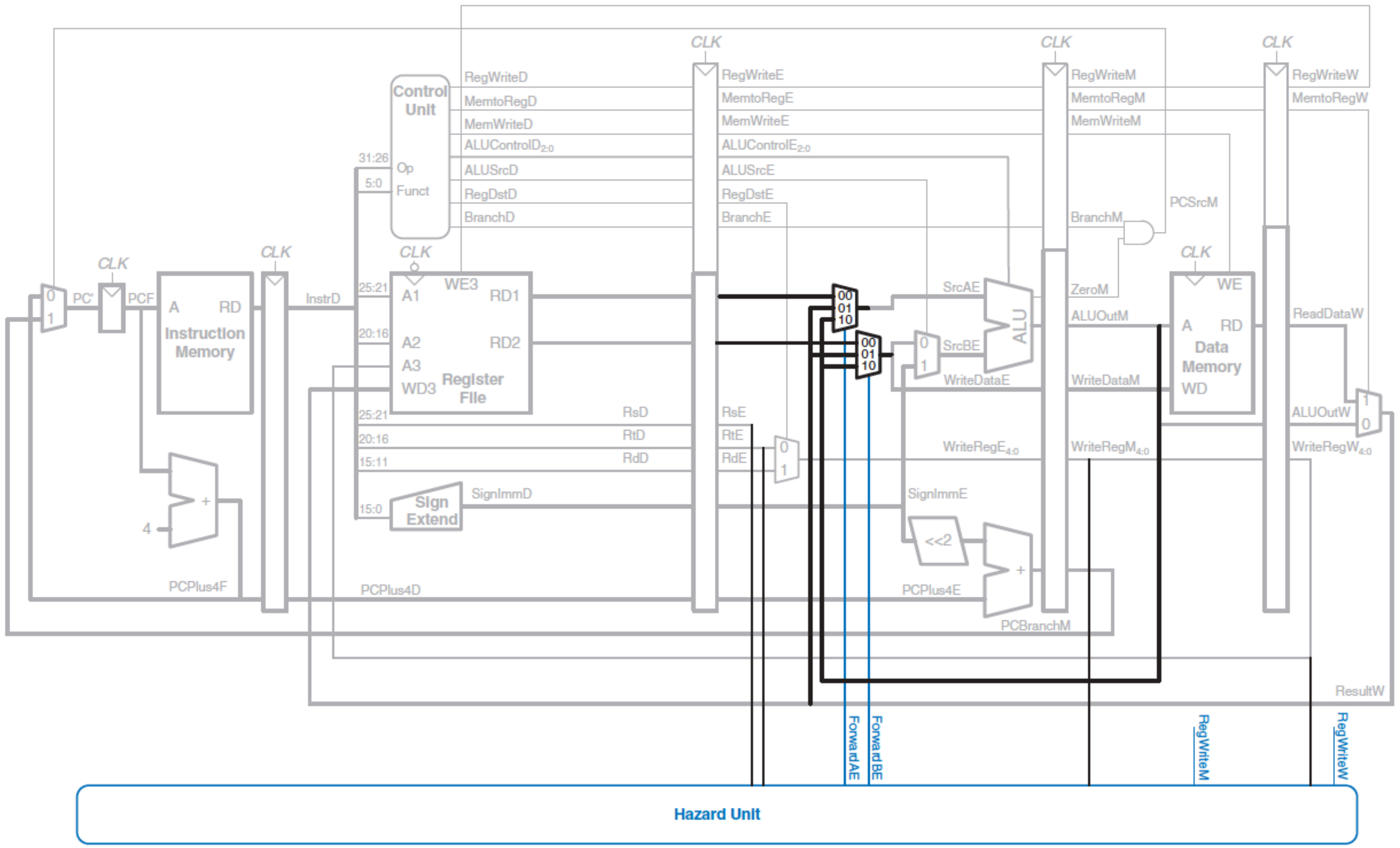


Implementing Forwarding

Add MUX in front of the ALU to determine whether operand is coming from register file, or the Mem/Writeback stage.






