

Fundamentals of Computer Systems

A Pipelined MIPS Processor

Stephen A. Edwards

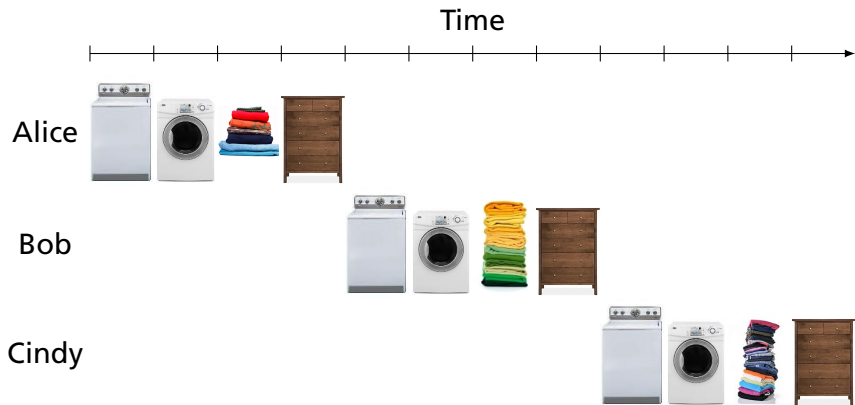
Columbia University

Summer 2021



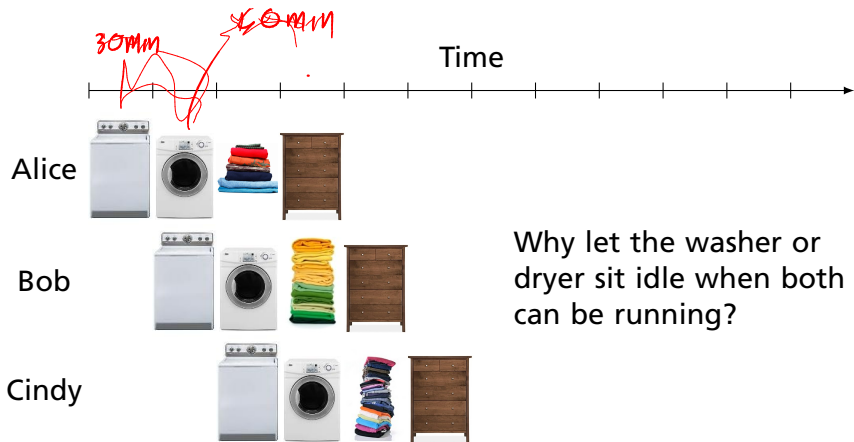
Technical Illustrations Copyright ©2007 Elsevier

Sequential Laundry

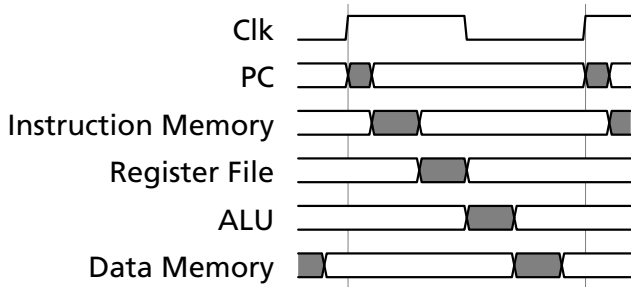
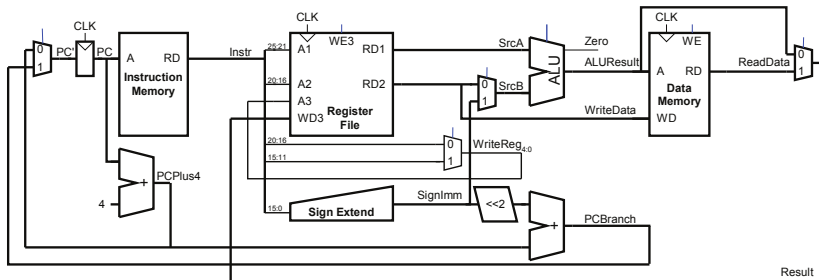


