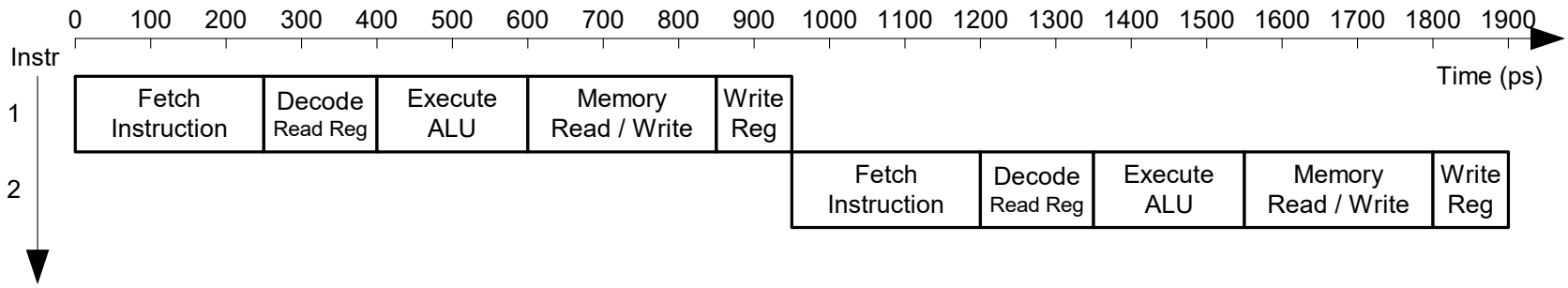
Pipelined MIPS Processor

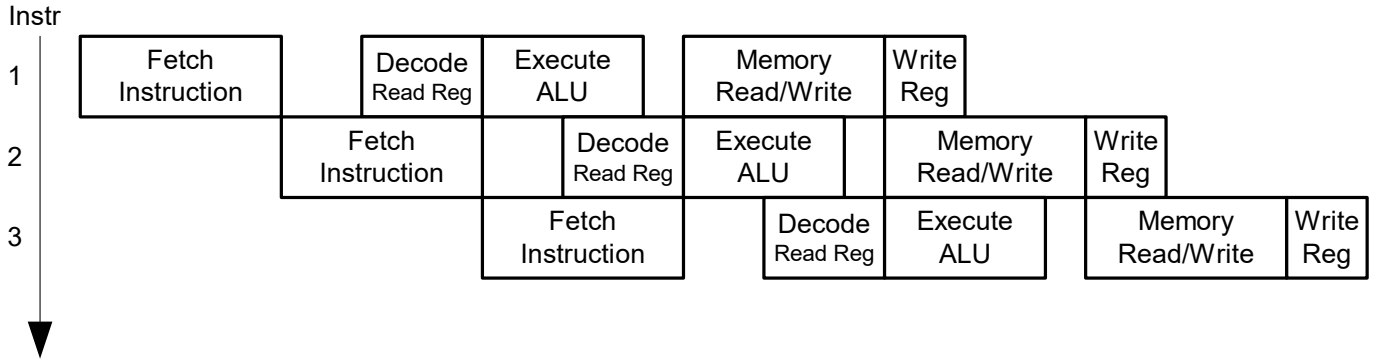
- Temporal parallelism
- Divide single-cycle processor into 5 stages:
  - Fetch
  - Decode
  - Execute
  - Memory
  - Writeback
- Add pipeline registers between stages