CLK
Control Unit
Op
Funct
31:26
5:0
RegWriteD
MemtoRegD
MemWriteD
ALUControlD2:0
ALUSrcD
RegDstD
BranchD
RegWriteE
MemtoRegE
MemWriteE
ALUControlE2:0
ALUSrcE
RegDstE
BranchE
RegWriteM
MemtoRegM
MemWriteM
BranchM
PCSrcM
RegWriteW
MemtoRegW
PC'
PCF
A
RD
Instruction Memory
InstrD
25:21
20:16
15:11
15:0
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
RsD
RtD
RdD
RsE
RtE
RdE
Sign Extend
SignImmD
SignImmE
SrcAE
SrcBE
WriteDataE
ALU
ZeroM
ALUOutM
WriteDataM
WriteRegE4:0
WriteRegM4:0
WriteRegW4:0
<<2
PCPlus4F
PCPlus4D
PCPlus4E
PCBranchM
4
Data Memory
WE
WD
ReadDataW
ALUOutW
ResultW
00
01
10
ForwardAE
ForwardBE
RegWriteM
RegWriteW
Hazard Unit

Hazard Unit

What conditions require forwarding?

- Forwarding needed when instruction in Execute stage has a source register matching the destination register of an instruction in the Memory or Writeback stage

Should forward if that stage will write a dest register and dest register matches the source register.

- $0 is hardwired, so it shouldn't be forwarded.
- If both Mem and writeback contain matching registers, memory should have priority

Why?

```
if      ((rsE != 0) AND (rsE == WriteRegM) AND RegWriteM) then
                                           ForwardAE = 10
else if ((rsE != 0) AND (rsE == WriteRegW) AND RegWriteW) then
                                           ForwardAE = 01
else                                       ForwardAE = 00
```

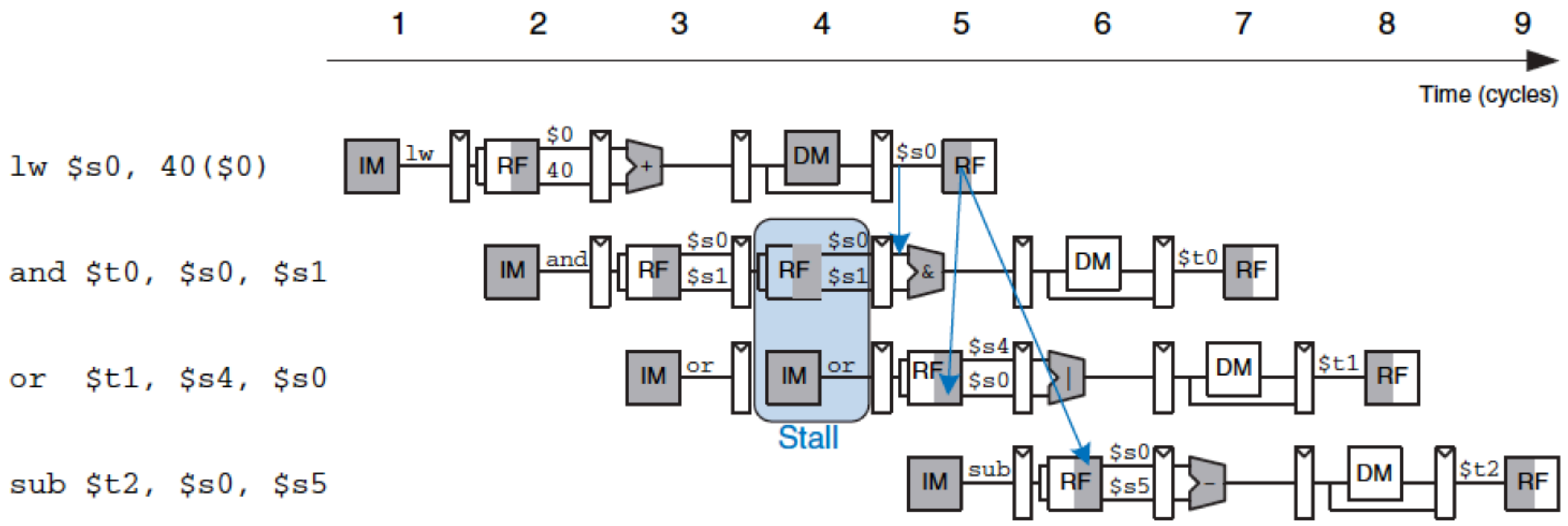
Solving Data Hazards with Stalls

Forwarding solves RAW data hazards when result is computed in the Execute stage

lw doesn't finish reading data until the end of the Memory stage

- Results cannot be forwarded to the Execute of the next instruction.
- Time must pass before the result is ready.
- lw has two cycle latency

