

Fundamentals of Computer Systems

A Pipelined MIPS Processor

Stephen A. Edwards

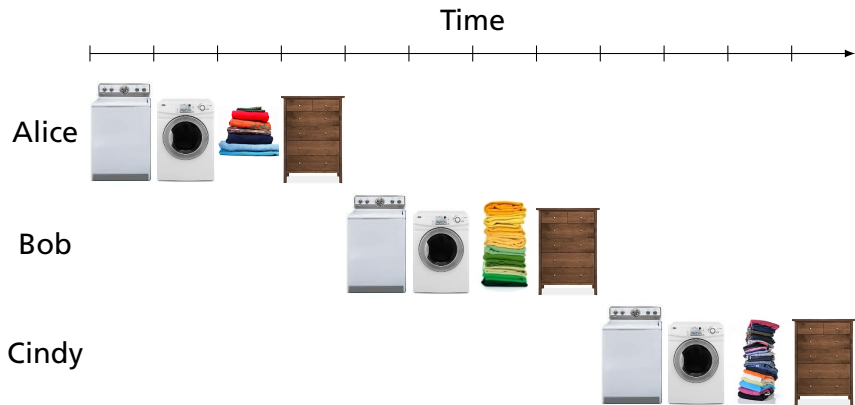
Columbia University

Summer 2020

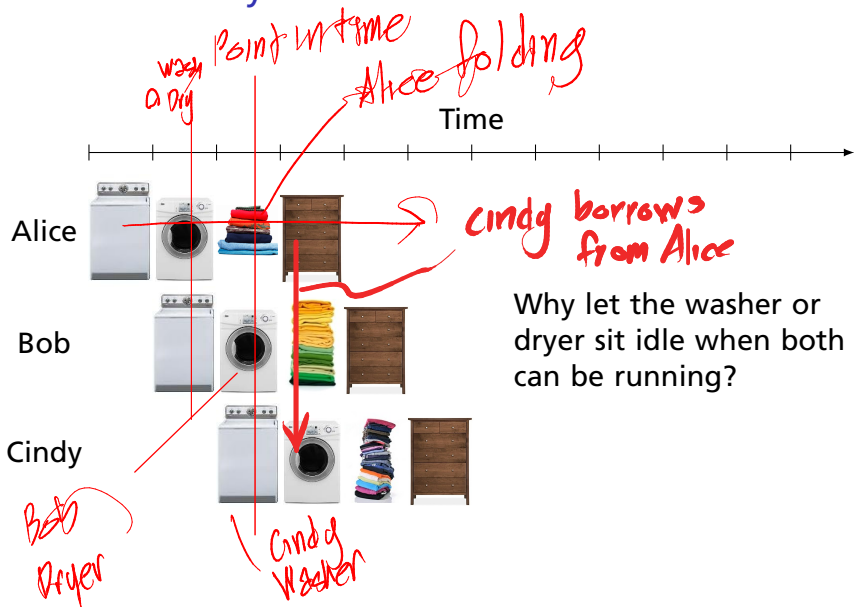


Technical Illustrations Copyright ©2007 Elsevier

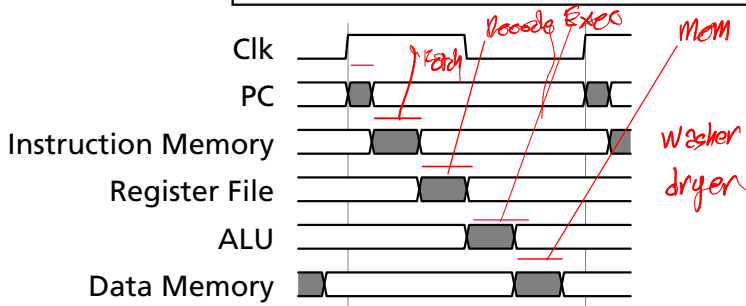
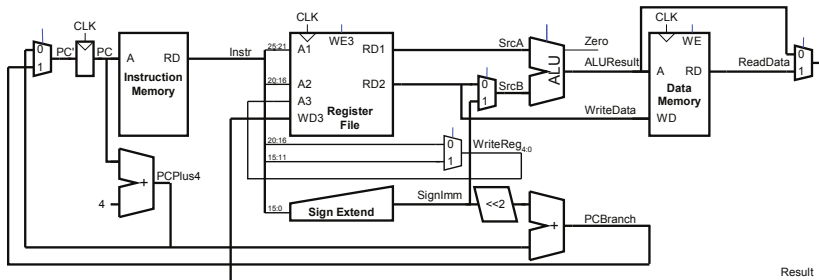
Sequential Laundry



