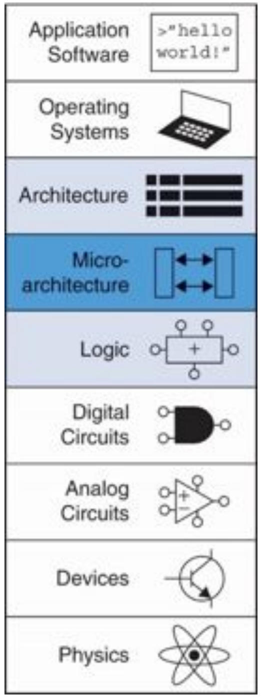
COSC175 (Systems I): Computer Organization & Design

Professor Lillian Pentecost
Fall 2024

Warm-Up December 2

- Where we were
  - Multi-cycle vs. single cycle RISC-V processor
- Where we are going
  - ***Pipelined*** execution
- Logistics, Reminders
  - TA help 7-9PM on Sundays, Tuesdays, Thursdays in C107
    - Use to start review of whole semester!
  - LP Office hours M 9-10:30AM, Th 2:30-4PM
  - **No more weekly exercises**! Focus on your final lab, and starting to review previous assignments + exercises
  - BRING YOUR PROCESSOR LAB QUESTIONS INTO LAB WEDNESDAY!!
  - **ATTEND FACULTY CANDIDATE TALK TODAY AT 4:30PM in A131, WITH SNACKS @ 4PM in C209**
    - *If you attend at least 3 candidate talks, then send me an email including 1 thing you learned from each talk, I'll give you 5% extra credit on any previous lab report*



Single- vs. Multicycle Processor

- **Single-cycle:**
  - + simple
  - - cycle time limited by longest instruction (lw)
  - - separate memories for instruction and data
  - - 3 adders/ALUs
- **Multicycle:** ***slower than single-cycle, unless clock cycle time and/or CPI improves***
  - + higher clock speed
  - + simpler instructions run faster
  - + reuse expensive hardware on multiple cycles
  - - sequencing overhead paid many times

Goals for Today

- **Introduce alternative design: pipelined processor**
- **Keep in mind: what are the pros and cons of this design vs. single-cycle?**
- **How is the performance?**

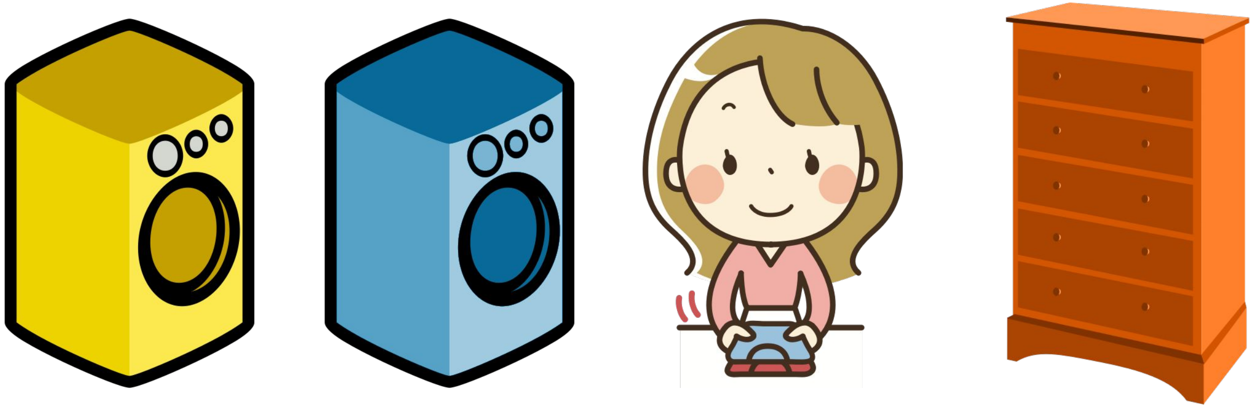
Same design steps as single-cycle:

- first datapath
- then control

… remember the memory hierarchy, tho?

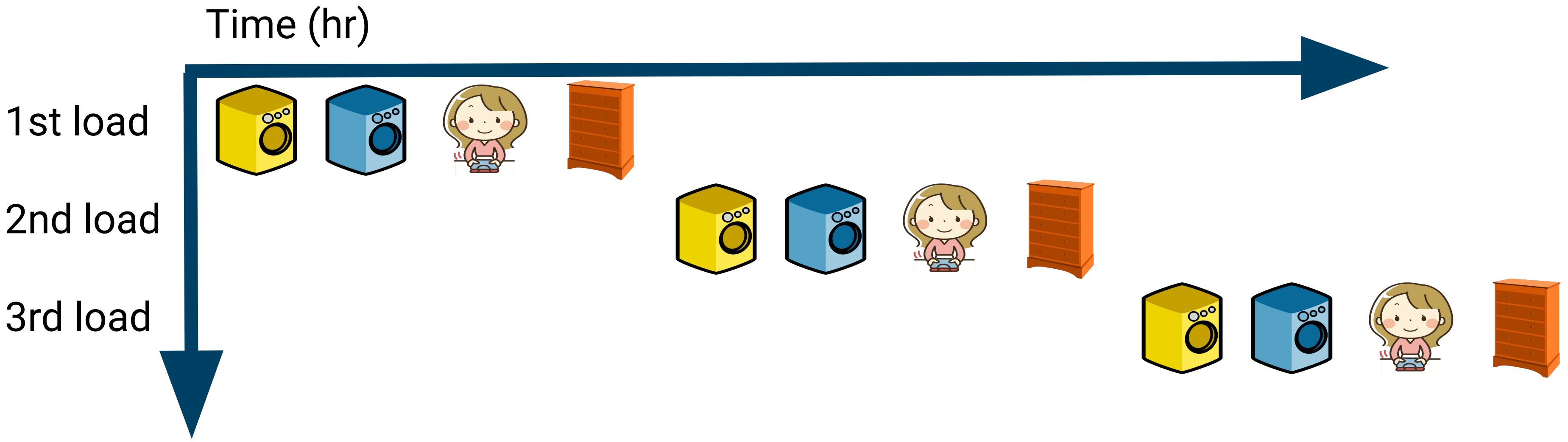
Introducing Pipelining

- ***Temporal Parallelism***: multiple instructions in-progress at the same time
- Start with an analogy: loads of laundry
- HW units available: washer, dryer, folder (you & a table), dresser