Dependent instruction cannot use the result until two cycles later.

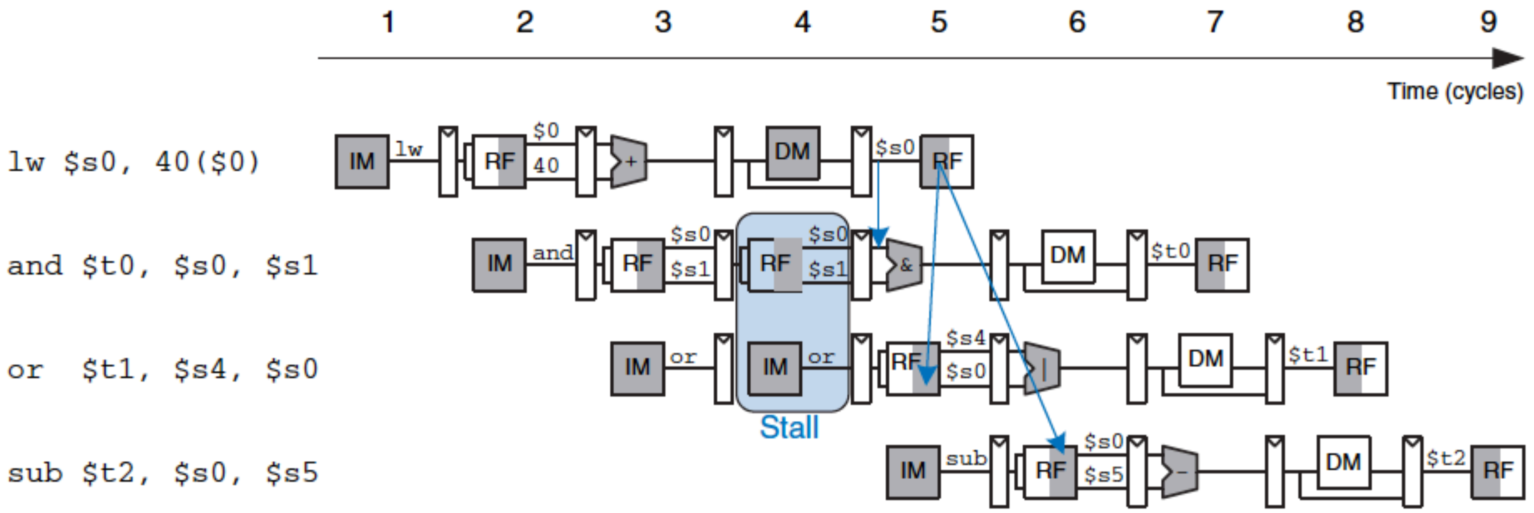

1
2
3
4
5
6
7
8
9
Time (cycles)
lw $s0, 40($0)
and $t0, $s0, $s1
or $t1, $s4, $s0
sub $t2, $s0, $s5
IM
lw
RF
$0
40
+
DM
$s0
RF
and
$s0
$s1
$s0
$s1
&
$t0
or
$s4
$s0
|
$t1
sub
$s5
-
$t2
Stall

Stalling

When a stage is stalled, all preceding stages must be stalled.

- Otherwise we will lose subsequent instructions.

Pipeline register directly after stalled stage must be cleared to prevent false information from propagating forward.



Stall the pipeline, holding up the operation until data is available.