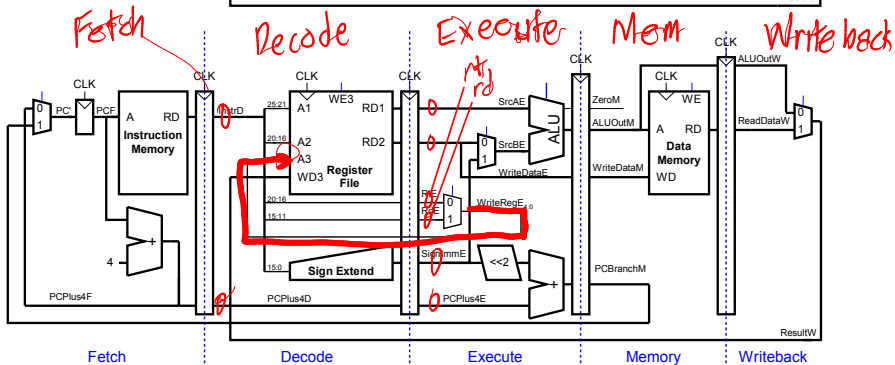
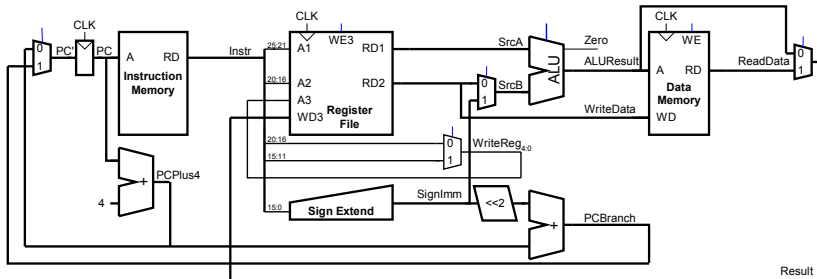
Pipelined Laundry