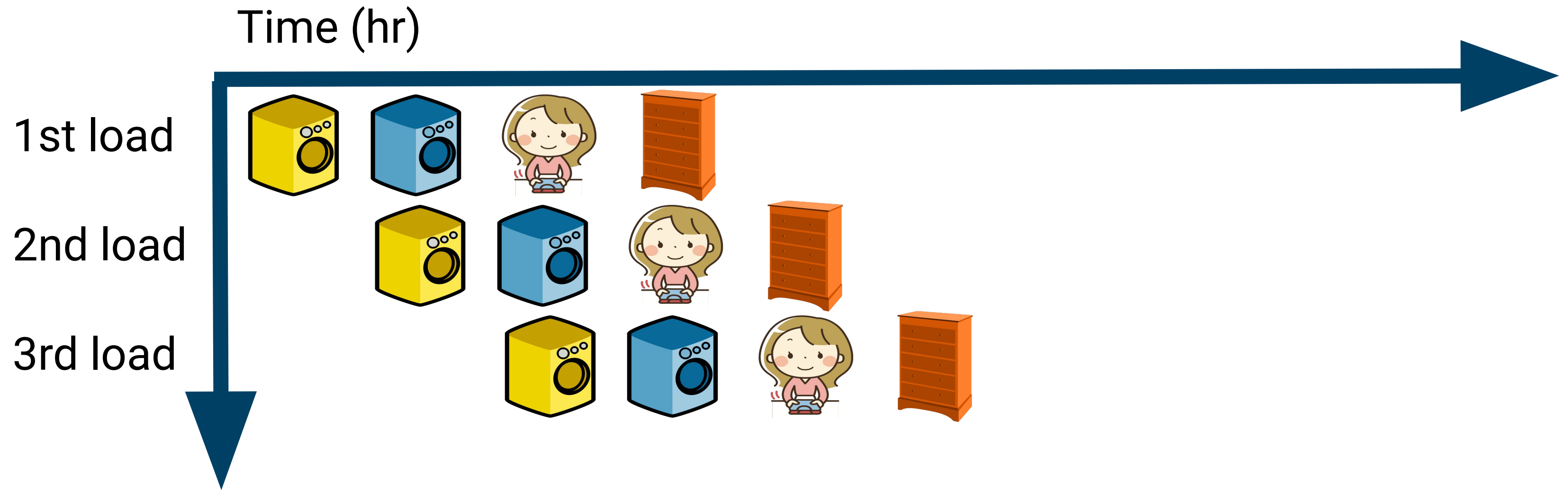


Execution stages: wash, dry, fold, put away

Single-Cycle vs. Pipelined Laundry



Single-Cycle vs. **Pipelined** Laundry



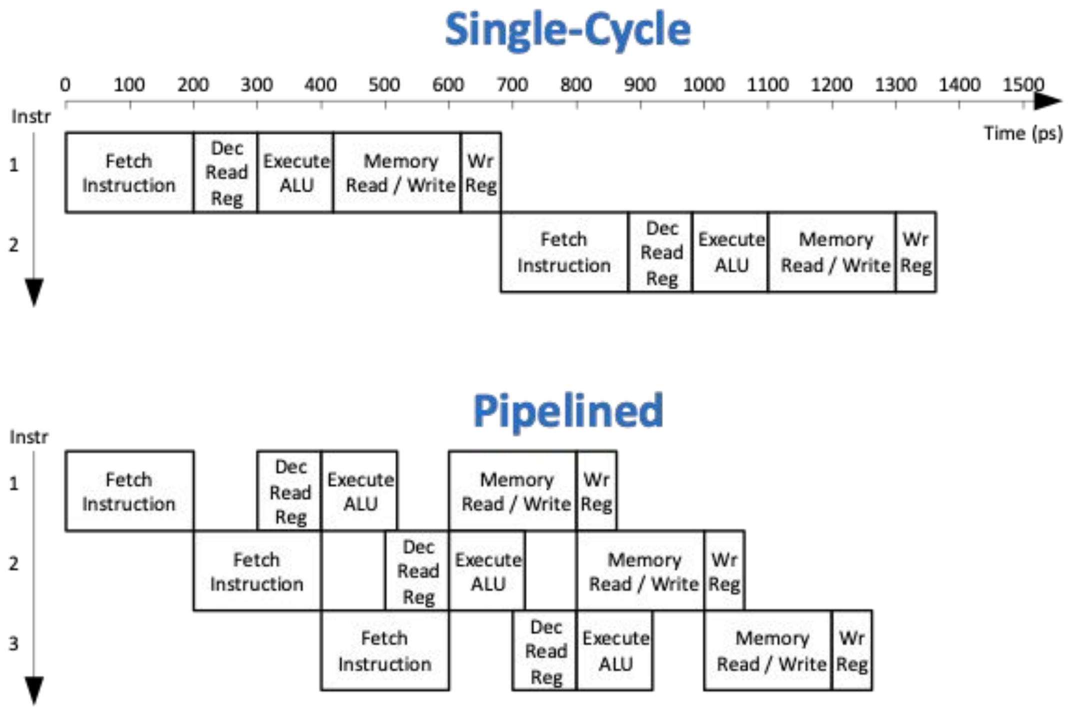
- ***Temporal Parallelism***: multiple instructions in-progress at the same time
- HW units stay busy

Introducing a Pipelined Processor Design

- ***Temporal Parallelism***: multiple instructions in-progress at the same time
- Divide single-cycle processor into 5 stages:
  - Fetch
  - Decode
  - Execute
  - Memory
  - Writeback
- Add **pipeline registers** between stages to propagate output of one stage as input to the next at the clock tick

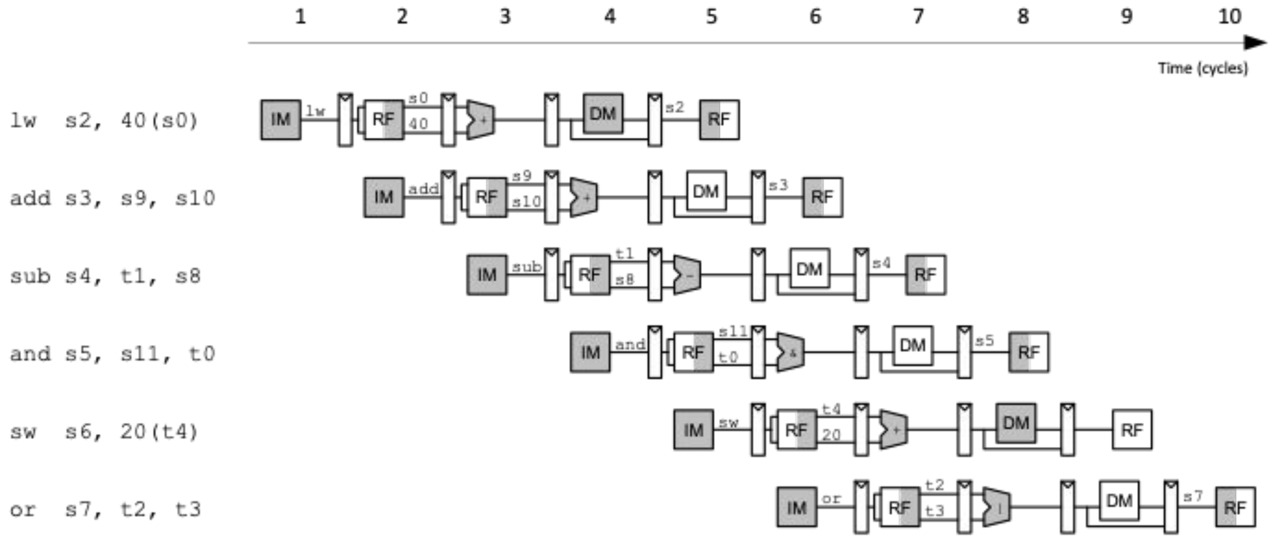
Introducing a Pipelined Processor Design

- Looks just like our laundry! HW units are kept busy by starting execution of subsequent instructions
- What do we like about this idea?
- Do we have any questions or concerns about this design?