Pipelined Laundry

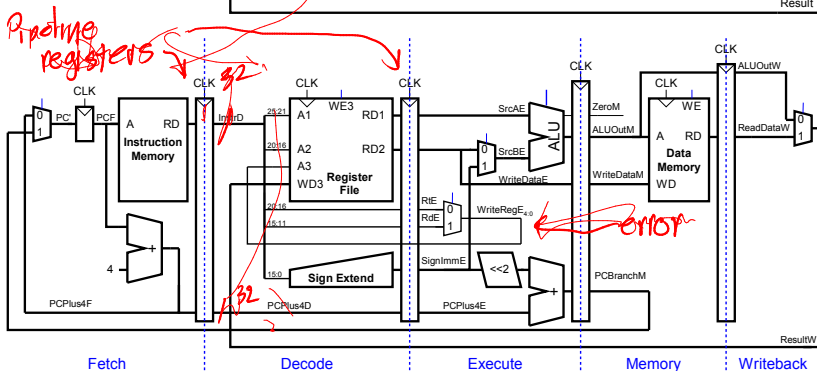
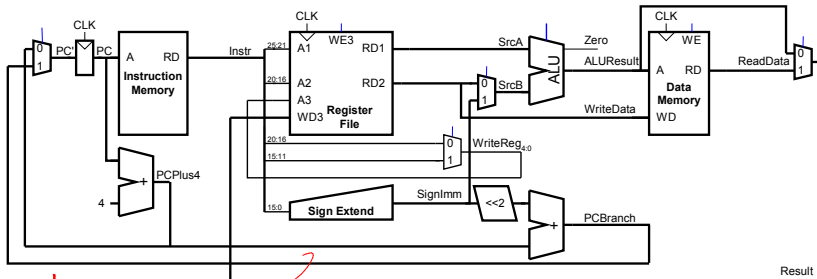
n washers
n dryers



Single-Cycle Datapath Timing

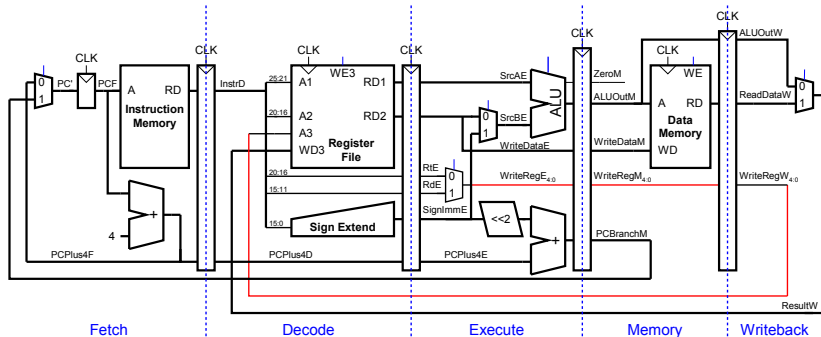


Single-Cycle vs. Pipelined Datapath



Corrected Pipelined Datapath

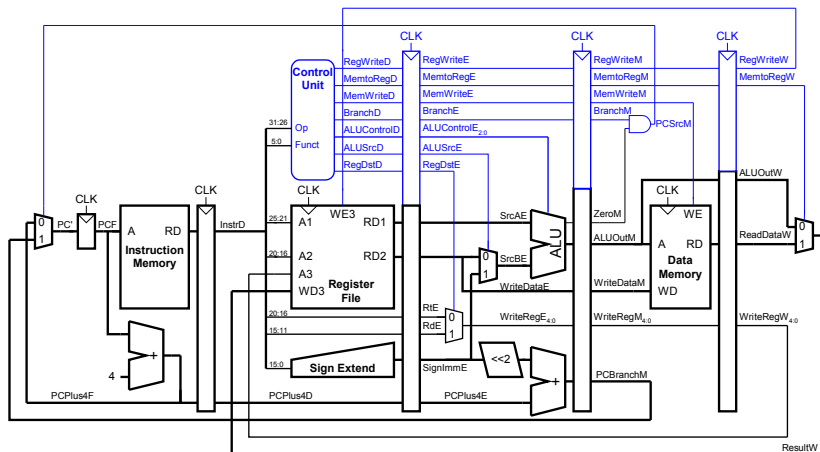
The register number to write (WriteReg) must stay synchronized with the result.



