Pipelining

Changelog

Changes made in this version not seen in first lecture:

- 5 October 2017: put addq timing slide before critical path slides
- 5 October 2017: slide 25: arrows to fetch/decode registers point to outputs, not inputs
- 5 October 2017: slide 27: sum with no pipelining was 550 ps, not 500 ps
- 5 October 2017: slide 33: e_dstE and W_dstE labels were swapped
- 5 October 2017: slide 34: rA should have been D_rA, e_valA should have been d_valA

logistics

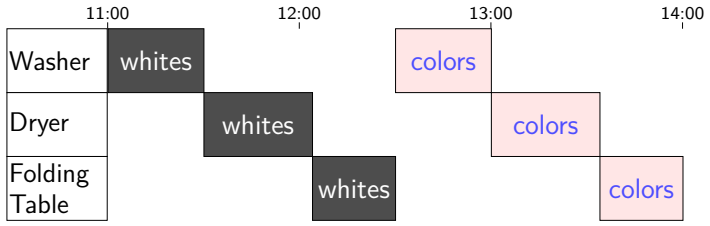
exam graded — scores on gradebook

- keys on Collab
- view exam, submit regrade requests via TPEGS
- note: two questions dropped outside of TPEGS

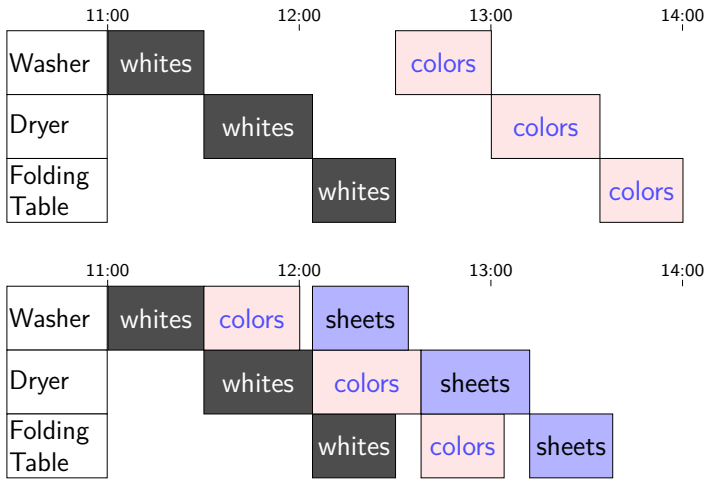
median: 83.5; 25th percentile: 77.3; 75th percentile: 90.7