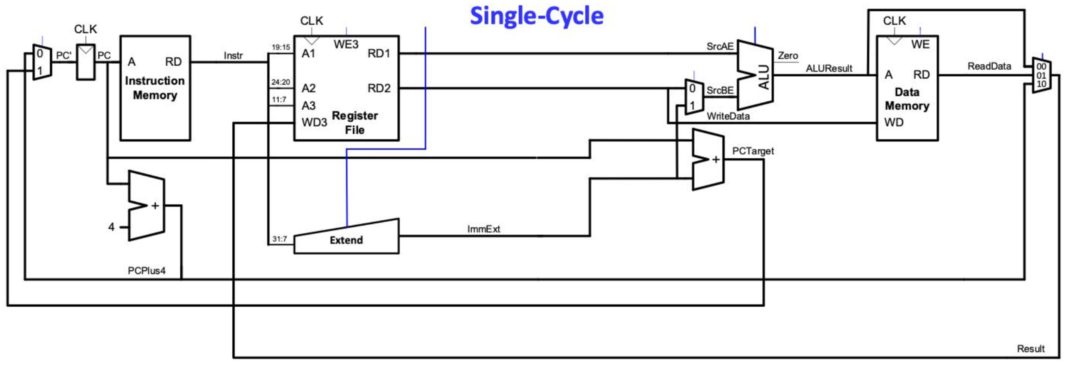
Introducing a Pipelined Processor Design

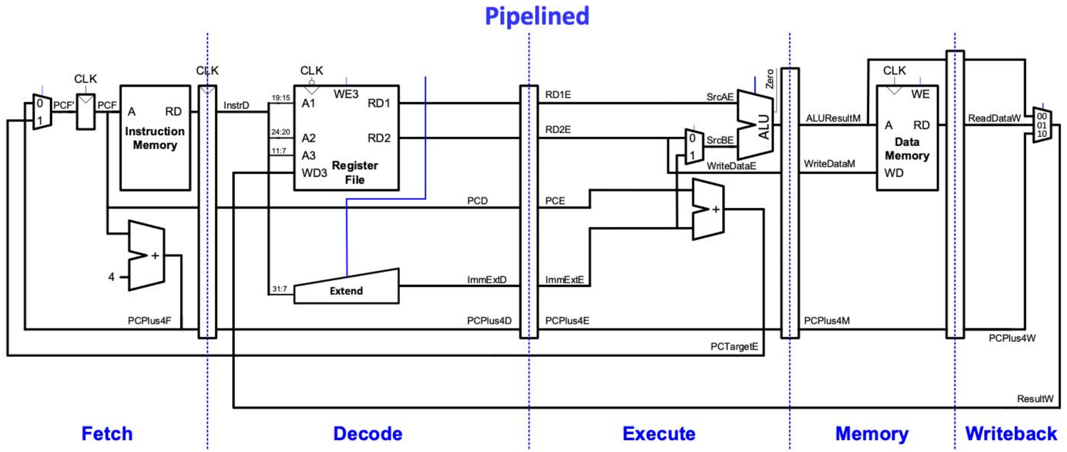
- Fetch reads IM, Decode reads RF, Execute uses ALU, Memory access DM, Writeback updates RF
- What do we like about this idea? **What is your estimate of the average CPI?**
- Do we have any questions or concerns about this design?



Updated Design Diagram

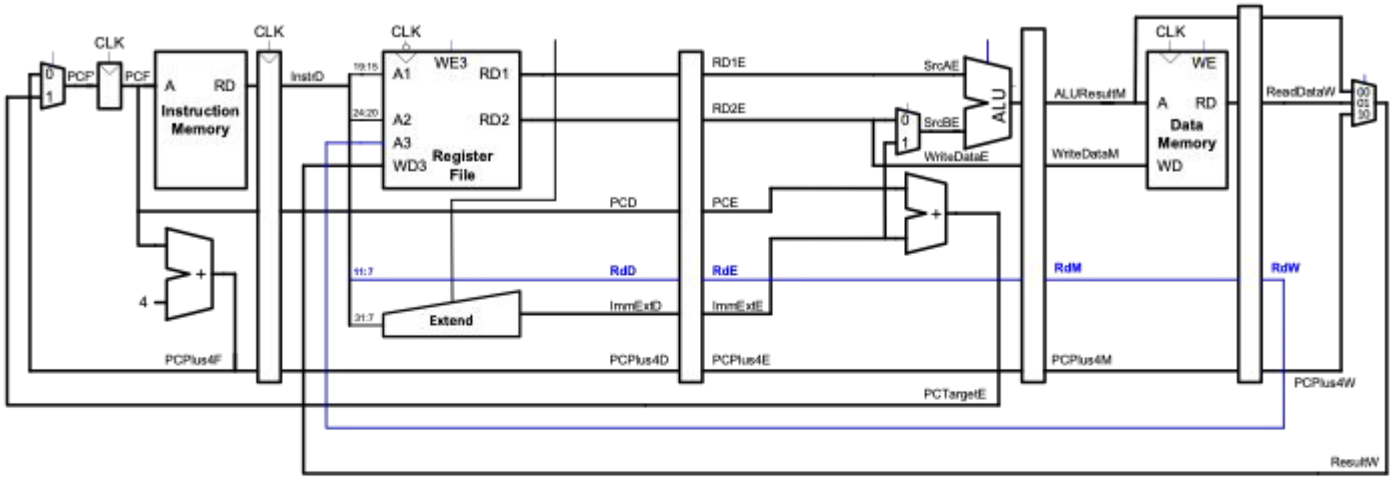
- Signals are now tied with specific stages, and we append the first letter of the relevant stage
- Are there any issues here?
- Let's trace through each stage