Single-Cycle vs. Pipelined

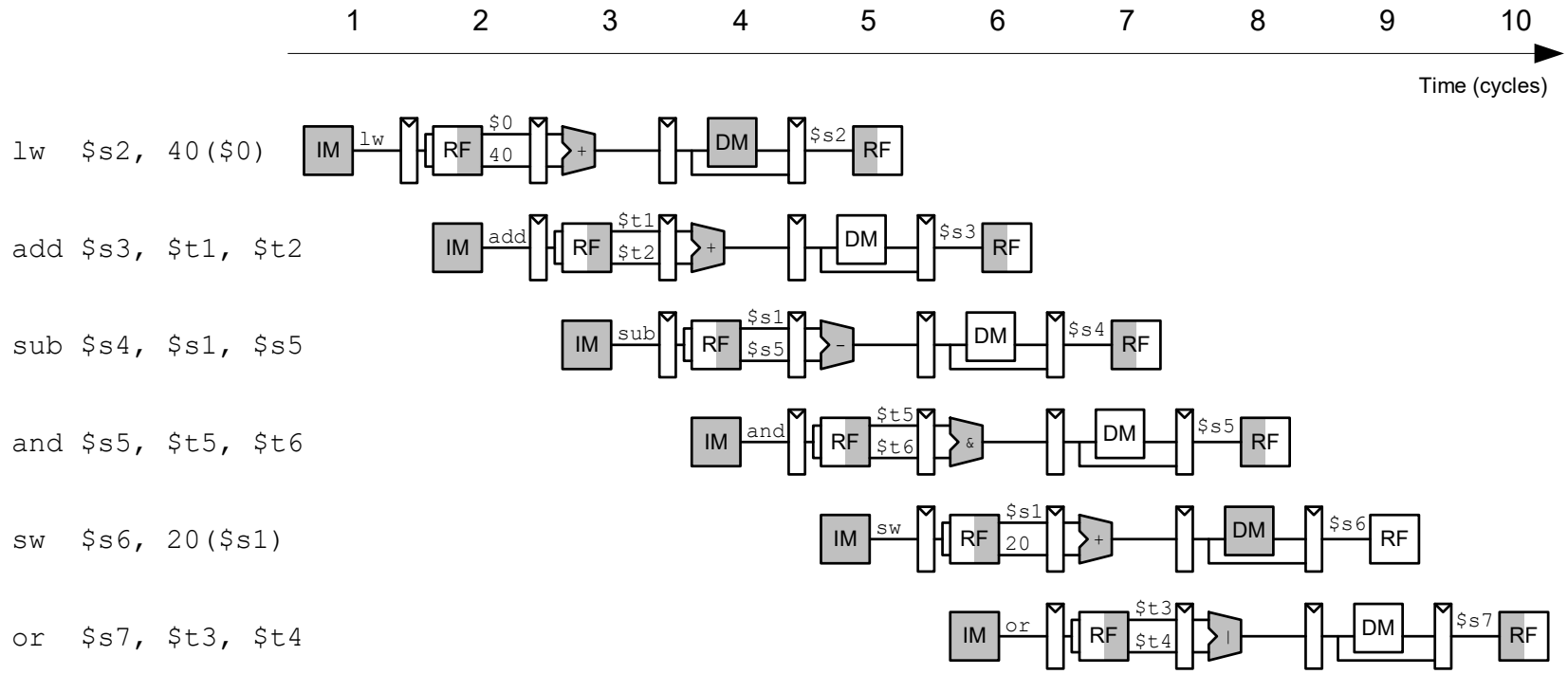
Single-Cycle



Pipelined



Pipelined Processor Abstraction



Pipelined & RTL

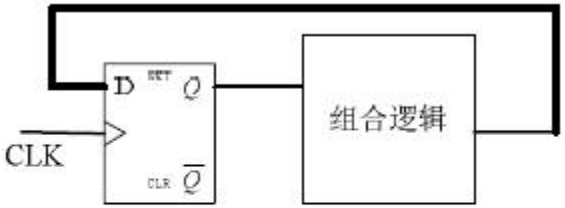


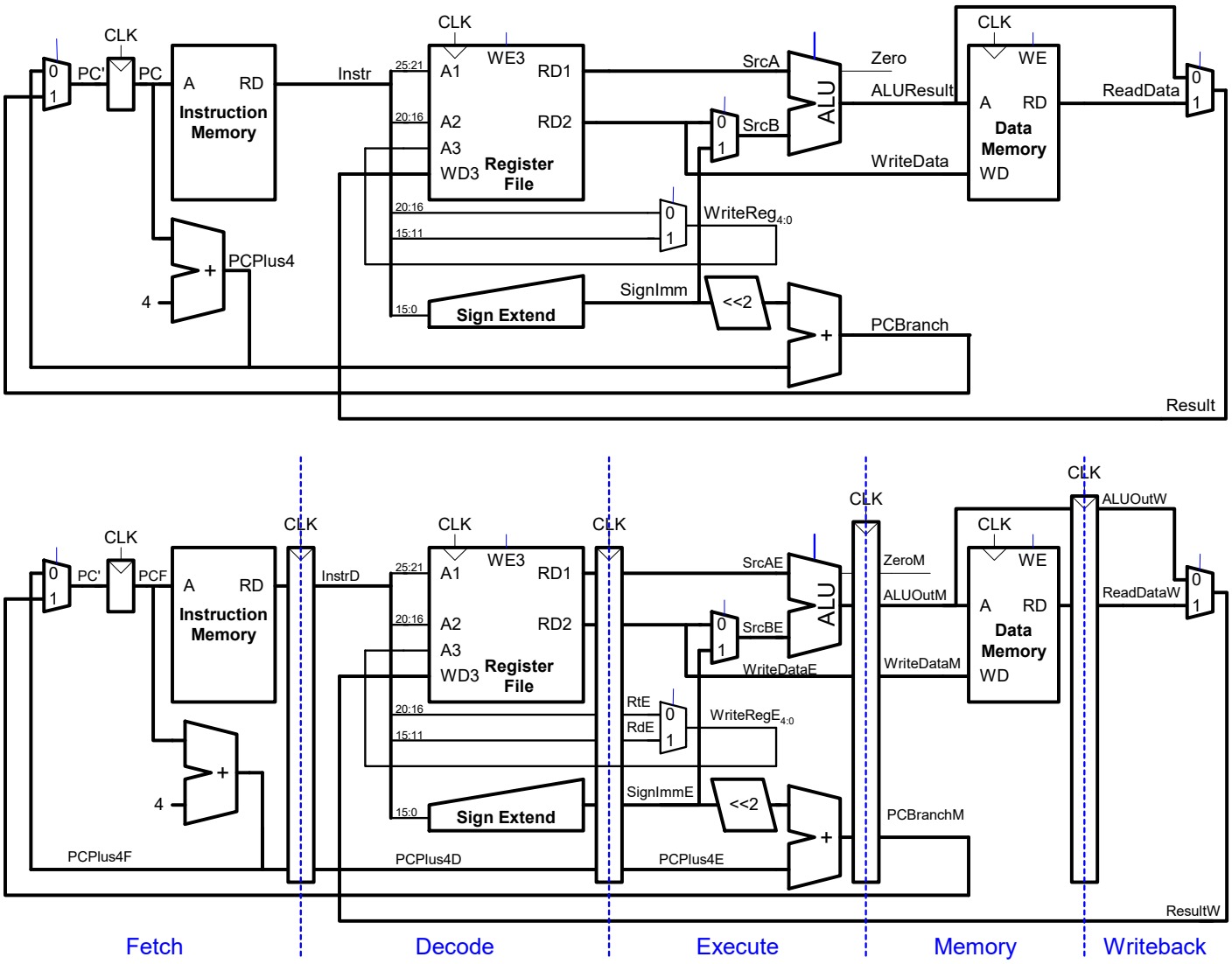
图 4-2 状态机的简单模型

图 4-3 流水线的简单模型

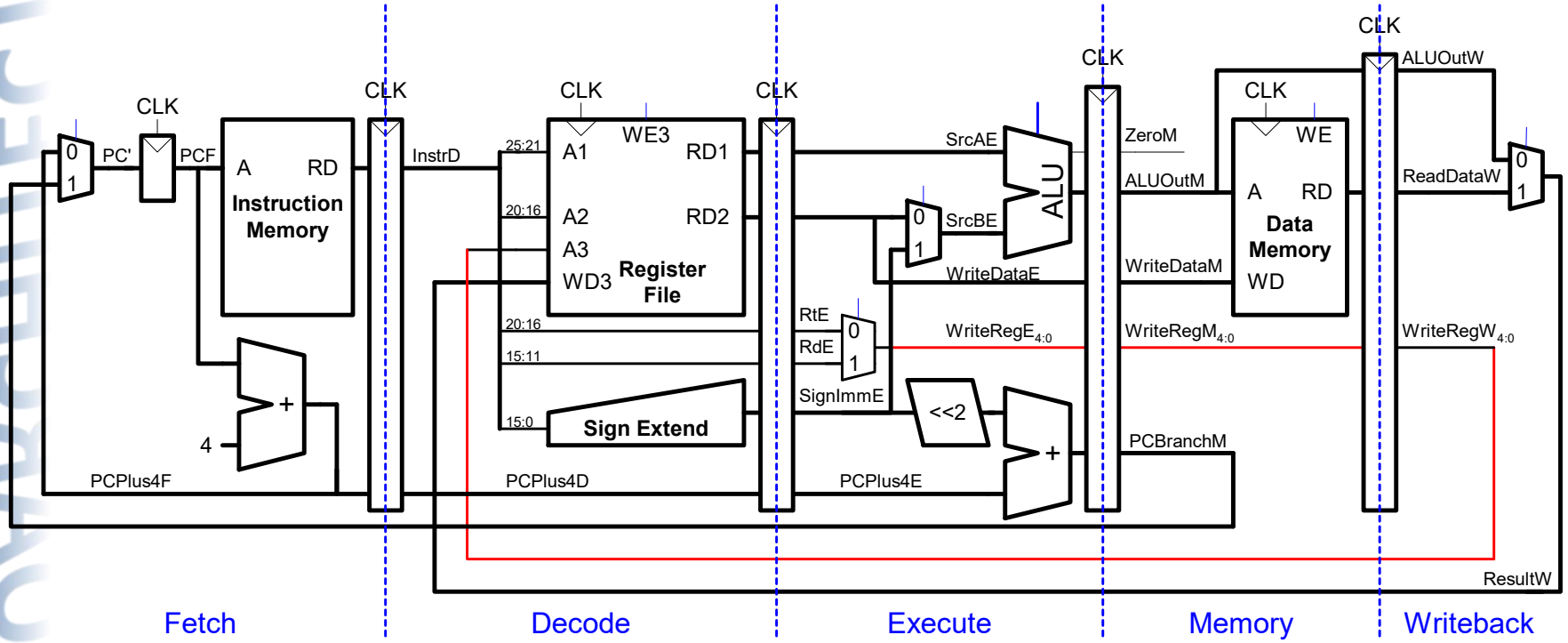
RTL: Register Transfer Level

The signal translate between registers, and every stage occurs their change for combined logic