Pipeline Control

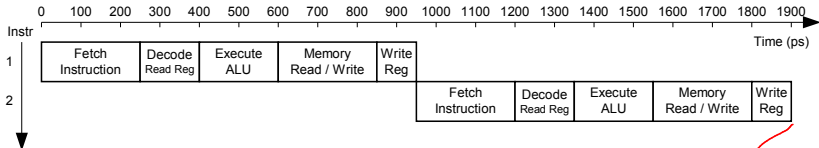
Same control unit as the single-cycle processor;
Control signals delayed across pipeline stages

Decode — Execute

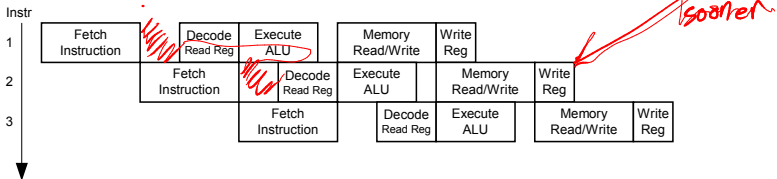


Single-Cycle vs. Pipeline Timing

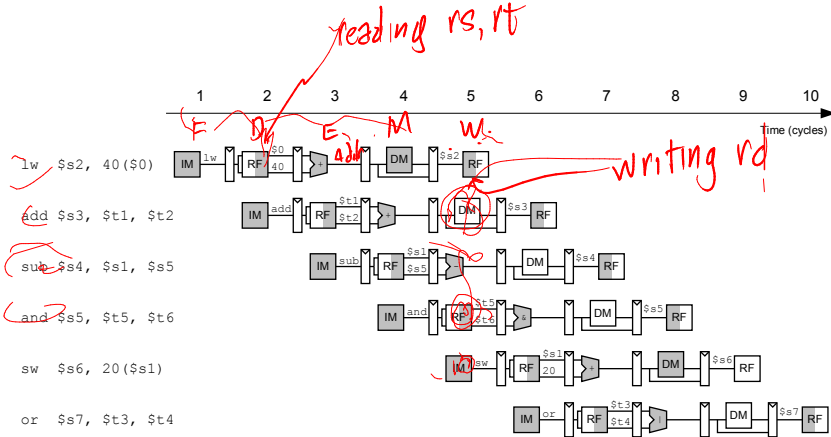
Single-Cycle



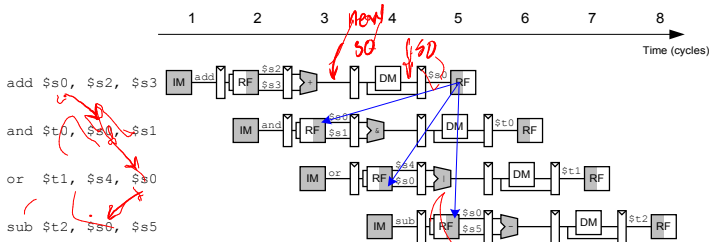
Pipelined



Pipelining Abstraction



Data Hazard



The first instruction produces a result (in \$s0) that later instructions need.

The ALU has computed the value in cycle 3, but it won't be written to the register file until cycle 5.

Probably OK ← register file write & read works



Biohazard



Water Hazard

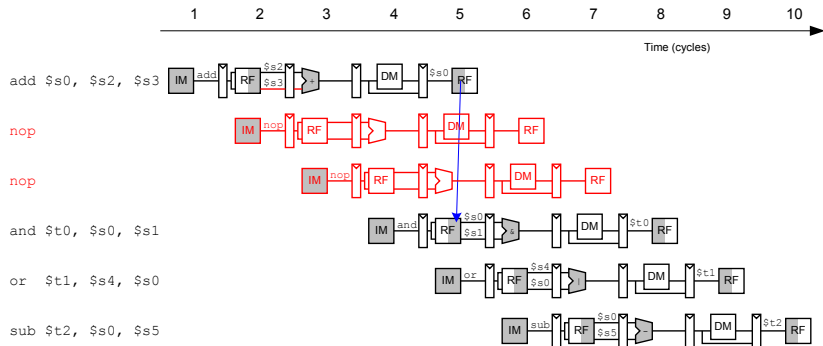


Hazard Lights



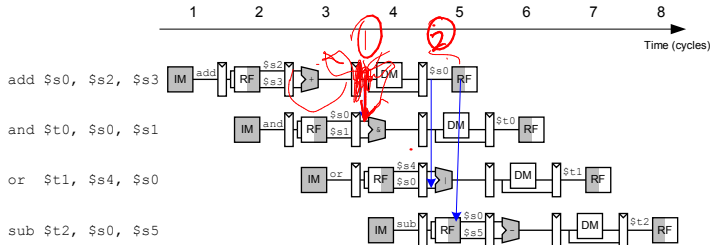
Dukes of Hazard

Eliminating Hazards at Compile-Time



Insert *nops* to delay later instructions; sometimes possible to put useful work in those slots.