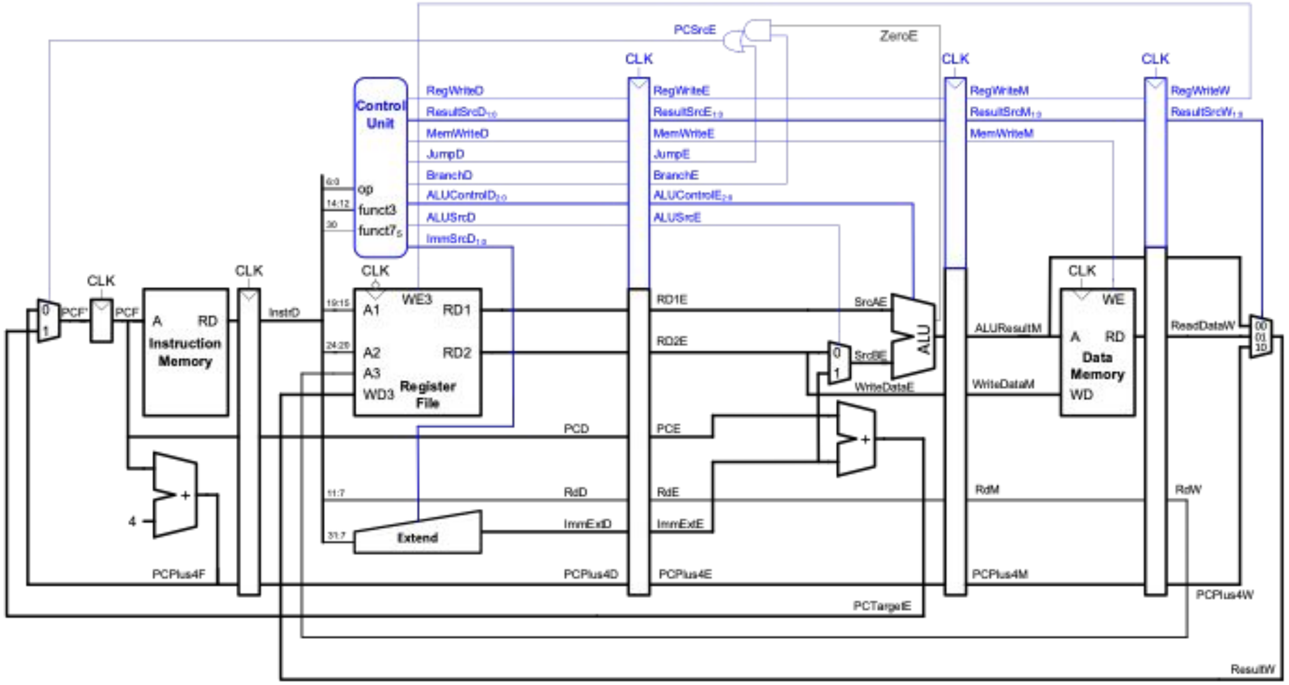
Updated Design Diagram: correct for WB

- **Rd** (destination register) should arrive at same time as **Result** for WB stage
- RF will be written on **falling edge** of CLK to account for delay



Control Unit is Same as Single-Cycle

- **Control Signals travel with instruction stage-by-stage**



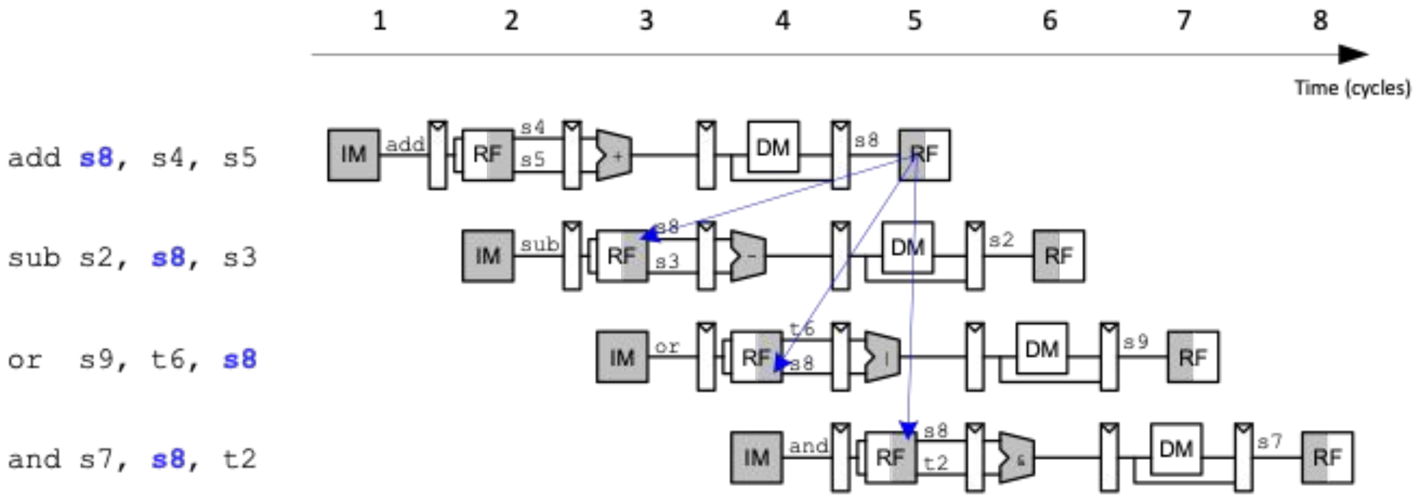
Let’s use the rest of class to discuss

- **The Good News:** performance of the pipelined processor is better than the single-cycle (we’ll compute by how much)

- **The Bad News:** We introduce **hazards** – subsequent instructions may depend on an instruction that ***hasn’t completed yet***
    - **Data Hazard** if a required register value is not yet written back to the register file
    - **Control Hazard** if the next instruction to execute is not determined yet (i.e., branch if X)

Data Hazard Example

- sub relies on the result of add , or relies on the result of sub, and so on…



<u>What might we do to fix this?</u>

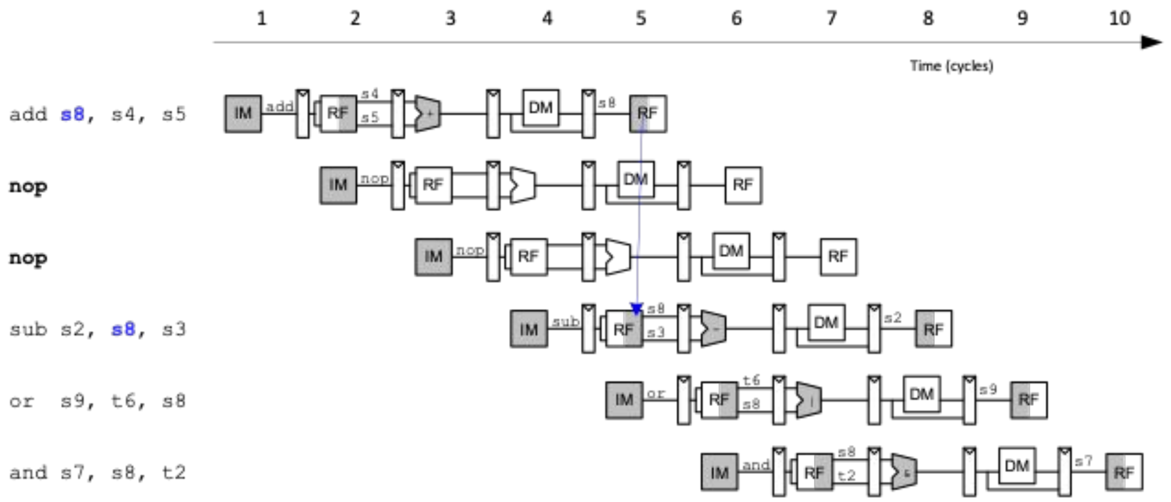
Resolving Data Hazards

- At the time we **compile** our code, we might check for data hazards and choose to:
    - Rearrange instruction order
    - Insert nops between dependent instructions

- At **execution** time (or run time), we might choose to:
    - Stall (i.e., sit idle) until the required value is ready
    - “Forward” data to a dependent instruction as soon as it is ready

Resolving Data Hazards: Compile Time

- Fill the time with useful, independent instructions until hazard resolves
- If there are no such independent instructions, insert nop