Single-Cycle & Pipelined Datapath

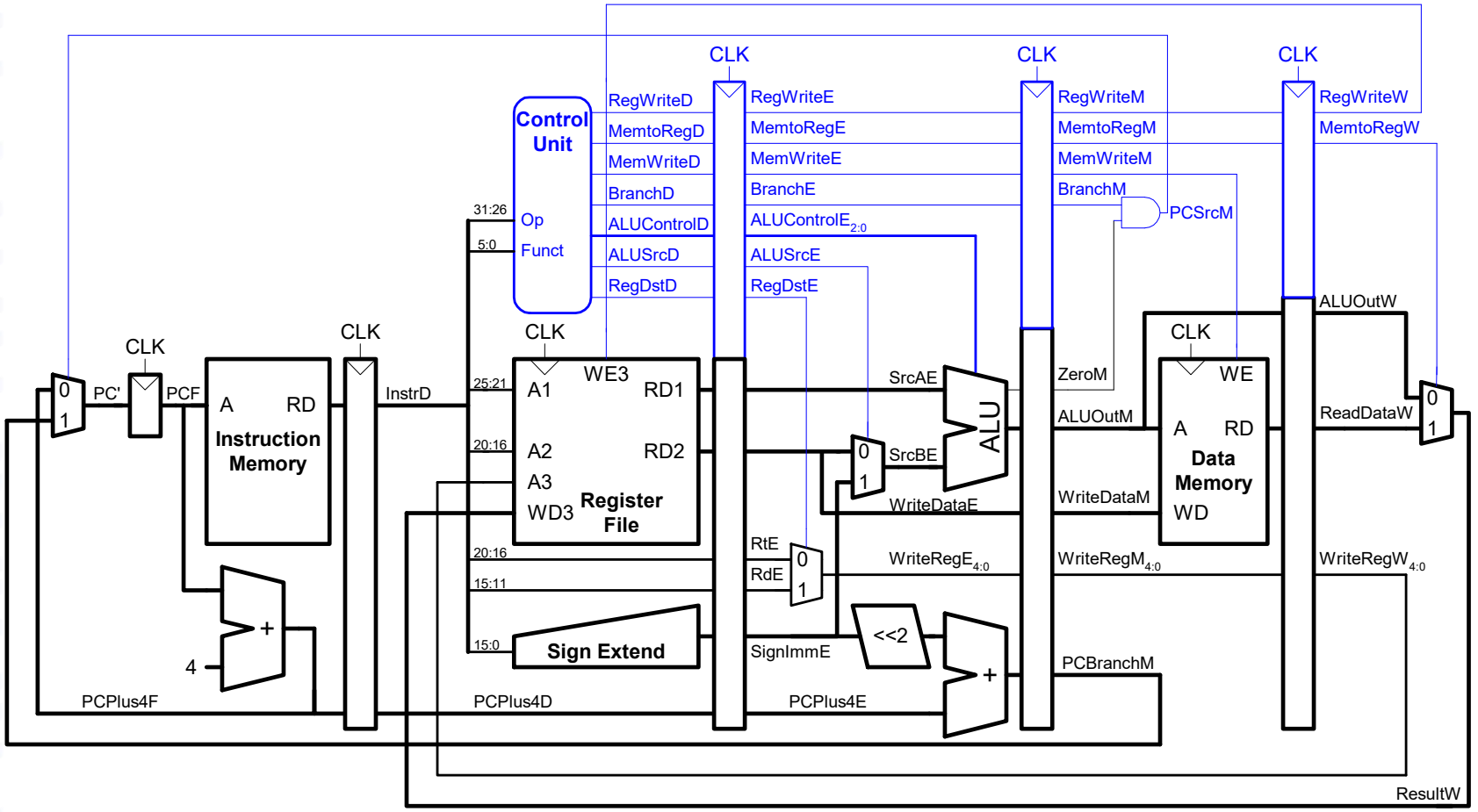


Corrected Pipelined Datapath



WriteReg* must arrive at same time as *Result

Pipelined Processor Control



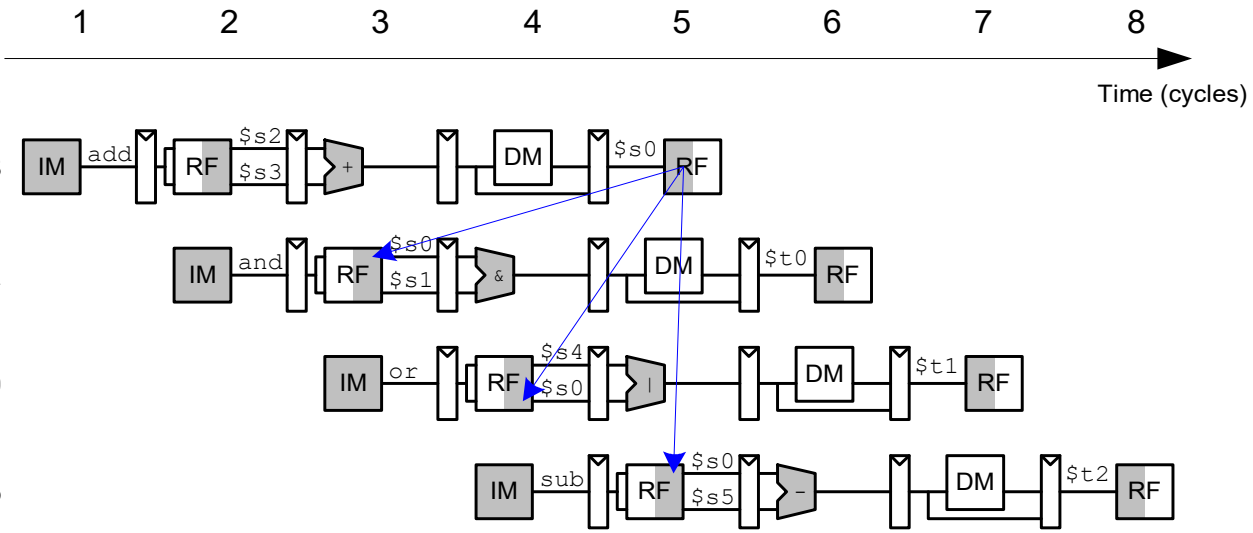
- **Same control unit as single-cycle processor**
- **Control delayed to proper pipeline stage**

Pipeline Hazards

- When an instruction depends on result from instruction that hasn't completed
- Types:
  - **Structural hazard:** We have already solve it by using Harvard architecture. (depart the instruction memory and data memory into two pieces)
  - **Data hazard:** register value not yet written back to register file
  - **Control hazard:** next instruction not decided yet (caused by branches)

Data Hazard

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

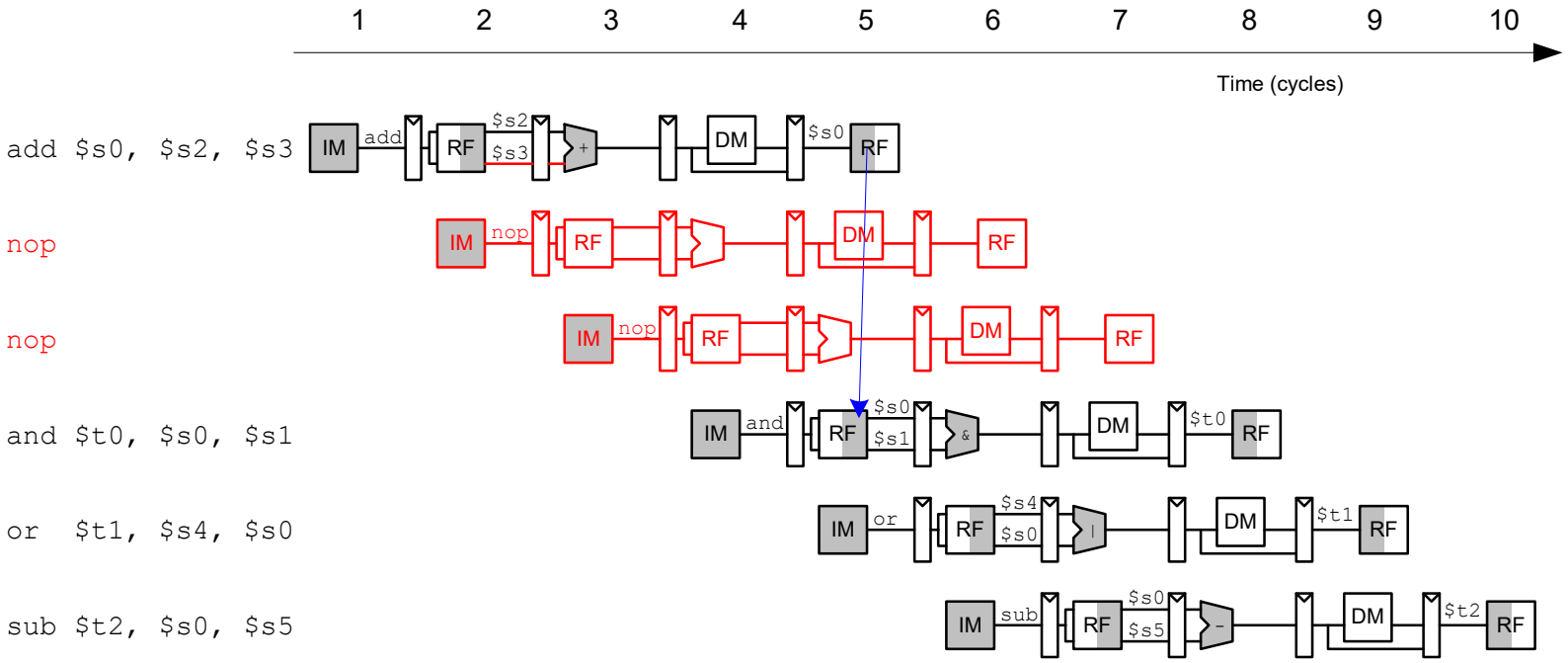


Handling Data Hazards

- Insert `nop`s in code at compile time
- Rearrange code at compile time
- Forward data at run time
- Stall the processor at run time