Single-Cycle Datapath Timing

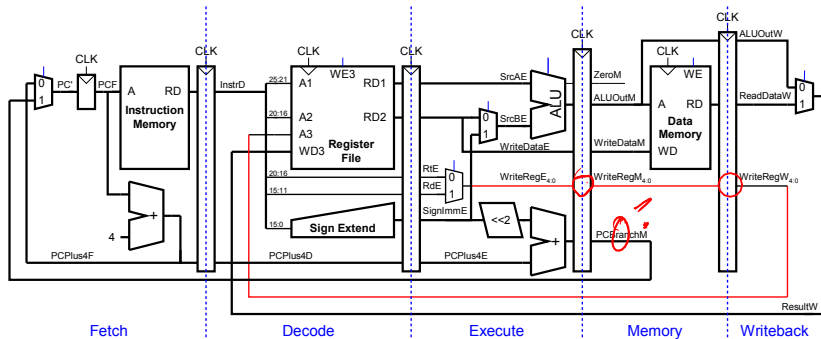


Single-Cycle vs. Pipelined Datapath



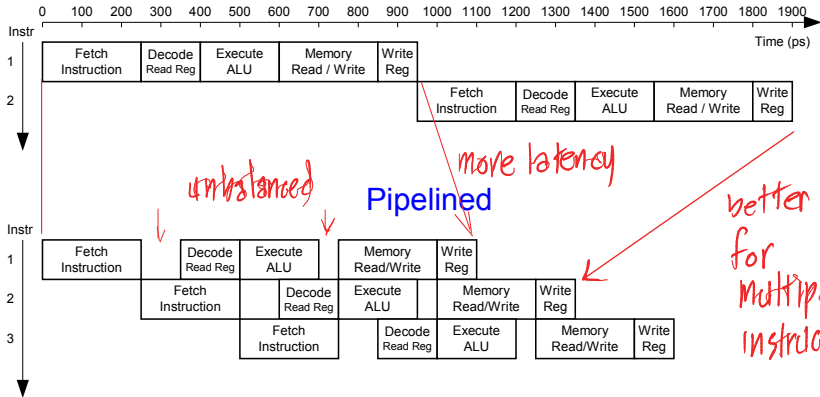
Corrected Pipelined Datapath

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

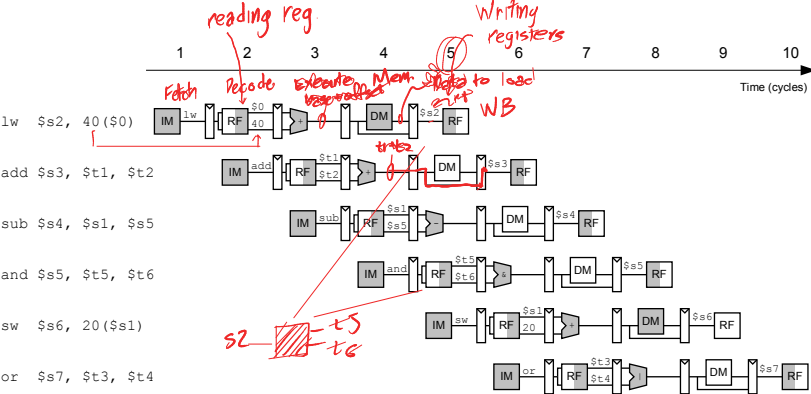


Single-Cycle vs. Pipeline Timing

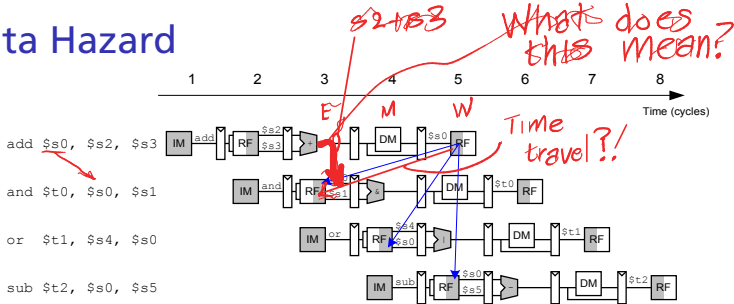
Single-Cycle



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.