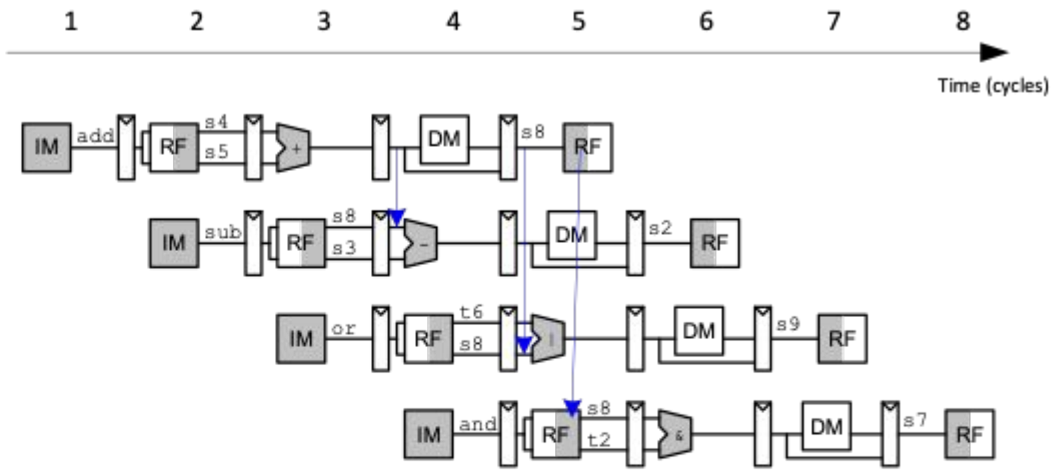


```
add s8, s4, s5
nop
nop
sub s2, s8, s3
or  s9, t6, s8
and s7, s8, t2
```

Resolving Data Hazards: Run Time

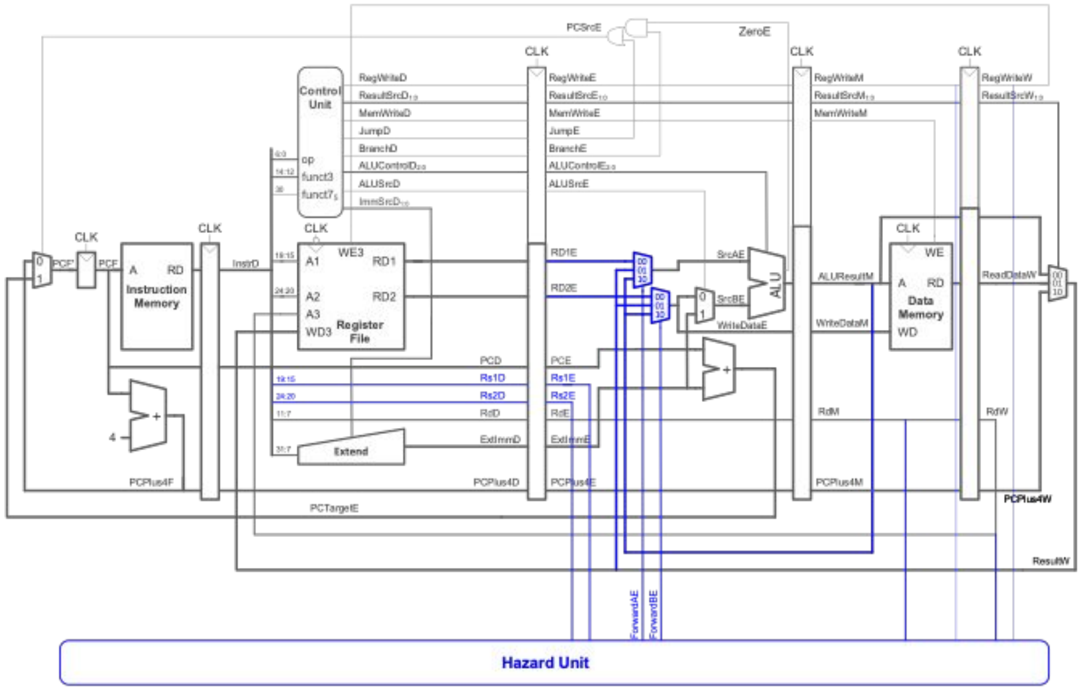
- Value needed will exist (i.e., ALU produced it) before it is written back to the RF
- Data Forwarding is a “sneak peak” of a previous instruction’s result to avoid stalling
- Compare if an instruction’s **source** register in Execute stage matches the **destination** register of an instruction in the Memory or WB stage; if yes, then forward **ALUResult**

```
add s8, s4, s5

sub s2, s8, s3

or  s9, t6, s8

and s7, s8, t2
```