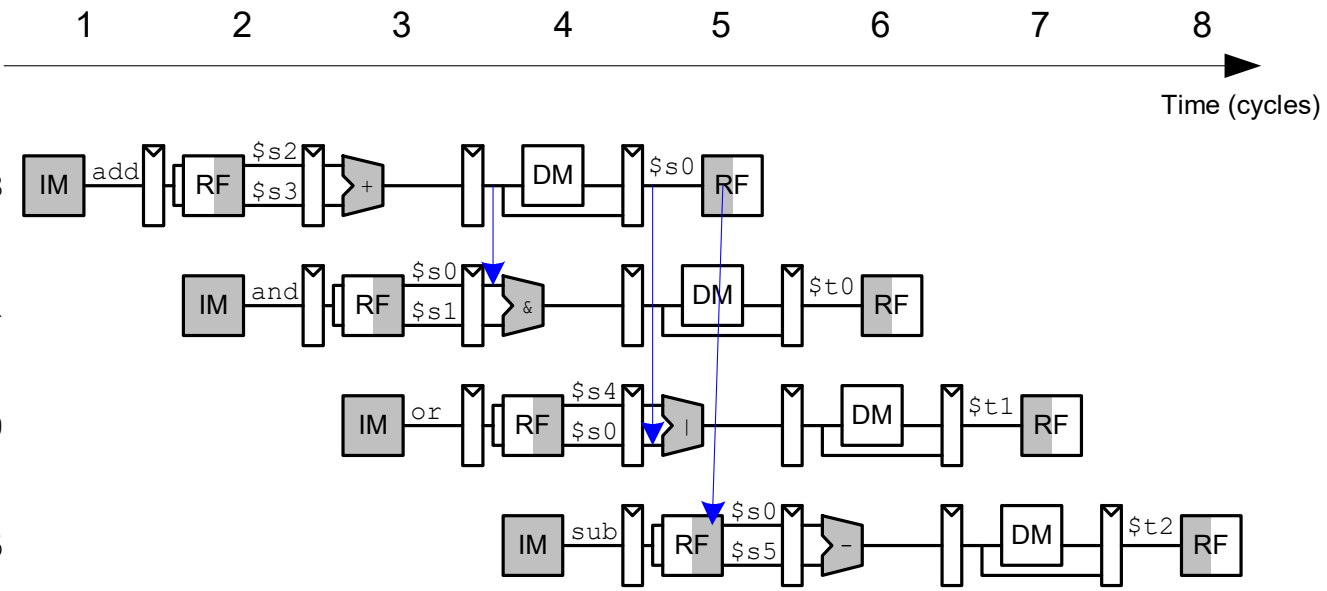
Compile-Time Hazard Elimination

- Insert enough `nop`s for result to be ready
- Or move independent useful instructions forward

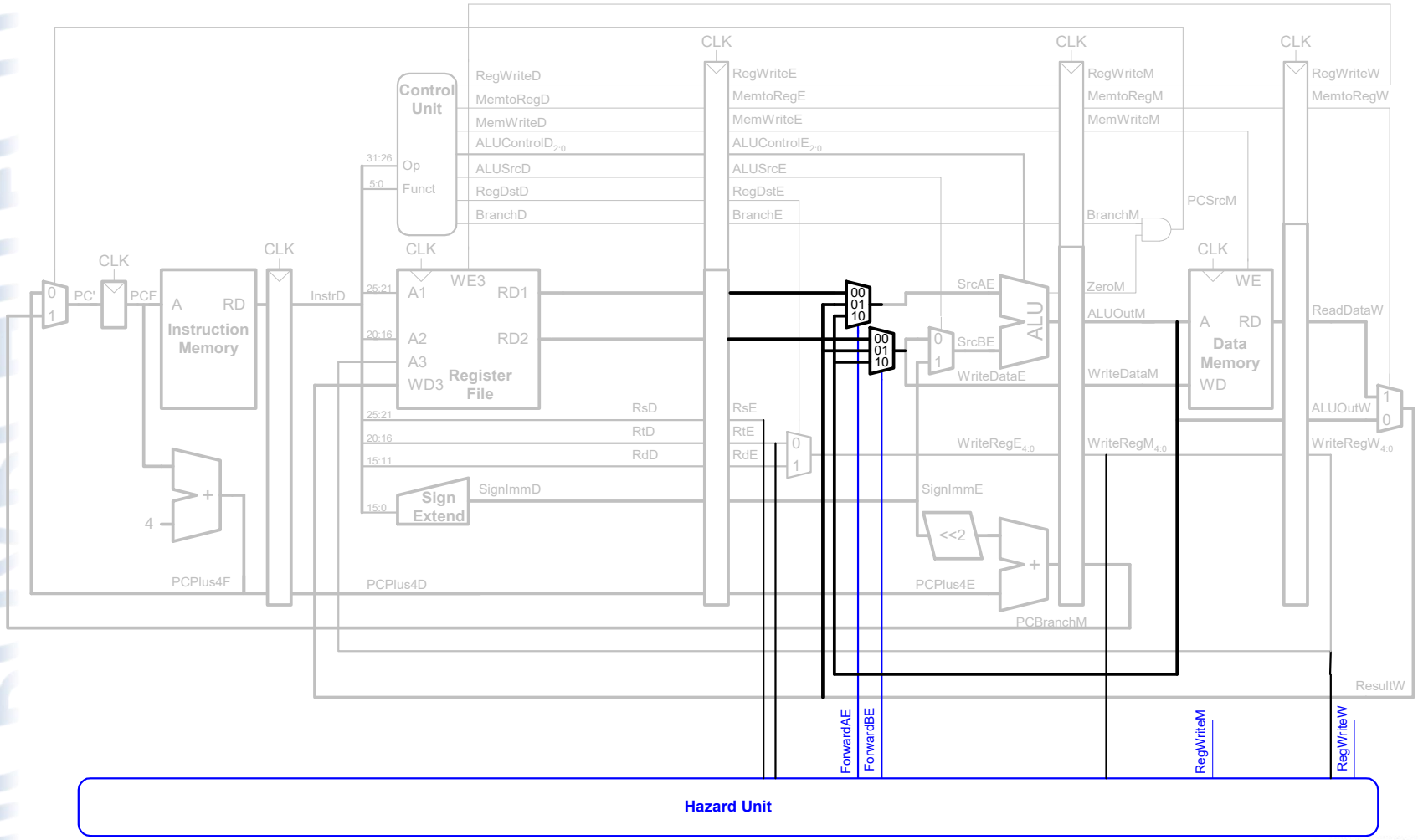


Data Forwarding

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



Data Forwarding



Data Forwarding

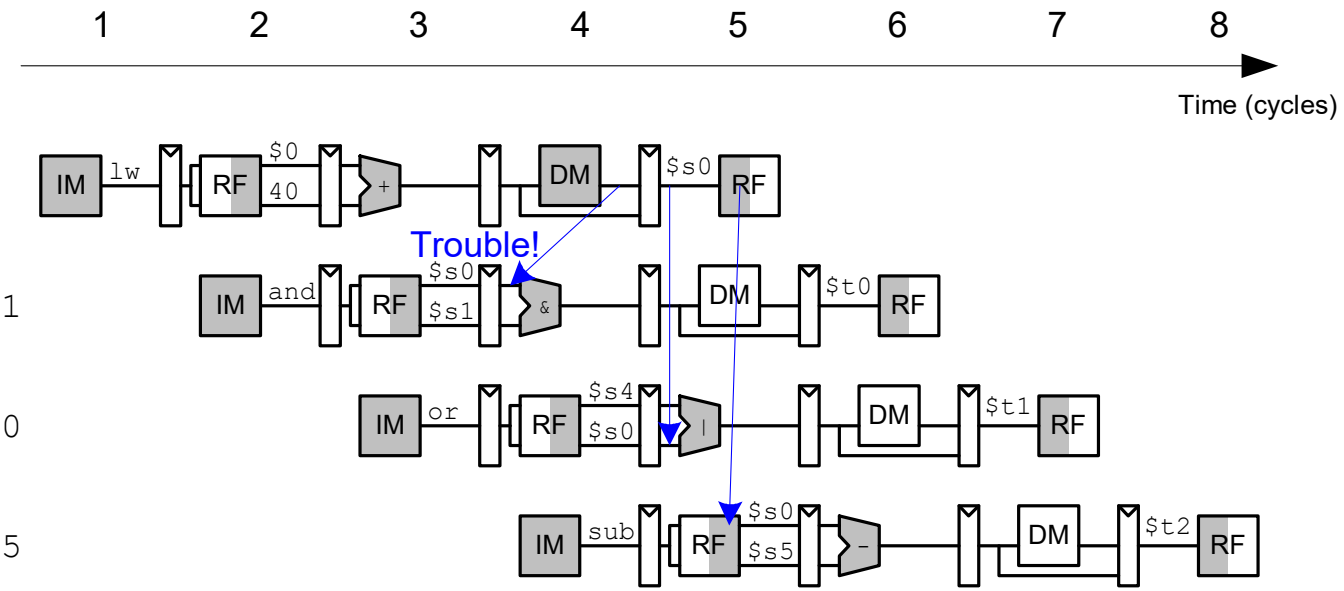
- Forward to Execute stage from either:
  - Memory stage or
  - Writeback stage
- Forwarding logic for *ForwardAE*:

```
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
```