And stalls in the Decode stage

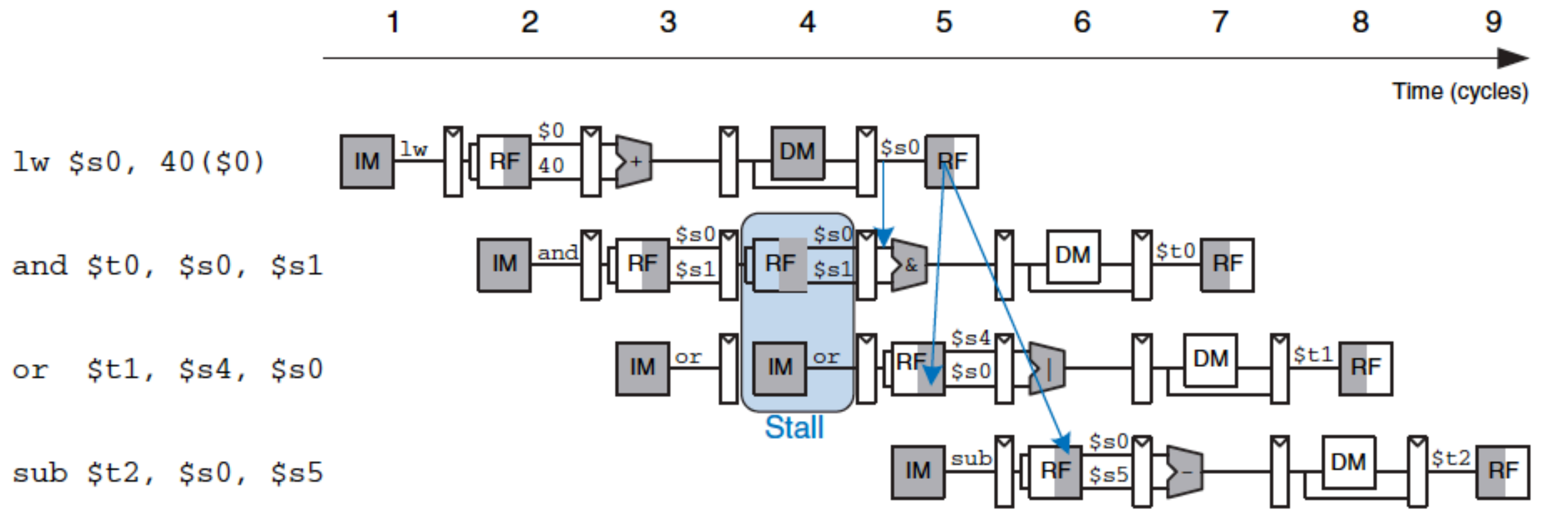
- Enters in cycle 3 and stalls through cycle 4
- Or must also stall in the Fetch stage as only on instruction can be in the stage at one time.

Notice that there will be an empty stage at cycle 4, 5, and 6.

- This "bubble" is essentially a nop instruction.

Control signals for these stages are set to 0 resulting in a nop.

- Stalling is performed by disabling the pipeline register for that stage
- Contents do not change.