HCL homework due next Wednesday

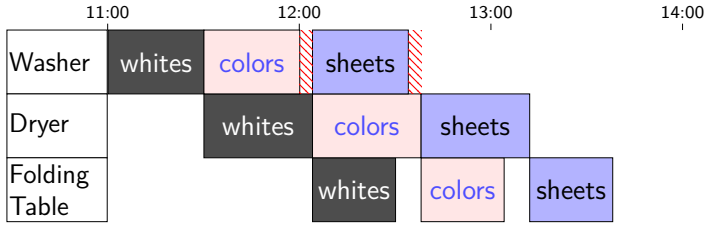
Human pipeline: laundry



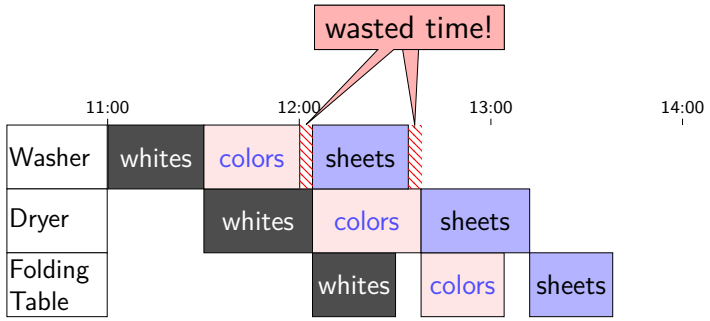
Human pipeline: laundry



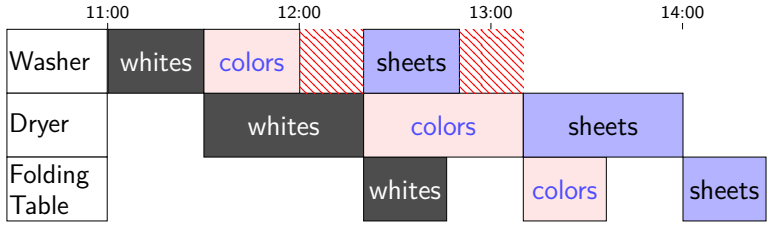
Waste (1)



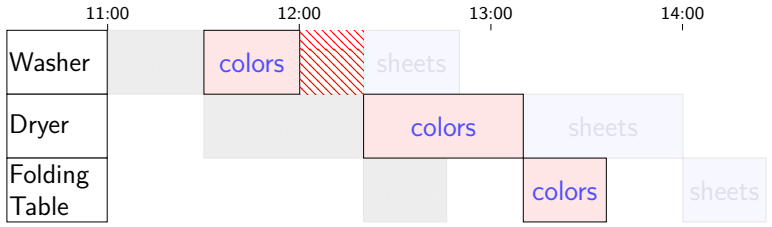
Waste (1)



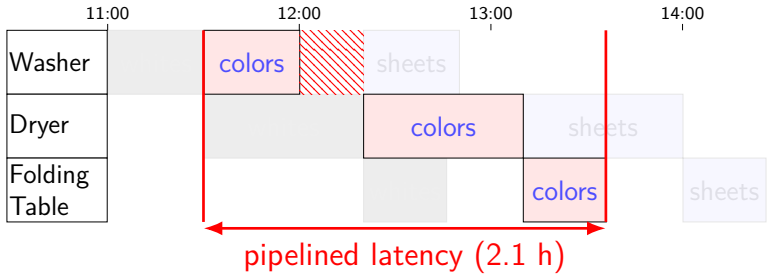
Waste (2)