Forwarding logic for *ForwardBE* same, but replace *rsE* with *rtE*

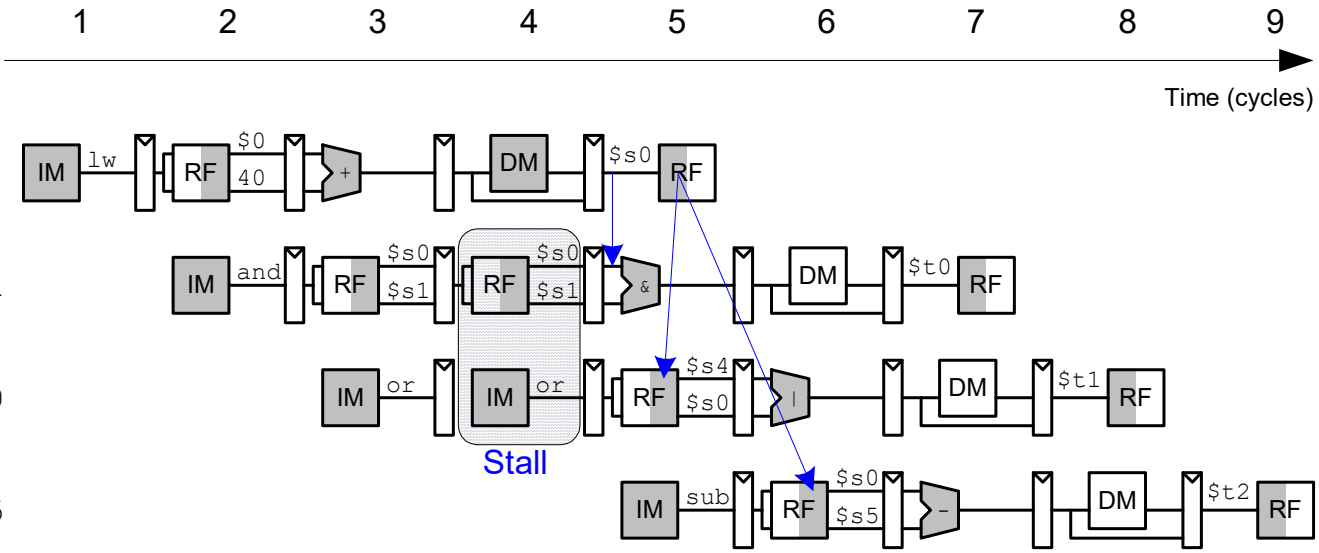
Stalling

```
lw $s0, 40($0)
and $t0, $s0, $s1
or  $t1, $s4, $s0
sub $t2, $s0, $s5
```

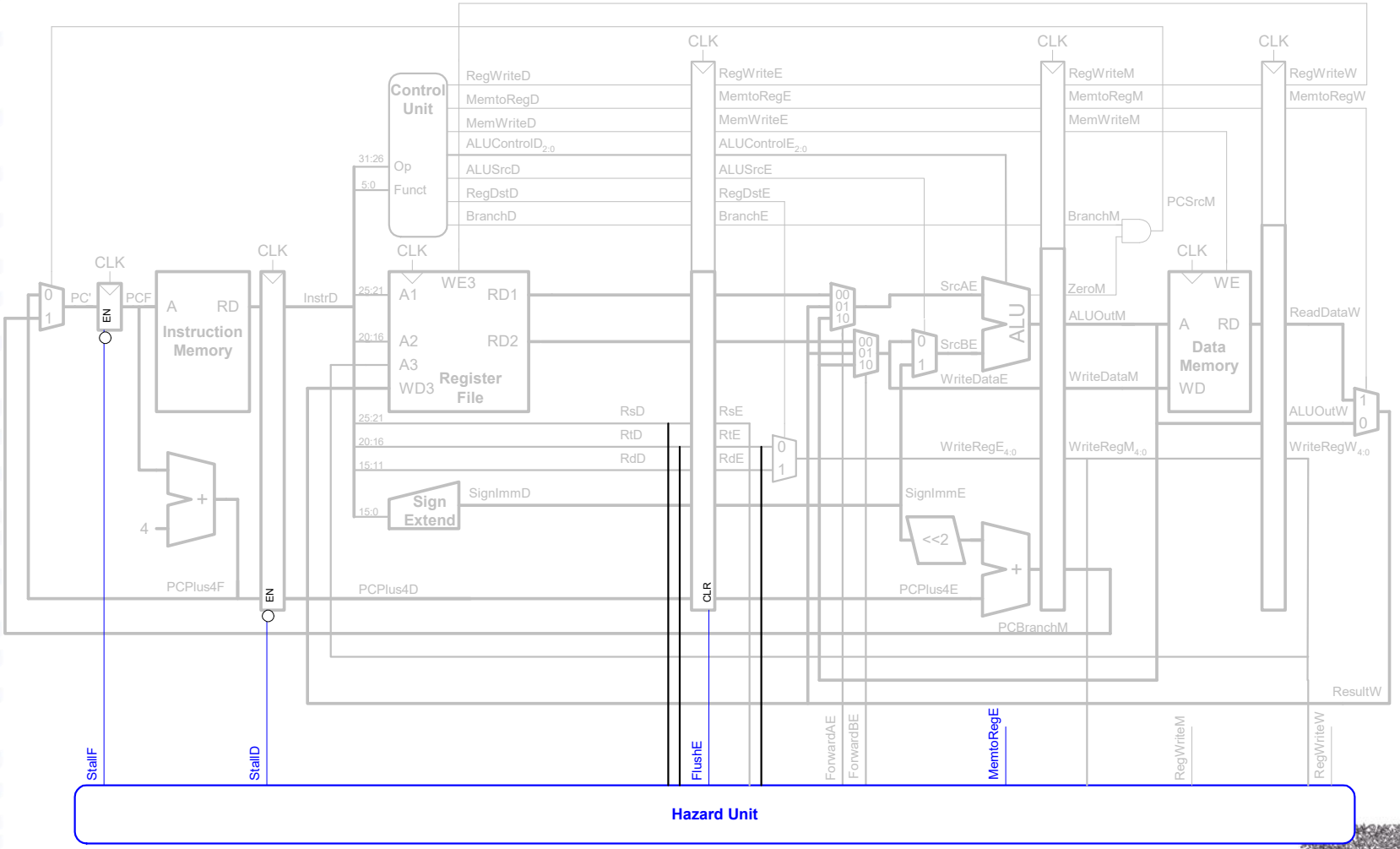


Stalling

```
lw $s0, 40($0)
and $t0, $s0, $s1
or  $t1, $s4, $s0
sub $t2, $s0, $s5
```