Eliminating Data Hazards through Forwarding



Add logic to send data “between” instructions; register file eventually written.

Here, the result is available at the end of cycle 3 and needed in cycles 4 and 5.

① ~~M → E~~

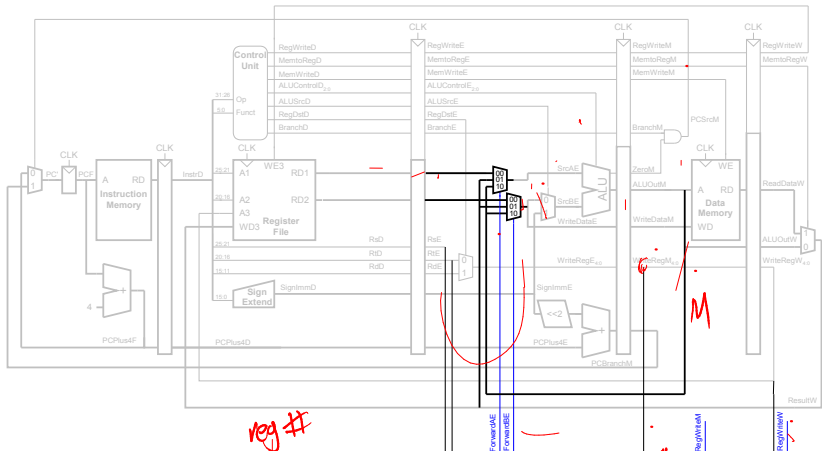
② ~~W → E~~

backwards
time
travel

Datapath with Data Forwarding

E

M



reg #

Hazard Unit

the same reg

going to update

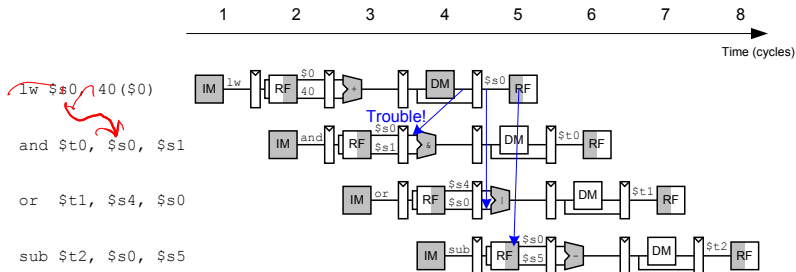
ForwardAE = $\begin{cases} 10 & \text{if } rsE \neq 0 \wedge rsE = WriteRegM \wedge RegWriteM, \\ 01 & \text{if } rsE \neq 0 \wedge rsE = WriteRegW \wedge RegWriteW, \\ 00 & \text{otherwise.} \end{cases}$