Biohazard



Water Hazard

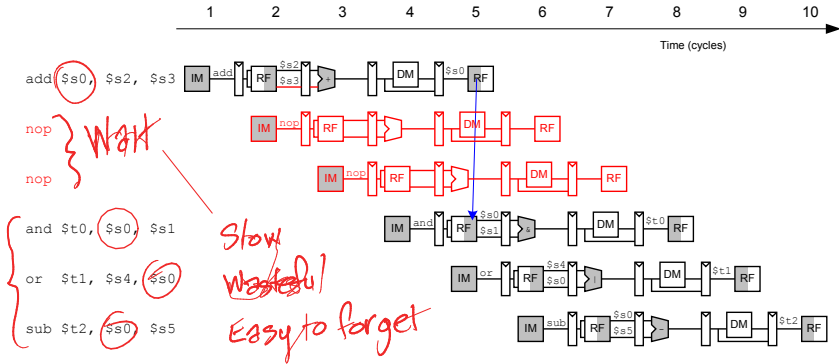


Hazard Lights



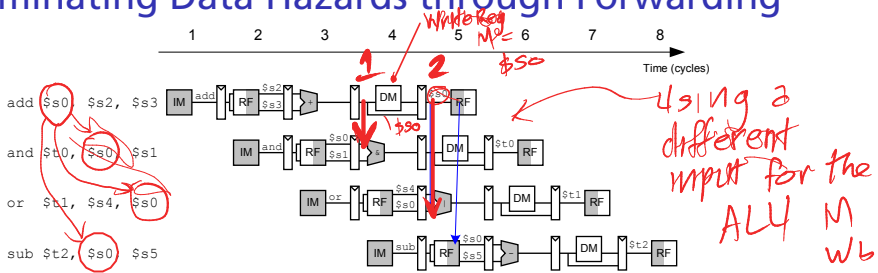
Dukes of Hazard

Eliminating Hazards at Compile-Time



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

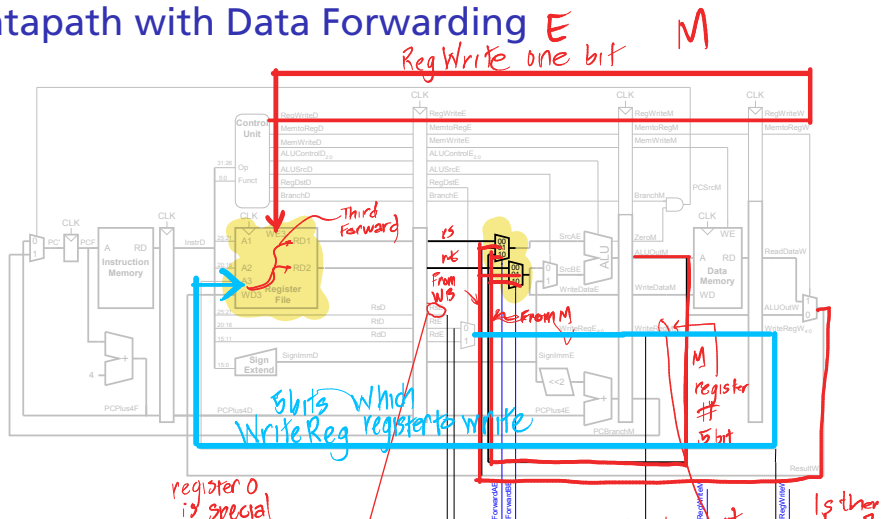
Eliminating Data Hazards through Forwarding



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.

Datapath with Data Forwarding

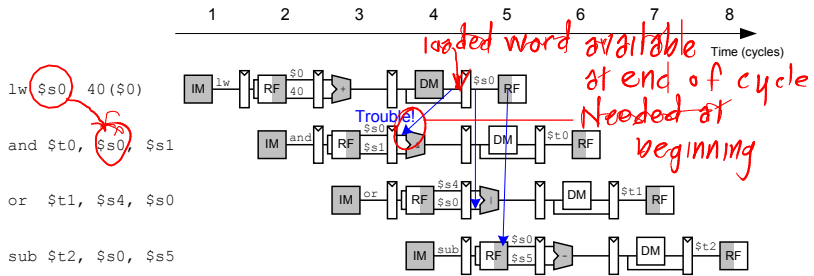


$ForwardAE = \begin{cases} 10 & \text{if } rsE \neq 0 \wedge rsE = WriteRegM \wedge RegWriteM \\ 01 & \text{if } rsE \neq 0 \wedge rsE = WriteRegW \wedge RegWriteW \\ 00 & \text{otherwise.} \end{cases}$

10 if $rsE \neq 0 \wedge rsE = WriteRegM \wedge RegWriteM$ in the M stage
 01 if $rsE \neq 0 \wedge rsE = WriteRegW \wedge RegWriteW$ W stage
 00 otherwise. M stage
 W add \$s0, ...
 M add \$s0, \$s0
 E add \$s0, ...

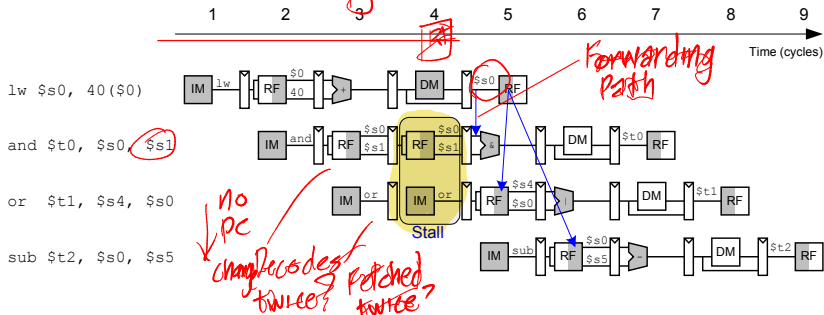
Hazard Unit
 What register to write
 source register # in E stage
 Which one should take precedence?
 $\Rightarrow M$, the latest
 Is there data?