Stalling Hardware



Stalling Logic

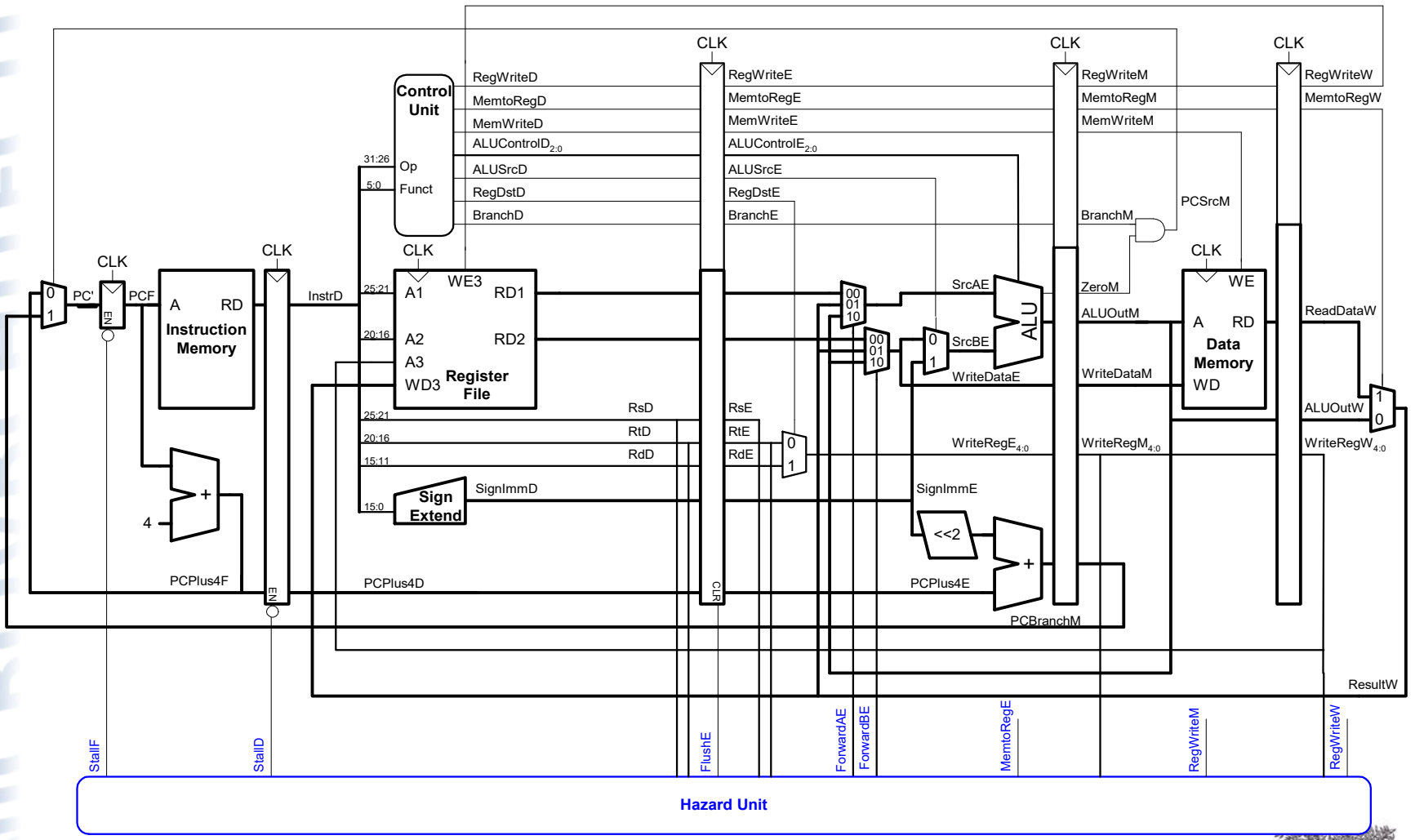
```
lwstall =
   ((rsD==rtE) OR (rtD==rtE)) AND MemtoRegE

StallF = StallD = FlushE = lwstall
```

Control Hazards

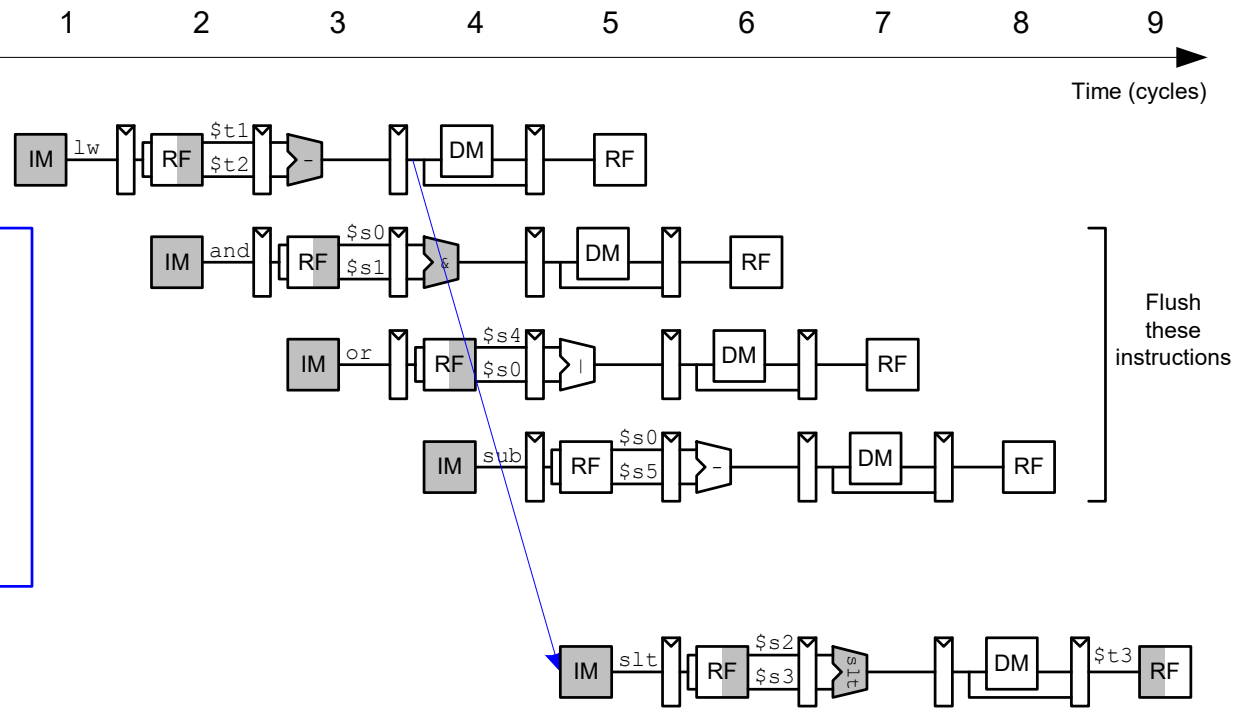
- **`beq`:**
  - branch not determined until 4th stage of pipeline
  - Instructions after branch fetched before branch occurs
  - These instructions must be flushed if branch happens
- **Branch misprediction penalty**
  - number of instruction flushed when branch is taken
  - May be reduced by determining branch earlier