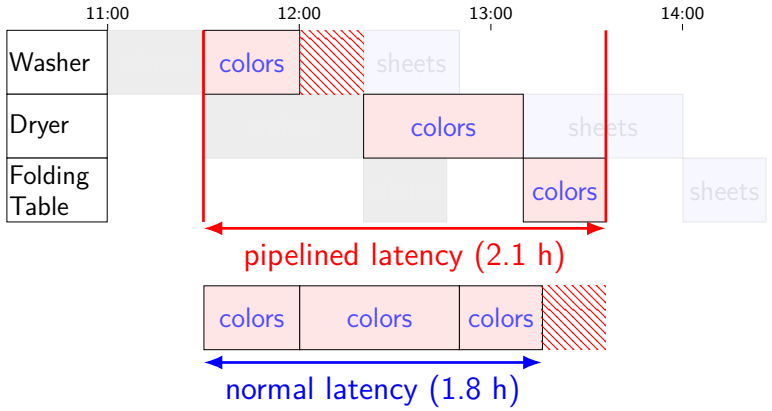
Latency — Time for One



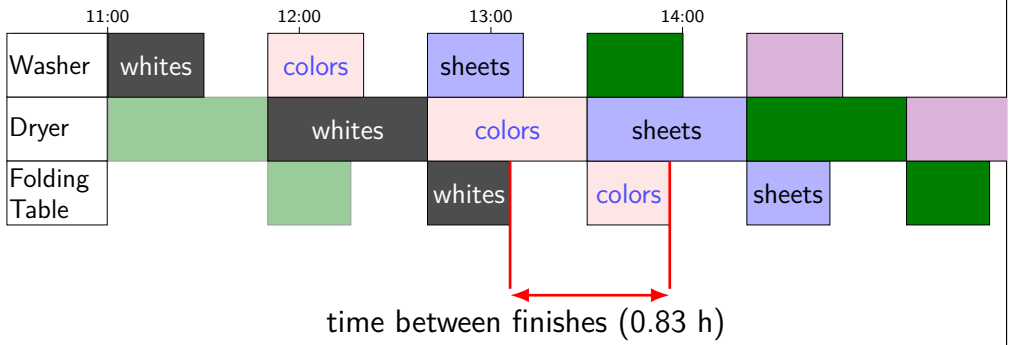
Latency — Time for One



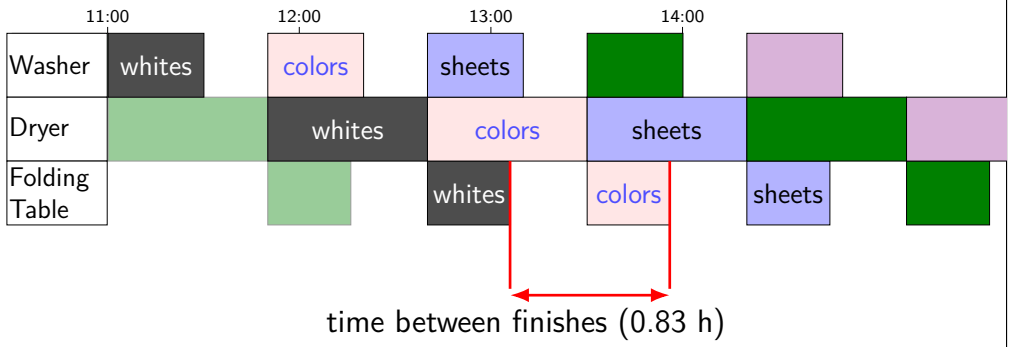
Latency — Time for One



Throughput — Rate of Many

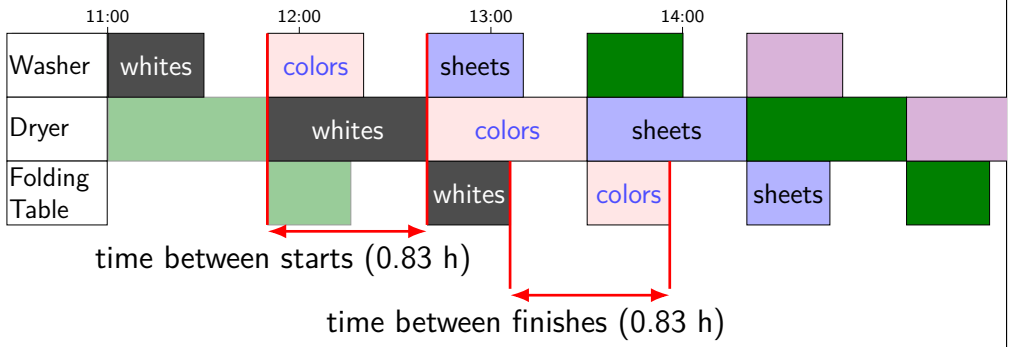


Throughput — Rate of Many



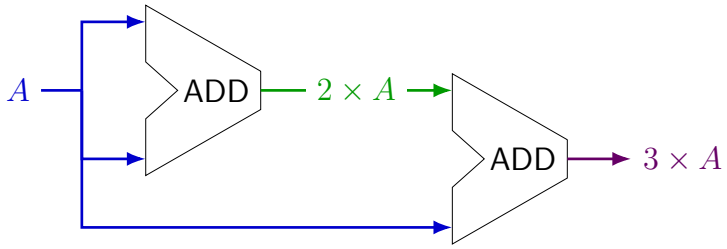
$$\frac{1 \text{ load}}{0.83\text{h}} = 1.2 \text{ loads/h}$$