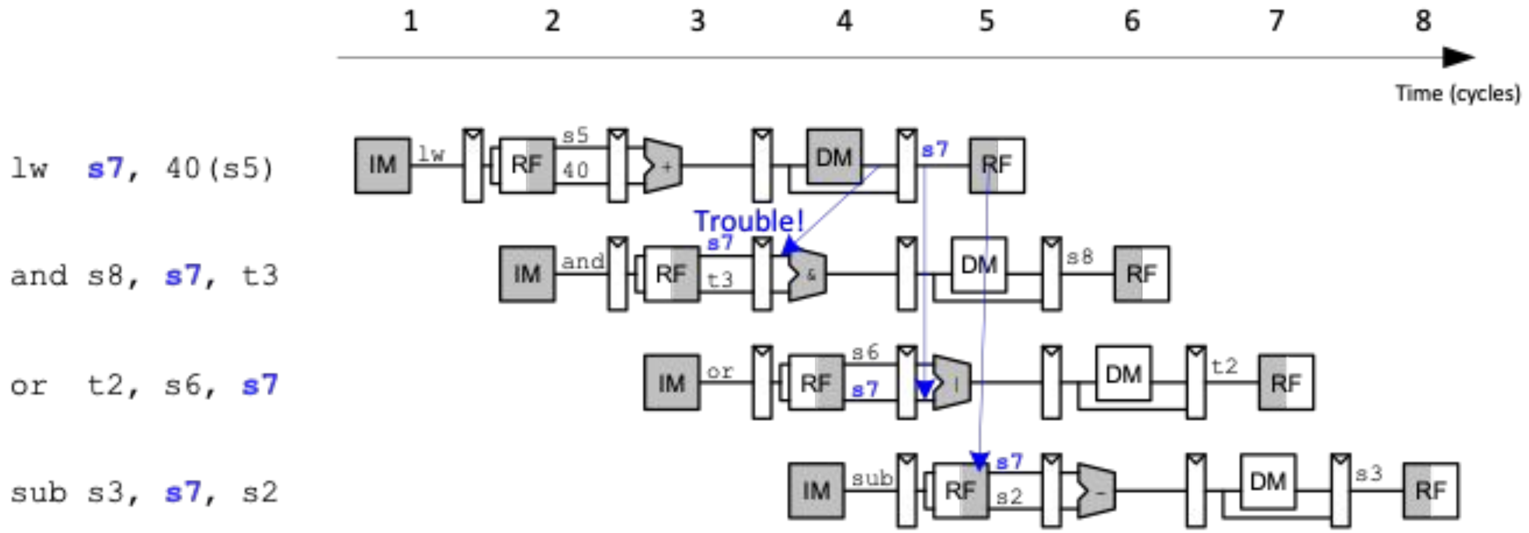
Resolving Data Hazards: Run Time

- “Hazard Unit” will check hazard condition(s) each clock cycle



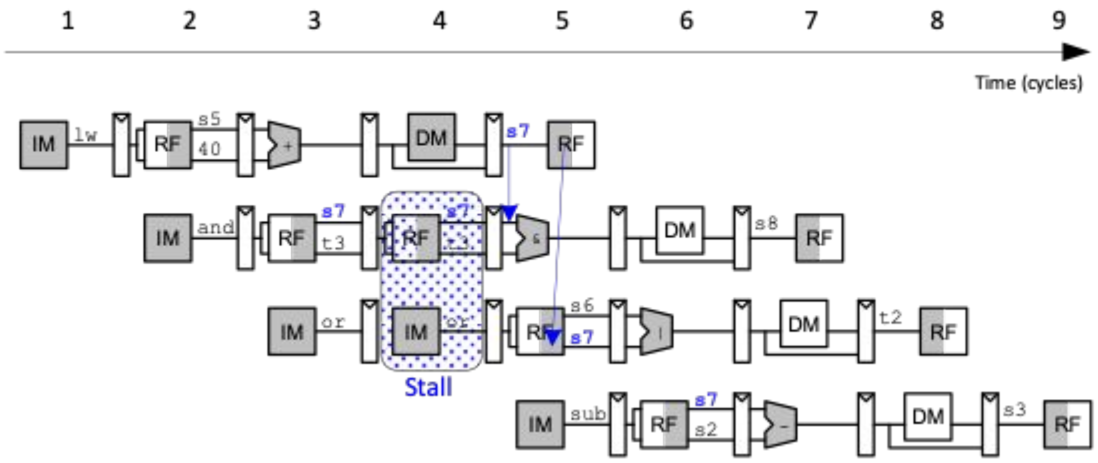
Resolving Data Hazards

- What other data hazards may exist?



Resolving Data Hazards: sometimes you just have to be patient (stall)

```
lw   s7, 40(s5)

and s8, s7, t3

or   t2, s6, s7

sub s3, s7, s2
```

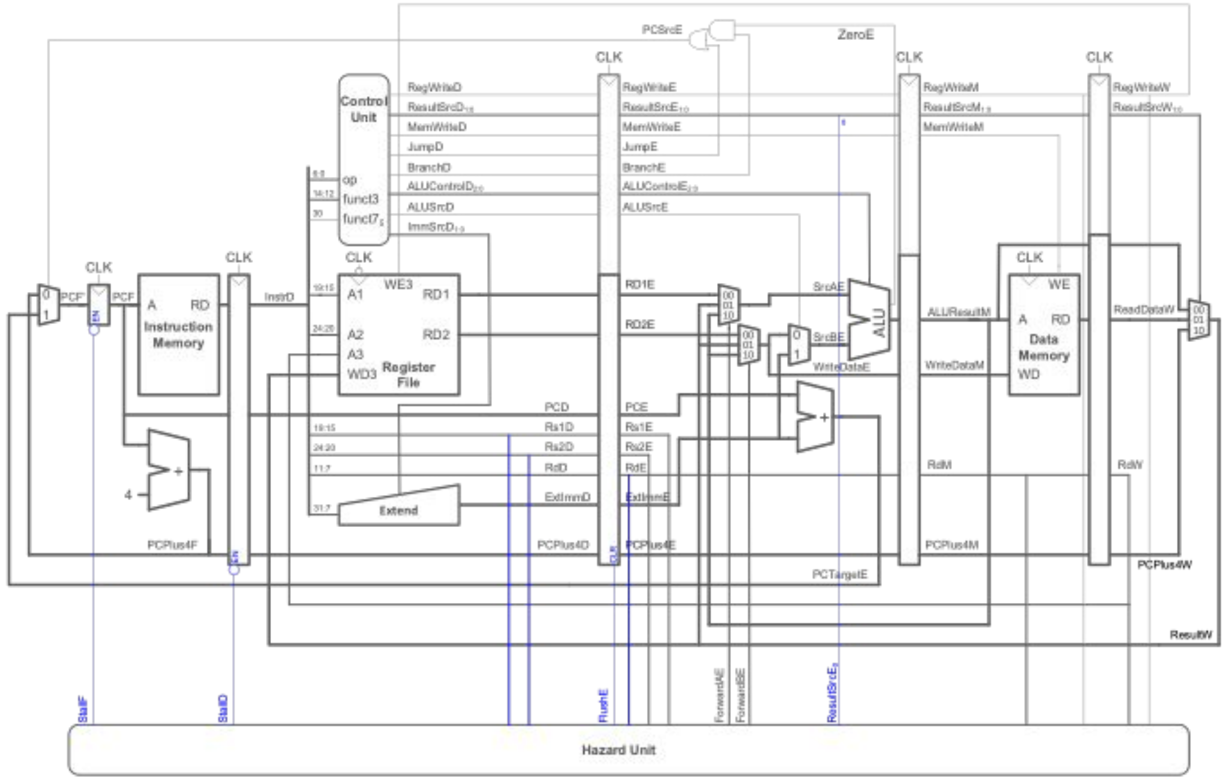


- Stall condition:
  - Is either **source** register in the Decode stage the same as the **destination** register in the Execute stage?

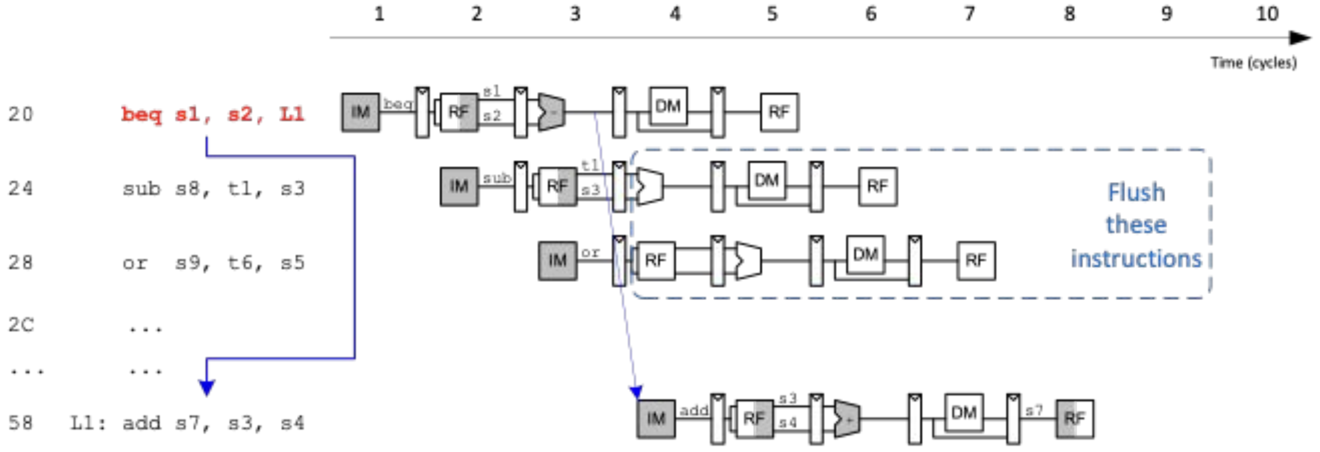
    AND

  - Is the instruction in the Execute stage a **lw**?