Control Hazards: Original Pipeline

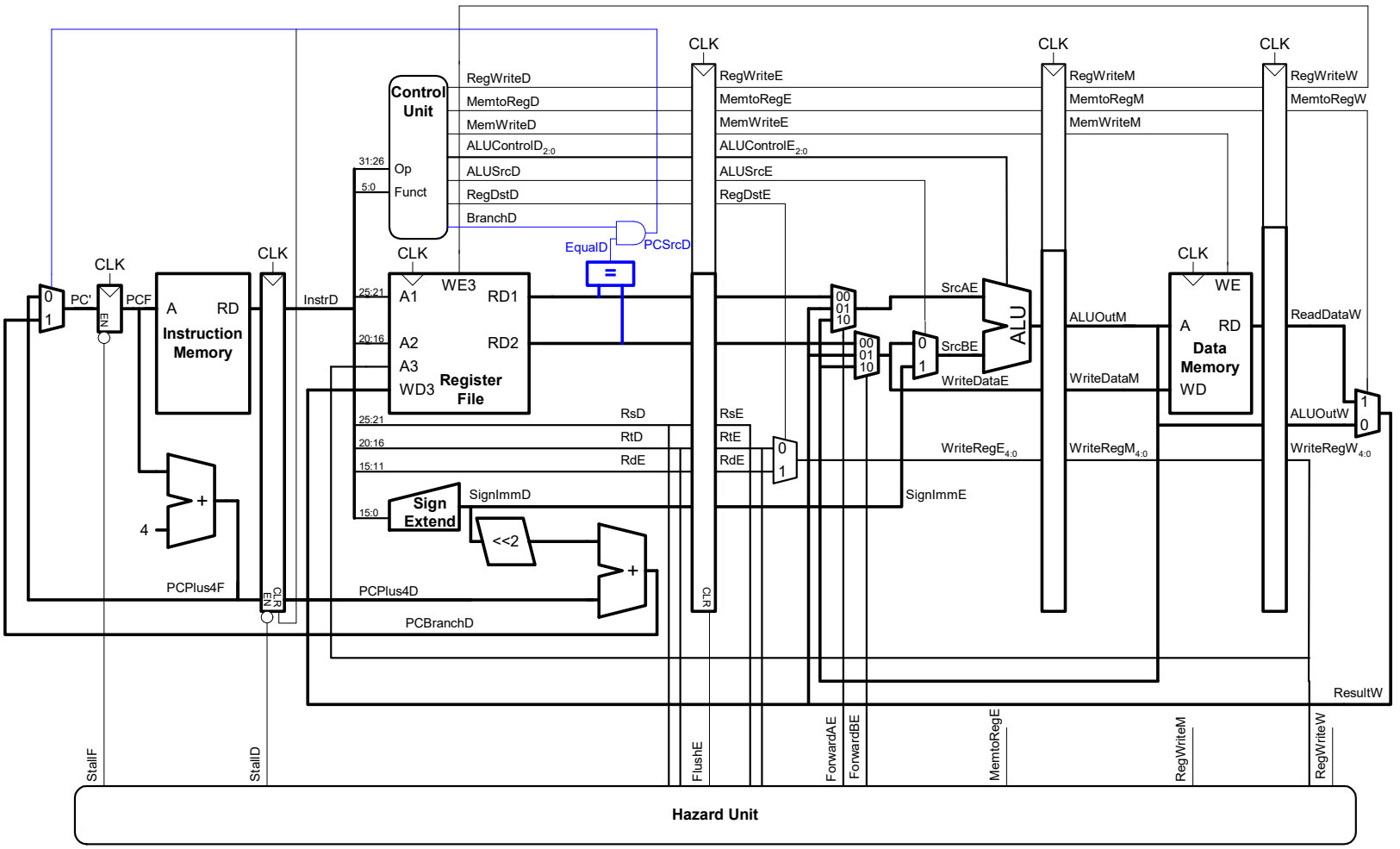


Control Hazards

```
20   beq $t1, $t2, 40
24   and $t0, $s0, $s1
28   or  $t1, $s4, $s0
2C   sub $t2, $s0, $s5
30   ...
...
64   slt $t3, $s2, $s3
```