M stage
W stage

pipeline reg # register file

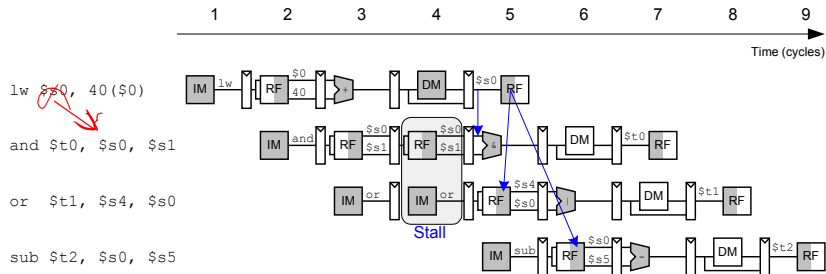
First ALU operand

Data Hazard that Demands a Stall



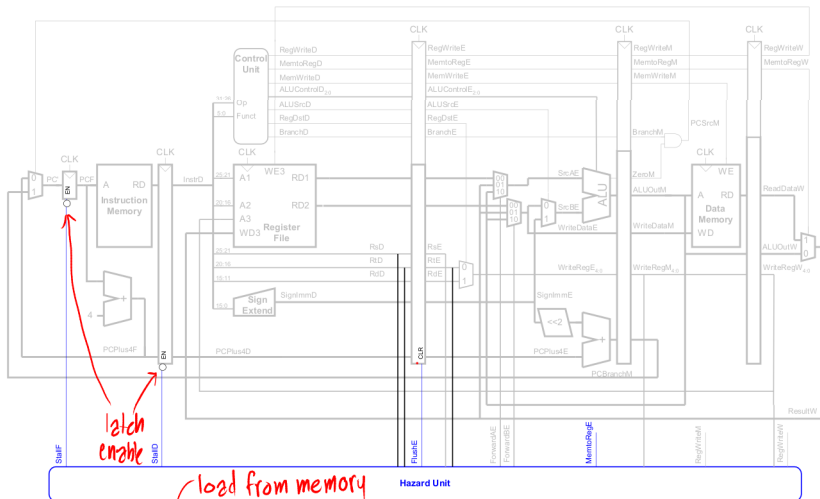
This data hazard can't be solved with forwarding because the value is only available at the end of cycle 4, yet is needed at the beginning.

Stalling a Pipeline



A *stall* tells an instruction to wait for a cycle before proceeding.