Pipelined Processor with Data Hazard Protection

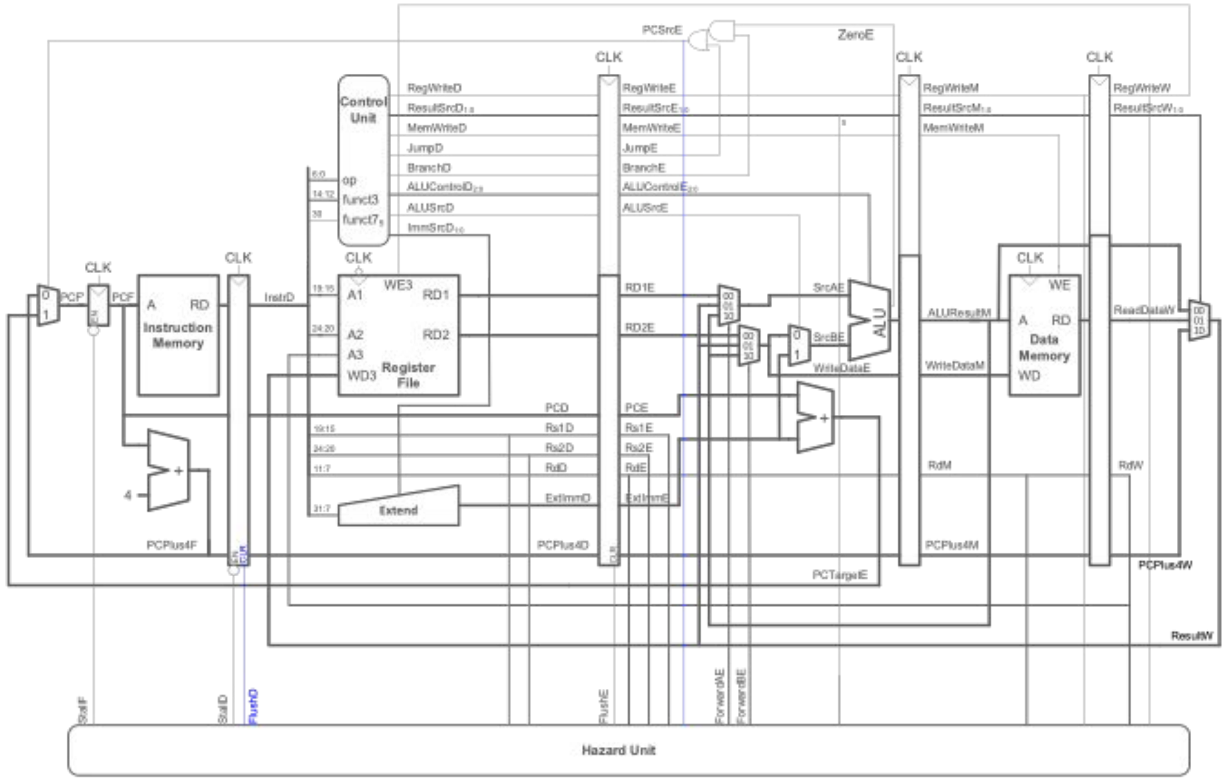


What about **control** hazards?



- We don't determine whether the branch is taken until **execute** stage
- If the branch is taken, we've started execution of 2 instructions that we shouldn't have! Uh oh!!
- In the case of a **taken** branch, we must **flush** the in-progress instructions and start fresh – in this case, a **branch misprediction penalty** of 2 instructions

Pipelined Processor with Hazard Protection! Now, what about performance?



Pipelined Processor with Hazard Protection! Now, what about performance?

- Remember SPECINT2000 benchmarks:
  - 25% loads
  - 10% stores
  - 13% branches
  - 52% R-type
- Suppose:
  - 40% of loads used by next instruction (causes 1 cycle stall)
  - 50% of branches mispredicted (causes 2 cycle penalty)
- What do we need to compute the performance?
  - CPI and Clock Cycle time, as usual!

Pipelined Processor with Hazard Protection! Now, what about performance?

- Remember SPECINT2000 benchmarks:
  - 25% loads
  - 10% stores
  - 13% branches
  - 52% R-type
- Suppose:
  - 40% of loads used by next instruction (causes 1 cycle stall)
  - 50% of branches mispredicted (causes 2 cycle penalty)
- ***Average CPI***: unfortunately, it isn't 1
  - CPI for a load instruction is 2 cycles 40% of the time
  - CPI for a branch is 3 cycles 50% of the time

Average CPI = (% load) (cycles per load) + (% store) (cycles per store) + (% branch) (cycles per branch) + (% R-type) (cycles per R-type)

Average CPI = (0.25) (1*0.6+2*0.4) + (0.1) (1) + (0.13) (2) + (0.52) (1) = **1.23**

Pipelined Processor with Hazard Protection! Now, what about performance?

Clock Cycle time of pipelined design? Determine critical path!

Tc_pipelined = **max** of

(tpcq + tmem + tsetup)	Fetch
2(tRFread + tsetup)	Decode
(tpcq + 4tmux + tALU + tAND-OR + tsetup)	Execute
(tpcq + tmem + tsetup)	Memory
2(tpcq + tmux + tRFwrite)	Writeback

Note: the 2X is because Decode and Writeback stages both use the register file in each cycle — Each stage gets half of the cycle time (Tc/2) to do their work, or, stated a different way, 2x of their work must fit in 1 cycle

Pipelined Processor with Hazard Protection! Now, what about performance?