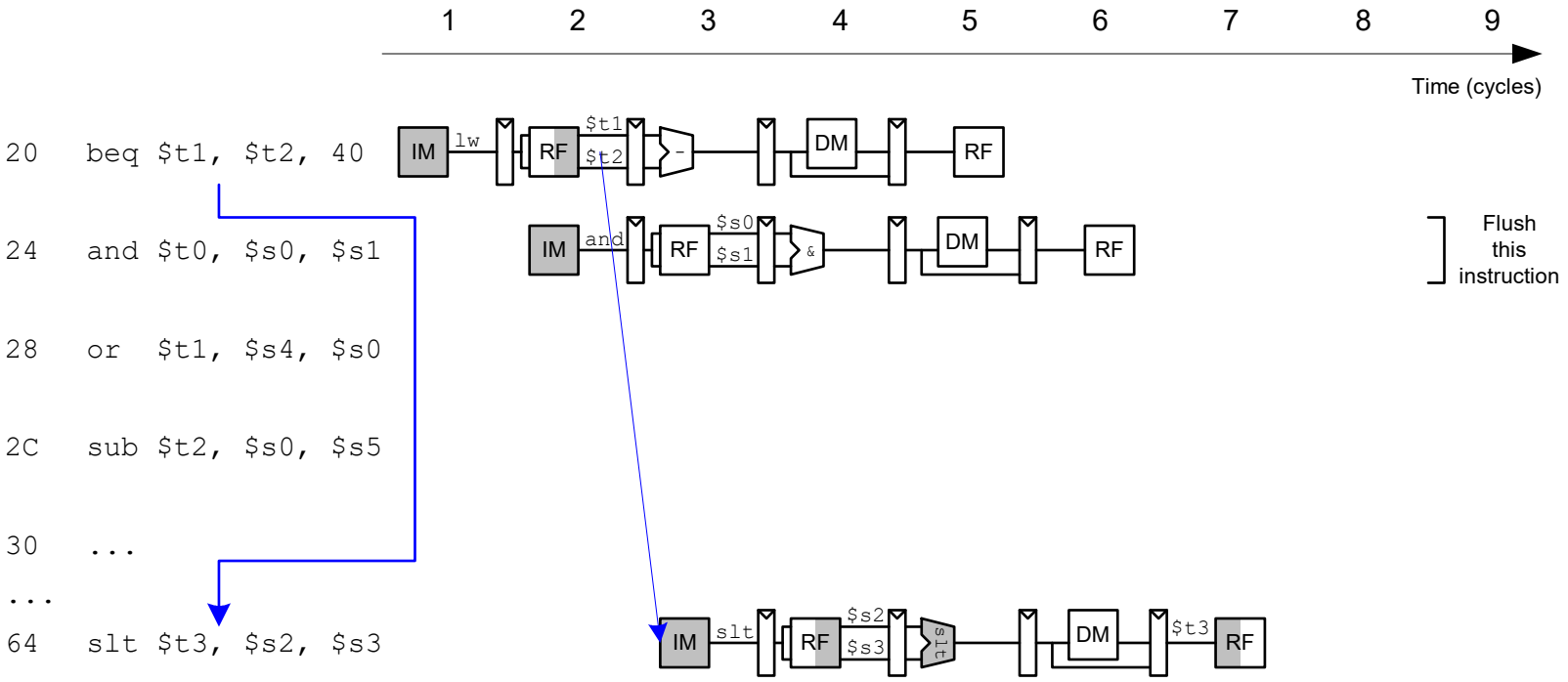


Early Branch Resolution

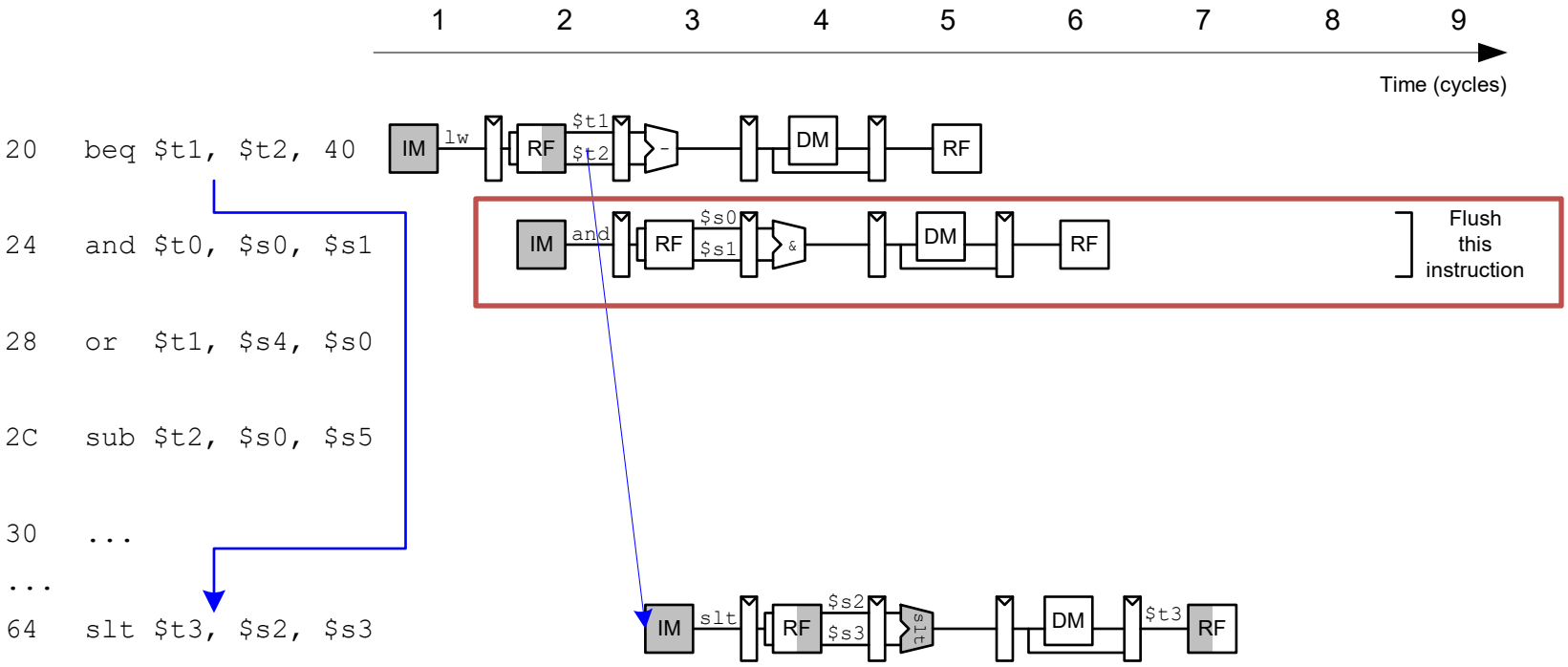


Introduced another data hazard in Decode stage

Early Branch Resolution



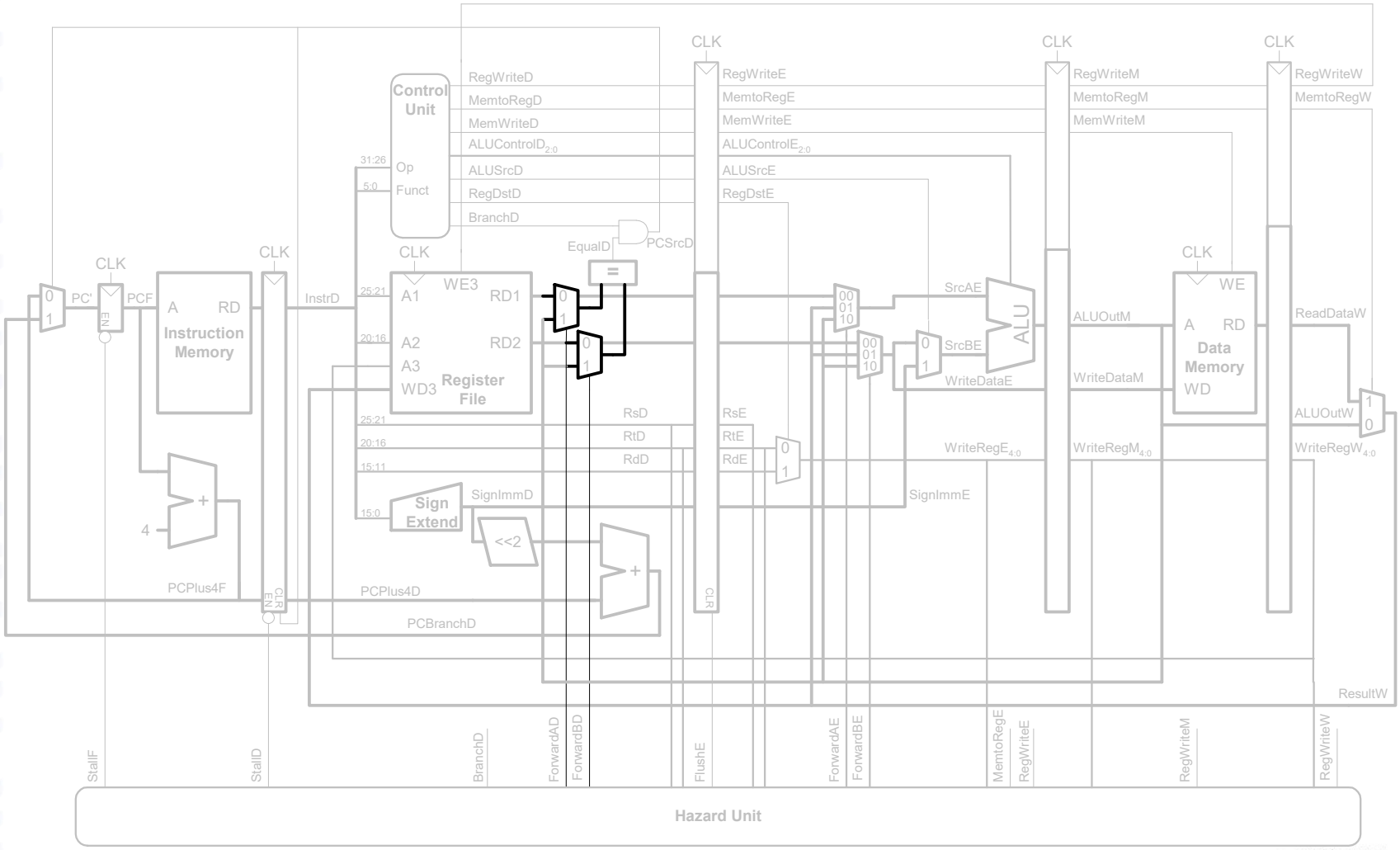
Branch delay slot



Branch delay slot: The instruction after branch always execute.

In this way, the only left instruction could use effectively, instead of wasting this stage

Handling Data & Control Hazards



Control Forwarding & Stalling Logic

- **Forwarding logic:**

```
ForwardAD = (rsD !=0) AND (rsD == WriteRegM) AND RegWriteM
ForwardBD = (rtD !=0) AND (rtD == WriteRegM) AND RegWriteM
```

- **Stalling logic:**

```
branchstall = BranchD AND RegWriteE AND
                (WriteRegE == rsD OR WriteRegE == rtD)
              OR
              BranchD AND MemtoRegM AND
                (WriteRegM == rsD OR WriteRegM == rtD)

StallF = StallD = FlushE = lwstall OR branchstall
```

Branch Prediction

- Guess whether branch will be taken
  - Backward branches are usually taken (loops)
  - Consider history to improve guess
- Good prediction reduces fraction of branches requiring a flush

Pipelined Performance Example

- SPECINT2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type
- Suppose:
  - 40% of loads used by next instruction
  - 25% of branches mispredicted
  - All jumps flush next instruction
- **What is the average CPI?**

Pipelined Performance Example

- SPECINT2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type
- Suppose:
  - 40% of loads used by next instruction
  - 25% of branches mispredicted
  - All jumps flush next instruction
- **What is the average CPI?**
  - Load/Branch CPI = 1 when no stalling, 2 when stalling
  - $CPI_{lw} = 1(0.6) + 2(0.4) = 1.4$
  - $CPI_{beq} = 1(0.75) + 2(0.25) = 1.25$

Average CPI = $(0.25)(1.4) + (0.1)(1) + (0.11)(1.25) + (0.02)(2) + (0.52)(1)$

$$= \mathbf{1.15}$$

Pipelined Performance

- Pipelined processor critical path:

$$T_c = \max \{$$
$$t_{pcq} + t_{\text{mem}} + t_{\text{setup}}$$
$$2(t_{RFread} + t_{\text{mux}} + t_{\text{eq}} + t_{\text{AND}} + t_{\text{mux}} + t_{\text{setup}})$$
$$t_{pcq} + t_{\text{mux}} + t_{\text{mux}} + t_{\text{ALU}} + t_{\text{setup}}$$
$$t_{pcq} + t_{\text{memwrite}} + t_{\text{setup}}$$
$$2(t_{pcq} + t_{\text{mux}} + t_{\text{RFwrite}}) \}$$

Pipelined Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	t_{RFread}	150
Register file setup	$t_{RFsetup}$	20
Equality comparator	t_{eq}	40
AND gate	t_{AND}	15
Memory write	$T_{memwrite}$	220
Register file write	$t_{RFwrite}$	100 ps

$$T_c = 2(t_{RFread} + t_{mux} + t_{eq} + t_{AND} + t_{mux} + t_{setup})$$
$$= 2[150 + 25 + 40 + 15 + 25 + 20] \text{ ps} = \mathbf{550 \text{ ps}}$$

Pipelined Performance Example

Program with 100 billion instructions

Execution Time = (# instructions) × CPI × T_c

$$= (100 \times 10^9)(1.15)(550 \times 10^{-12})$$

= 63 seconds

Processor Performance Comparison

Processor	Execution Time (seconds)	Speedup (single-cycle as baseline)
Single-cycle	92.5	1
Multicycle	133	0.70
Pipelined	63	1.47