Throughput — Rate of Many

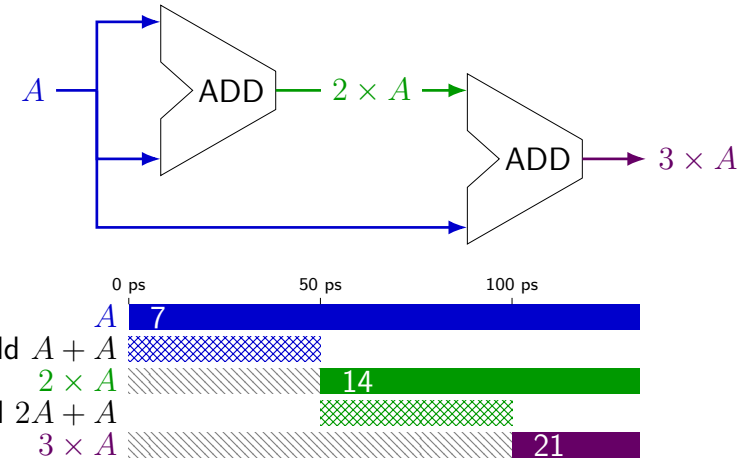


$$\frac{1 \text{ load}}{0.83\text{h}} = 1.2 \text{ loads/h}$$

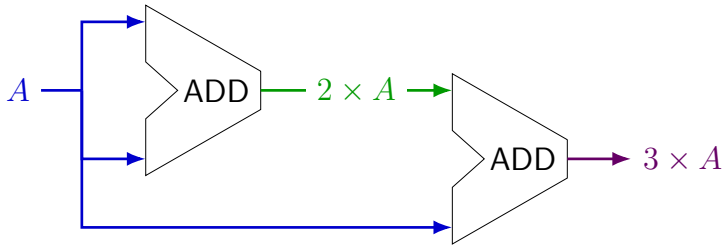
times three circuit



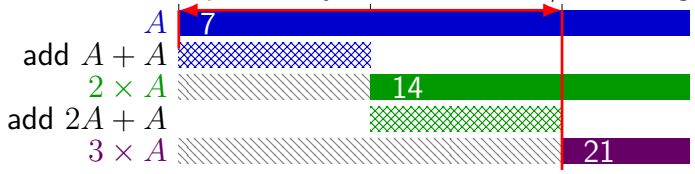
times three circuit



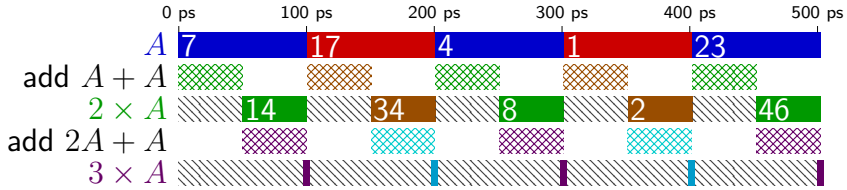
times three circuit



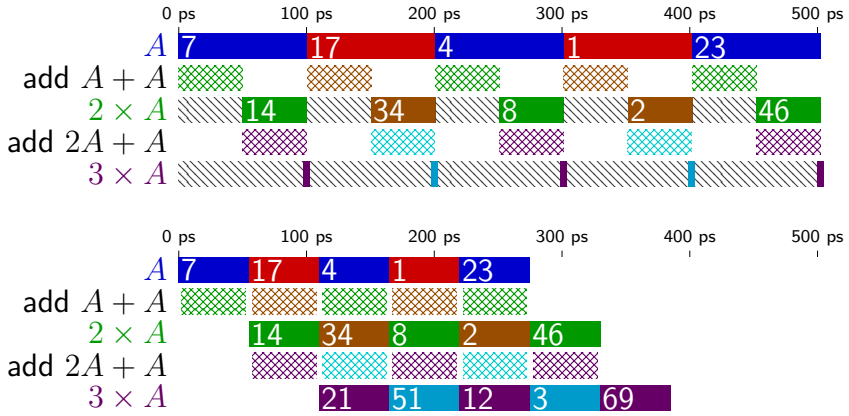
100 ps latency $\Longrightarrow$ 10 results/ns throughput