Data Hazard that Demands a Stall



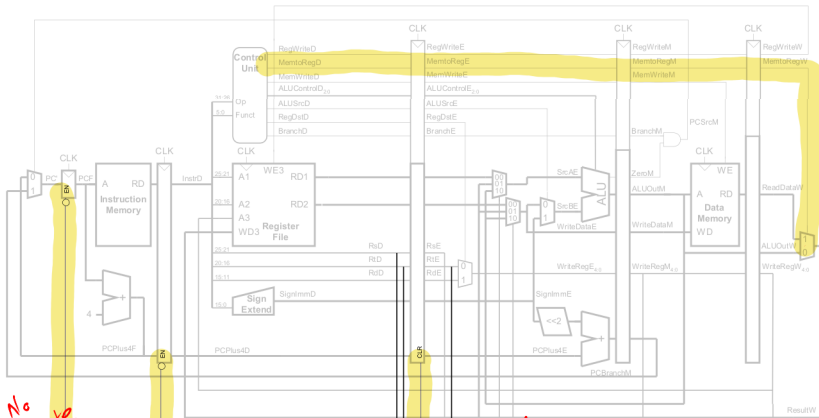
This data hazard can't be solved with forwarding because the value is only available at the end of cycle 4, yet is needed at the beginning.

Stalling a Pipeline



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

Stalling Hardware



No PC update

Fetch instruction again - Decode again

Reading from mem

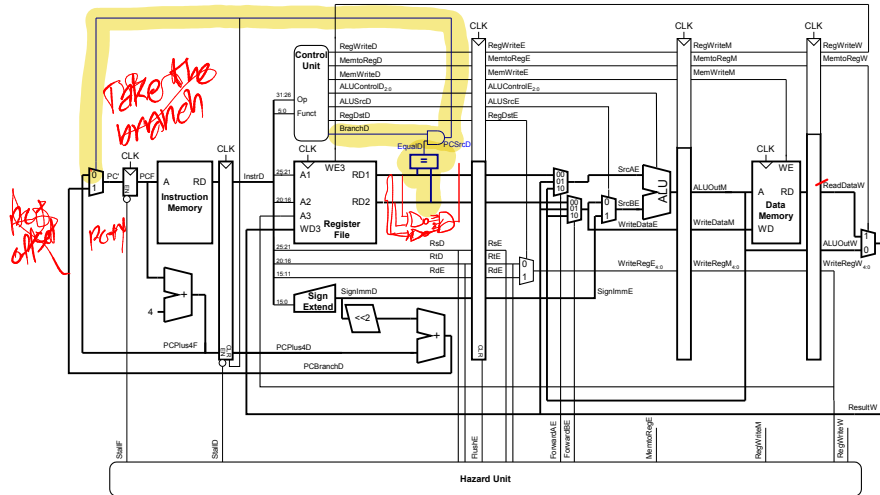
$$lwstall = \text{MemToRegE} \wedge ((rsD = rtE) \vee (rtD = rtE))$$