Stalling Hardware

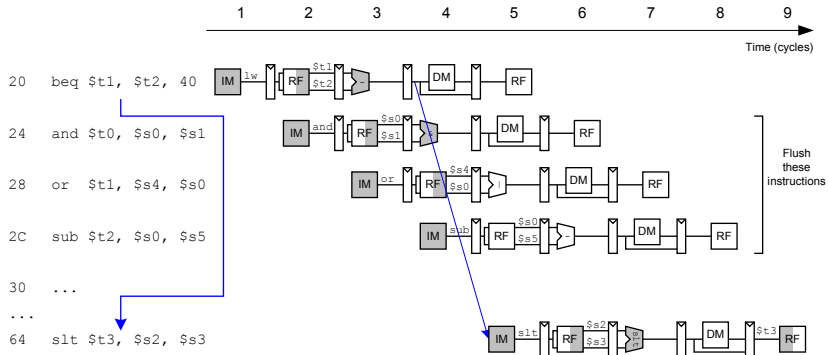


$$lwstall = MemToRegE \wedge ((rsD = rtE) \vee (rtD = rtE))$$

$$StallF, StallD, FlushE = lwstall$$

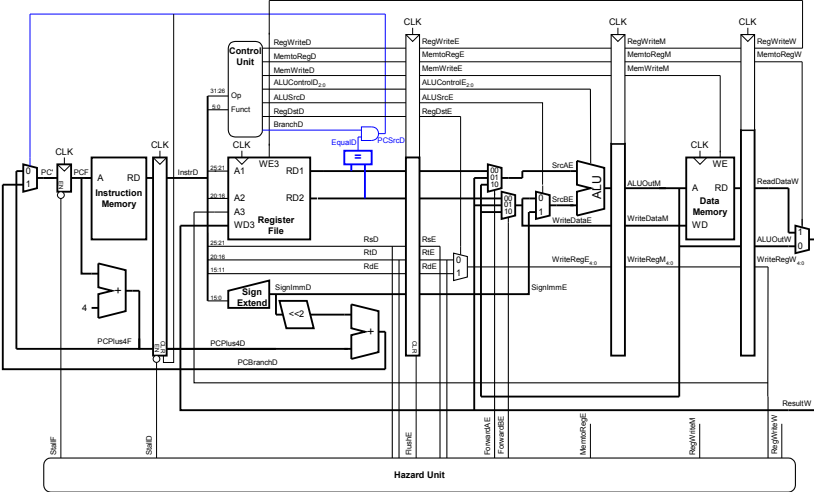
register being loaded from memory

Control Hazards



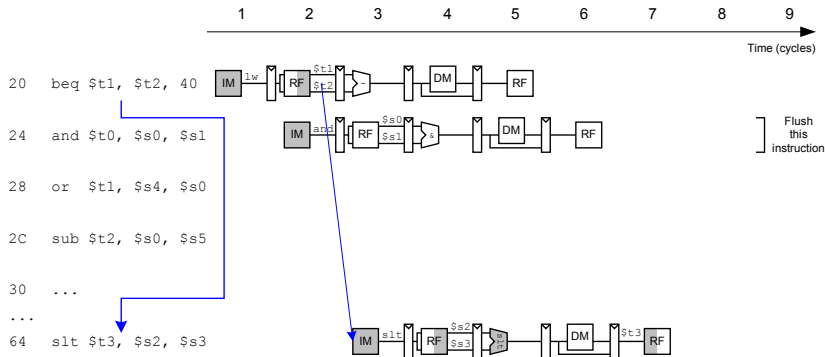
Whether to branch isn't determined until the beginning of the fourth cycle; three instructions would be executed erroneously if the branch were taken.

Early Branch Resolution

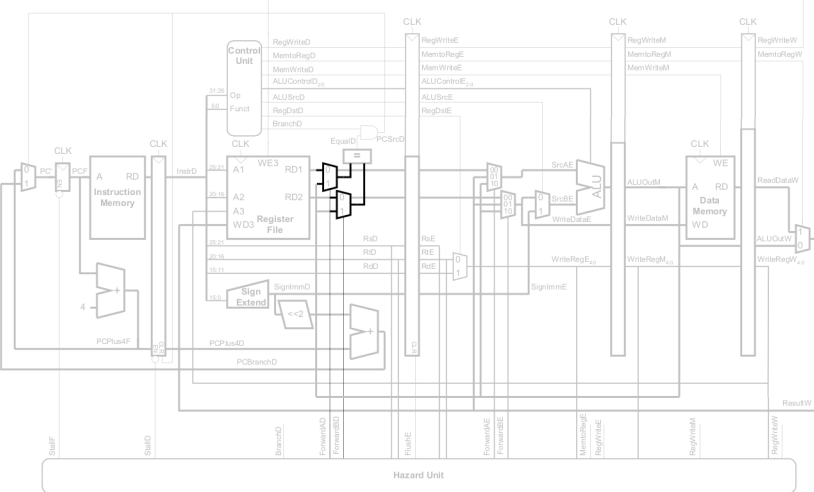


Introduced another data hazard in the decode stage

Control Hazards w/ Early Branch Resolution



Handling Data and Control Hazards



Forwarding and Stalling Logic

“Forward ALU result to branch-if-equal comparator if we would read its destination register”

$$\text{ForwardAD} = (\text{rsD} \neq 0 \wedge (\text{rsD} = \text{WriteRegM}) \wedge \text{RegWriteM})$$

$$\text{ForwardBD} = (\text{rtD} \neq 0 \wedge (\text{rtD} = \text{WriteRegM}) \wedge \text{RegWriteM})$$

“Stall if the branch would test the result of an ALU operation or a memory read”

branchstall =

$$\begin{aligned} & (\text{BranchD} \wedge \text{RegWriteE} \wedge (\text{WriteRegE} = \text{rsD} \vee \text{WriteRegE} = \text{rtD})) \vee \\ & (\text{BranchD} \wedge \text{MemToRegM} \wedge (\text{WriteRegM} = \text{rsD} \vee \text{WriteRegM} = \text{rtD})) \end{aligned}$$

“Stall if we need to read the result of a memory read or of a branch”

$$\text{StallF}, \text{StallD}, \text{FlushE} = \text{lwstall} \vee \text{branchstall}$$

Pipeline Performance CPI Example

Ideal CPI = 1; stalls reduce this, but how much?

Instructions in SPECINT2000 benchmark:

- 52% R-type
- 25% Loads
- 10% Stores
- 11% Branches
- 2% Jumps

If 40% of loads are used by the next instruction and 25% of branches are mispredicted, what is the average CPI?

Pipeline Performance CPI Example

Ideal CPI = 1; stalls reduce this, but how much?

Instructions in SPECINT2000 benchmark:

- 52% R-type
- 25% Loads
- 10% Stores
- 11% Branches
- 2% Jumps

If 40% of loads are used by the next instruction and 25% of branches are mispredicted, what is the average CPI?