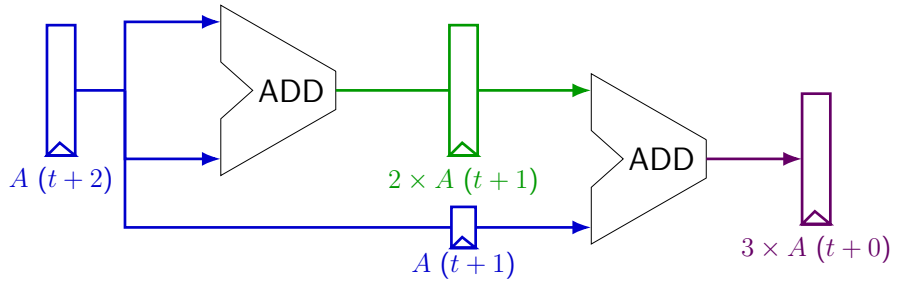
times three and repeat



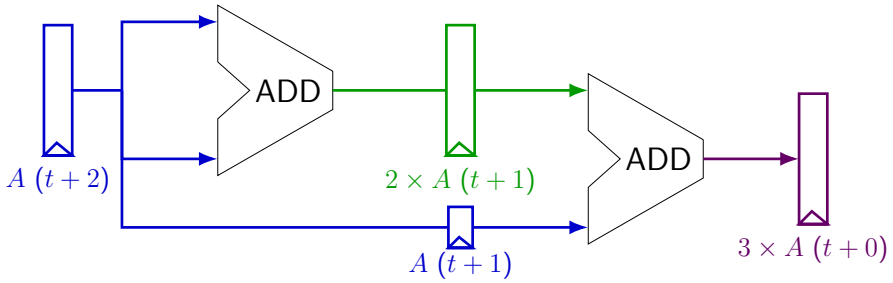
times three and repeat

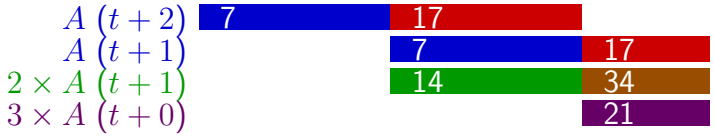


pipelined times three

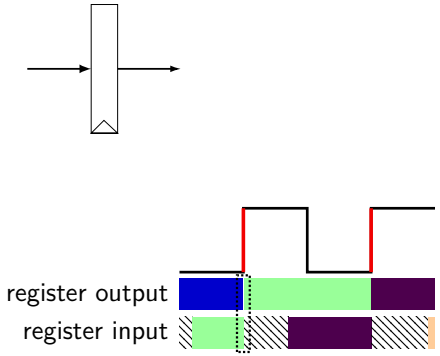


pipelined times three