$$StallF, StallD, FlushE = lwstall$$

decode and execute are using the same Reg
 Into what reg are we loading

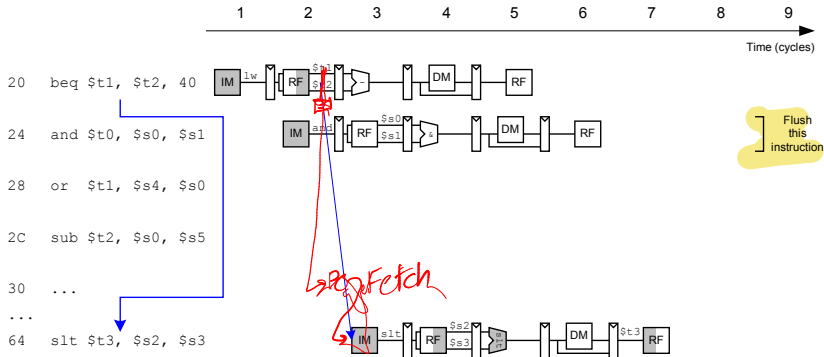
Early Branch Resolution

Decoder

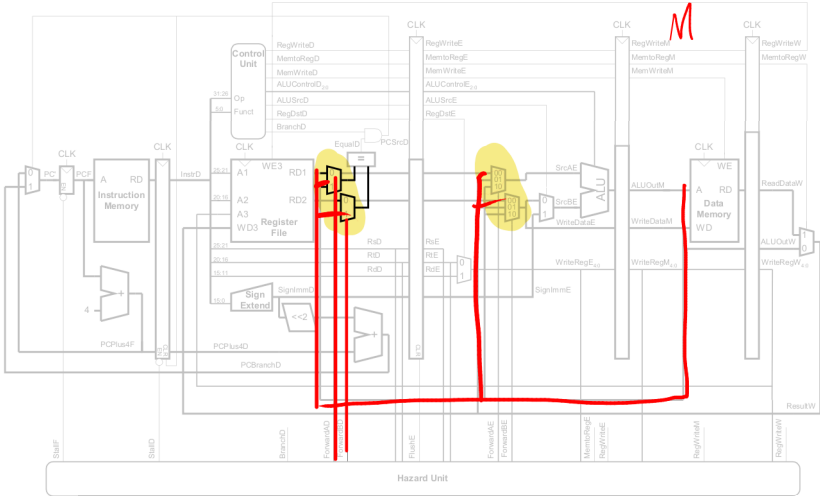


Introduced another data hazard in the decode stage

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 =

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

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

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

Pipeline Performance CPI Example

cycles per instruction

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

Instructions in SPECINT2000 benchmark:

52% R-type

25% Loads

10% Stores

11% Branches

2% Jumps

CPI = 2

CPI = 2

*cycle 6
stall
lw \$t0
add \$t0, \$t0
2 CPI*

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

$$\begin{aligned} \text{Average CPI} &= 1 \cdot 0.52 \quad \text{R-type} \\ &+ (1 \cdot 0.25 + 2 \cdot 0.40) \cdot 0.25 \quad \text{loads} \\ &+ (1 \cdot 0.10 + 2 \cdot 0.25) \cdot 0.11 \quad \text{branches} \\ &+ 2 \cdot 0.02 \quad \text{jumps} \end{aligned}$$

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

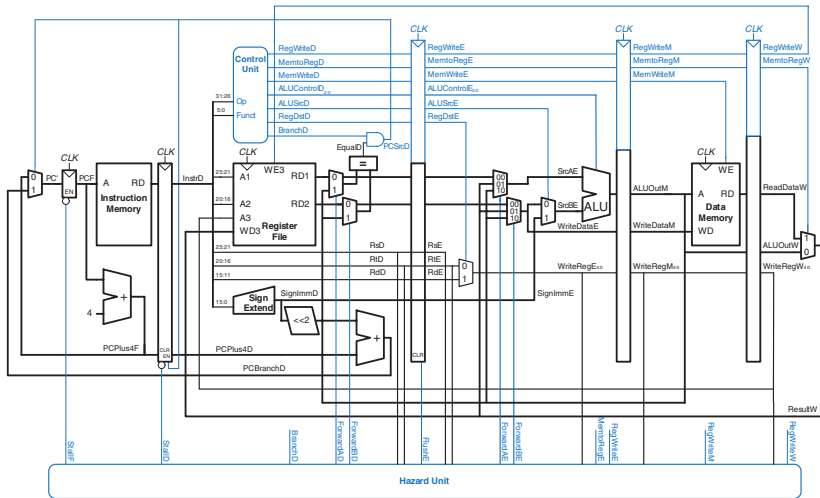
$$\begin{aligned} \text{Average CPI} &= 1 \cdot 0.52 + && R\text{-type} \\ &+ (2 \cdot 0.40 + 1 \cdot 0.60) \cdot 0.25 + && \text{Loads} \\ &+ 1 \cdot 0.10 + && \text{Stores} \\ &+ (2 \cdot 0.25 + 1 \cdot 0.75) \cdot 0.11 + && \text{Branches} \\ &+ 2 \cdot 0.02 && \text{Jumps} \end{aligned}$$

= 1.1475

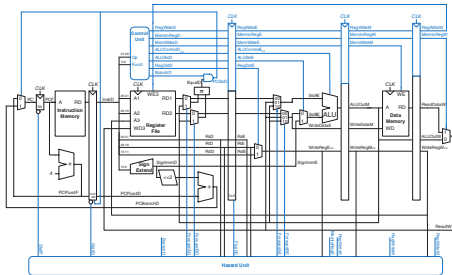
~~instead of 1~~

multicycle
4, 1/2

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 |

crazy

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

Fetch
Decode
Execute
Memory
Writeback

pipeline stages

Decode stage

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

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 |

Memory kills!

< 5 ☹

CPI 1
CPI 4.12,
CPI 1.15