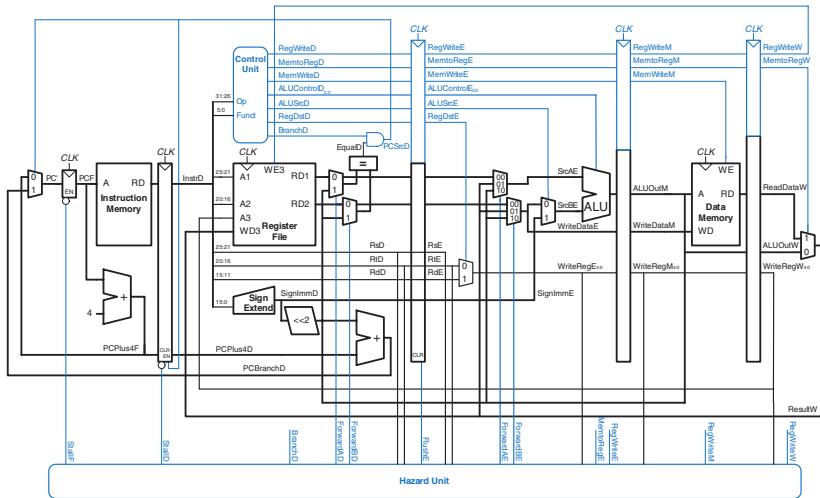
Load CPI = 2 when next instruction uses; 1 otherwise

Branch CPI = 2 when mispredicted; 1 otherwise

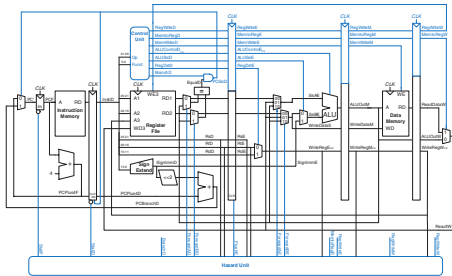
Jump CPI = 2

$$\begin{aligned} \text{Average CPI} &= 1 \cdot 0.52 + && R\text{-type} \\ & (2 \cdot 0.40 + 1 \cdot 0.60) \cdot 0.25 + && \text{Loads} \\ & 1 \cdot 0.10 + && \text{Stores} \\ & (2 \cdot 0.25 + 1 \cdot 0.75) \cdot 0.11 + && \text{Branches} \\ & 2 \cdot 0.02 && \text{Jumps} \\ &= 1.1475 \end{aligned}$$

Fully Bypassed Processor



Pipelined Processor Critical Path



Element	Delay
Register clk-to-Q	t_{pcq} 30 ps
Register setup	t_{setup} 20
Multiplexer	t_{mux} 25
ALU	t_{ALU} 200
Memory Read	$t_{memread}$ 250
Register file read	t_{RFread} 150
Register file setup	$t_{RFsetup}$ 20
Equality	t_{eq} 40
AND gate	t_{AND} 15
Memory Write	$t_{memwrite}$ 220
Register file write	$t_{RFwrite}$ 100

$$T_C = \max \left\{ \begin{array}{l} t_{pcq} + t_{memread} + t_{setup} \\ 2(t_{RFread} + t_{mux} + t_{eq} + t_{AND} + t_{mux} + t_{RFsetup}) \\ t_{pcq} + t_{mux} + t_{mux} + t_{ALU} + t_{setup} \\ t_{pcq} + t_{memwrite} + t_{setup} \\ 2(t_{pcq} + t_{mux} + t_{RFwrite}) \end{array} \right. \left. \begin{array}{l} \textit{Fetch} \\ \textit{Decode} \\ \textit{Execute} \\ \textit{Memory} \\ \textit{Writeback} \end{array} \right\}$$

$$= 2(150 + 25 + 40 + 15 + 25 + 20) \text{ ps} = 550 \text{ ps}$$

Why 2? We assume it takes half a cycle for a newly written register's value (WD3) to propagate to RD1 or RD2, i.e., when an earlier instruction writes a register used by the current one.

Pipelined Processor Performance

For a 100 billion-instruction task on our pipelined processor, each instruction takes 1.15 cycles on average. With a 550 ps clock period,

$$\text{time} = 100 \times 10^9 \times 1.15 \times 550 \text{ ps} = 63 \text{ seconds}$$

Processor	Execution Time	Speedup
Single-Cycle	92.5 s	1.00 (by definition)
Multi-Cycle	133.9	0.70
Pipelined	63.25	1.46