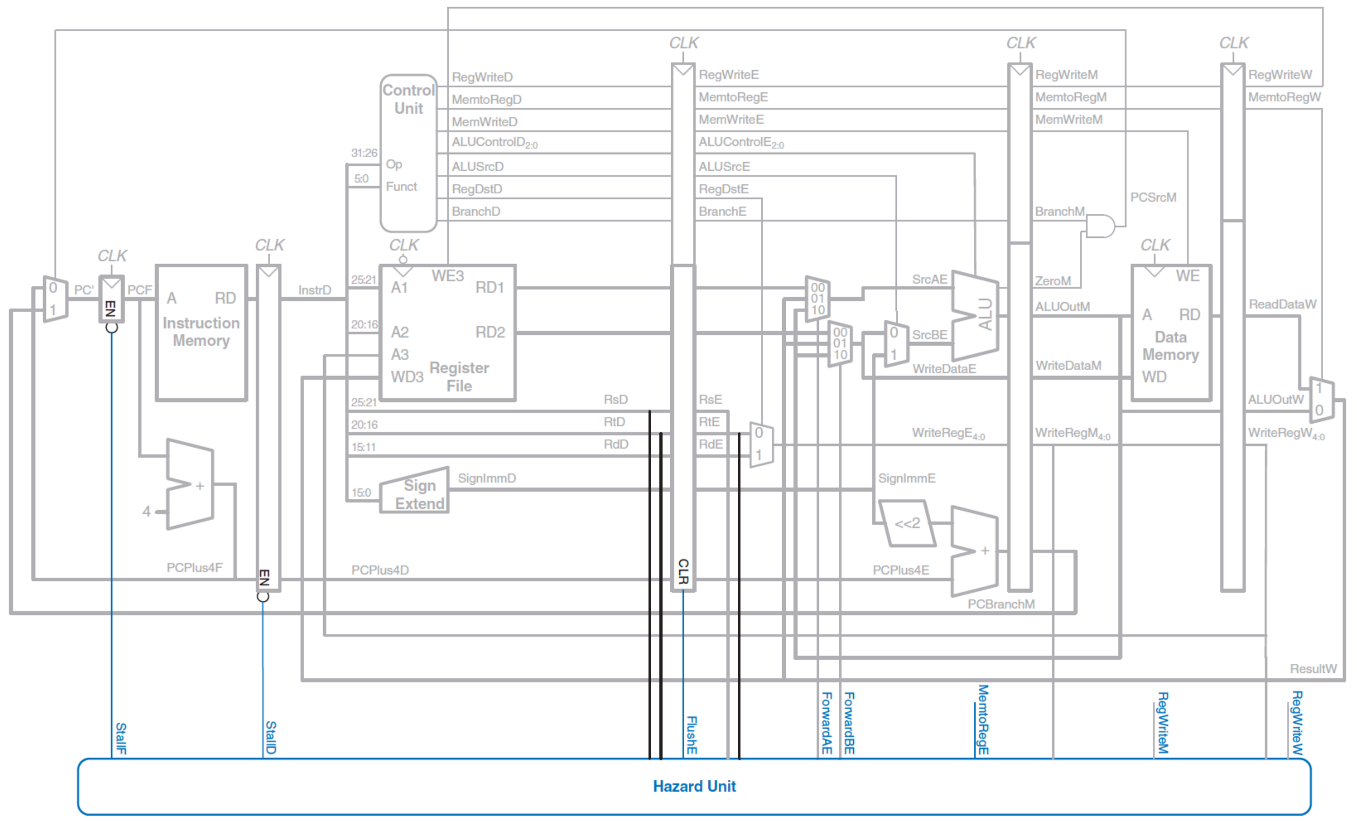


Stalling CPU Modification

Hazard unit examines instruction in the execute stage, if lw and dest matches source of the Decode, instruction must be stalled until source operand is ready

Stalls supported by adding enable inputs to the fetch and decode pipeline registers and a synchronous reset/clear to the execute pipeline register.

- StallD and StallF asserted during a stall.
- FlushE also asserted.


CLK
Control Unit
Op
Funct
RegWriteD
MemtoRegD
MemWriteD
ALUControlD2:0
ALUSrcD
RegDstD
BranchD
RegWriteE
MemtoRegE
MemWriteE
ALUControlE2:0
ALUSrcE
RegDstE
BranchE
RegWriteM
MemtoRegM
MemWriteM
BranchM
PCSrcM
RegWriteW
MemtoRegW
PC'
PCF
EN
A RD
Instruction Memory
InstrD
WE3
A1 RD1
A2 RD2
A3
WD3
Register File
RsD
RtD
RdD
RsE
RtE
RdE
Sign Extend
SignImmD
SignImmE
SrcAE
SrcBE
ALU
ZeroM
ALUOutM
WriteDataE
WriteDataM
WE
A RD
Data Memory
WD
ReadDataW
ALUOutW
WriteRegE4:0
WriteRegM4:0
WriteRegW4:0
<<2
PCPlus4F
PCPlus4D
PCPlus4E
PCBranchM
CLR
ResultW
StallF
StallD
FlushE
ForwardAE
ForwardBE
MemtoRegE
RegWriteM
RegWriteW
Hazard Unit