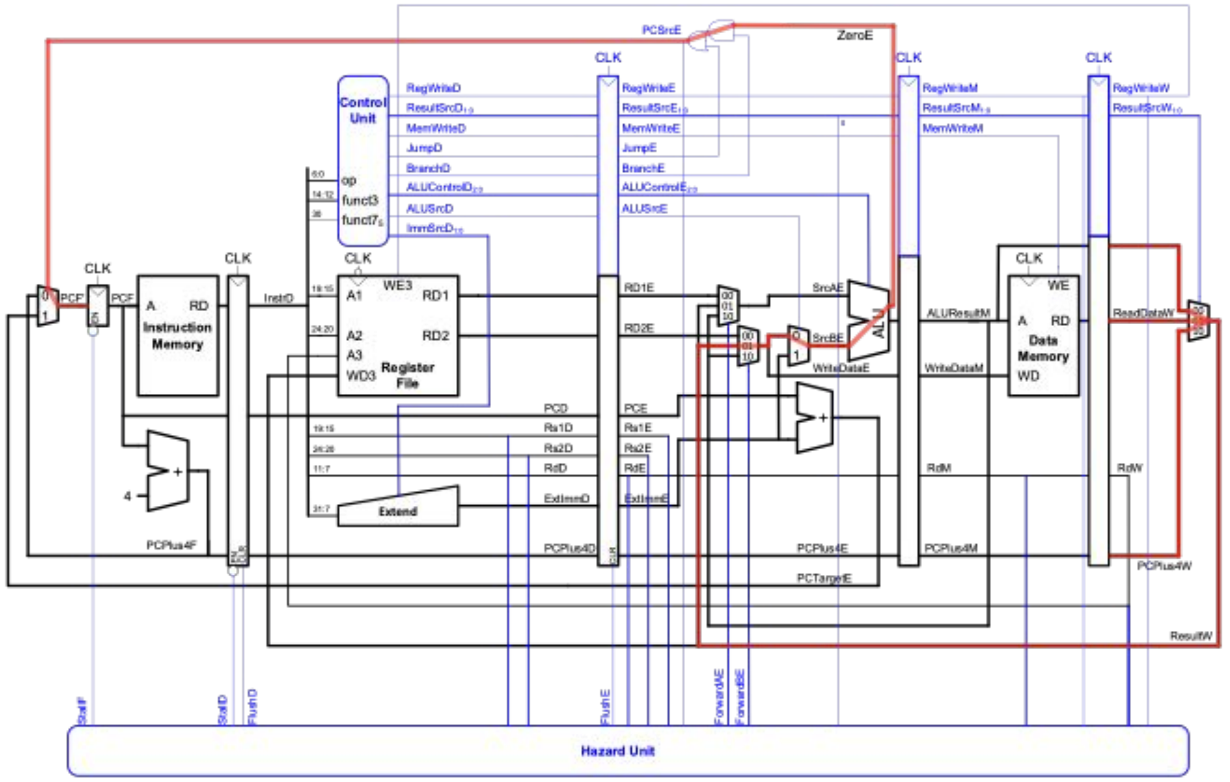
Clock Cycle time of pipelined design? Determine critical path!

Tc_pipelined = **max** of

(tpcq + tmem + tsetup)	Fetch
2(tRFread + tsetup)	Decode
(tpcq + 4tmux + tALU + tAND-OR + tsetup)	**Execute is critical path**
(tpcq + tmem + tsetup)	Memory
2(tpcq + tmux + tRFwrite)	Writeback

Note: the 2X is because Decode and Writeback stages both use the register file in each cycle — Each stage gets half of the cycle time (Tc/2) to do their work, or, stated a different way, 2x of their work must fit in 1 cycle

Pipelined Processor with Hazard Protection! Now, what about performance?



Pipelined Processor with Hazard Protection! Now, what about performance?

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	40
Register setup	t_{setup}	50
Multiplexer	t_{mux}	30
AND-OR gate	$t_{\mathrm{AND\text{-}OR}}$	20
ALU	t_{ALU}	120
Decoder (Control Unit)	t_{dec}	25
Extend unit	t_{dec}	35
Memory read	t_{mem}	200
Register file read	$t_{RF\mathrm{read}}$	100
Register file setup	$t_{RF\mathrm{setup}}$	60

$$T_{c\_pipelined} = t_{pcq} + 4t_{\mathrm{mux}} + t_{\mathrm{ALU}} + t_{AND\text{-}OR} + t_{\mathrm{setup}}$$
$$= (40 + 4*30 + 120 + 20 + 50)\ \mathrm{ps} = \mathbf{350\ ps}$$

Pipelined Processor with Hazard Protection! Now, what about performance?

We have an estimated CPI, a clock cycle time, let's consider our example program!

Check-in Activity, in pairs: What is the program execution time of a program with 100 billion instructions with our pipelined design (assume SPEC CPI)? How much faster than single-cycle?

Repeat for extreme cases:

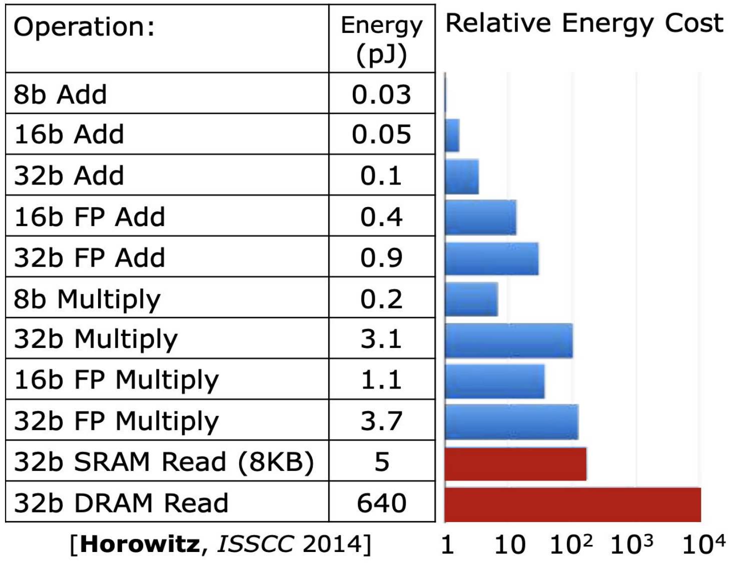
(1) If we **always** mispredict branches and we **always** have a dependency following `lw` – in this case, does pipelined design do better or worse than single cycle?

(2) If we **never** mispredict branches and we **never** have a dependency following `lw` – in this case, does pipelined design do better or worse than single cycle?

Computer design question to worry about over break

- We have been using an ideal (super unrealistic) memory array
- How can we maintain program performance if memory access actually takes at least ***10-100x*** longer than our ALU?

	Capacity	Read Latency (approx)
RegFile	< 1KB	10-100ps
SRAM	1-64MB	1-10ns
DRAM	4-64GB	~100ns (maybe more)
Flash, SSD, hard drive	256-1024GB	~100us



Operation:	Energy (pJ)	Relative Energy Cost
8b Add	0.03	
16b Add	0.05	
32b Add	0.1	
16b FP Add	0.4	
32b FP Add	0.9	
8b Multiply	0.2	
32b Multiply	3.1	
16b FP Multiply	1.1	
32b FP Multiply	3.7	
32b SRAM Read (8KB)	5	
32b DRAM Read	640	

[**Horowitz**, *ISSCC* 2014]

Wrap-Up December 2

SCAN ME

- Coming up next!
  - Realistic Memory Systems; looking ahead to COSC275
- Logistics, Reminders
  - TA help 7-9PM on Sundays, Tuesdays, Thursdays in C107
  - LP Office hours M 9-10:30AM, Th 2:30-4PM
  - After receiving additional requests, weekly exercise solutions will be posted next week (Dec 8) to help with final exam review
  - BRING YOUR PROCESSOR LAB QUESTIONS INTO LAB WEDNESDAY!!
    - Final Lab Report (all stages) due Monday, Dec. 9 10PM
  - **ATTEND FACULTY CANDIDATE TALK TODAY AT 4:30PM in A131, WITH SNACKS @ 4PM in C209**
    - *If you attend at least 3 candidate talks, then send me an email including 1 thing you learned from each talk, I'll give you 5% extra credit on any previous lab report*
  - ***POLL FOR ADVANCED TOPIC!! Vote tonight if you haven't already!!!***
- FEEDBACK
  - https://forms.gle/5Aafcm3iJthX78jx6