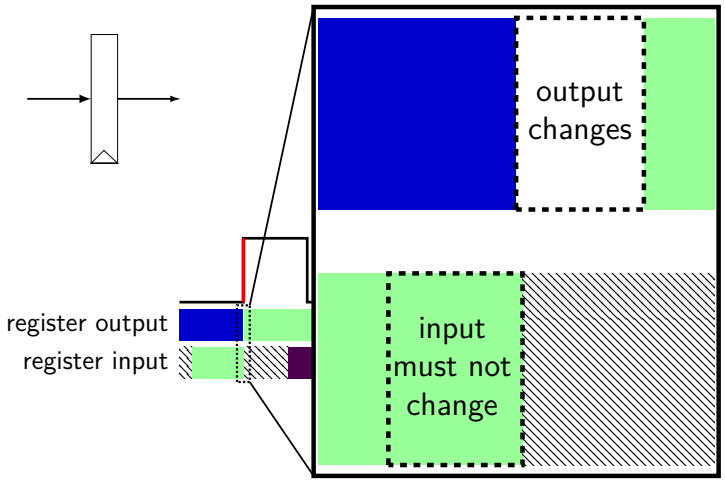




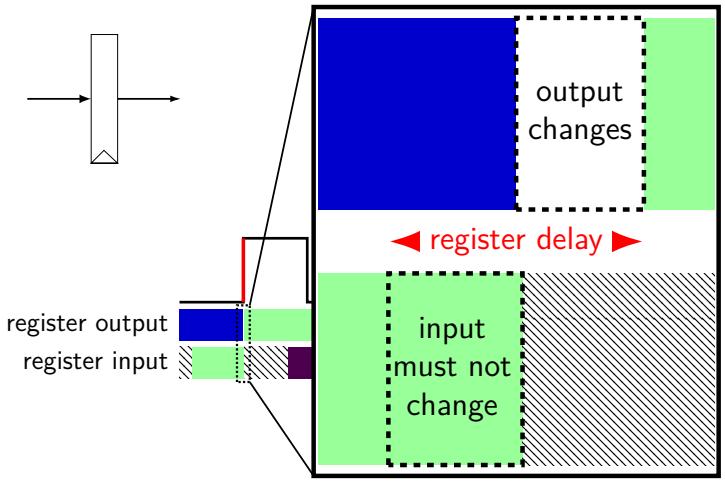
register tolerances



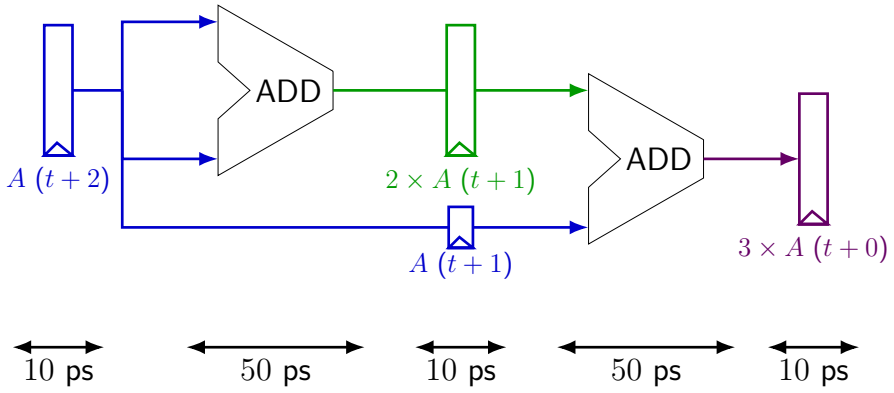
register tolerances



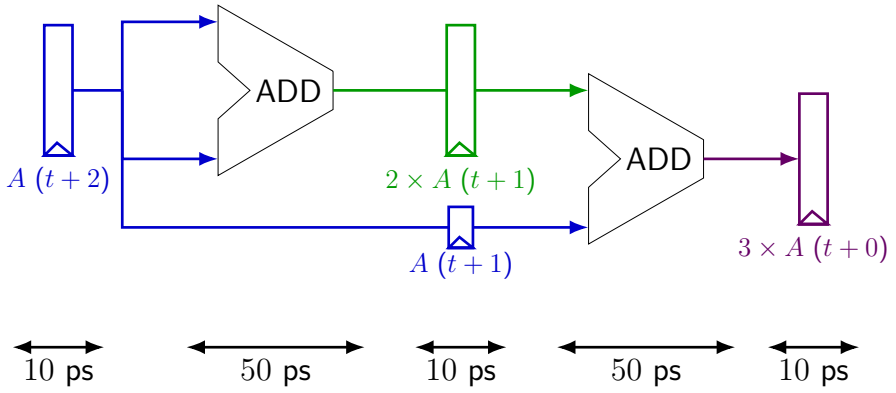
register tolerances



times three pipeline timing

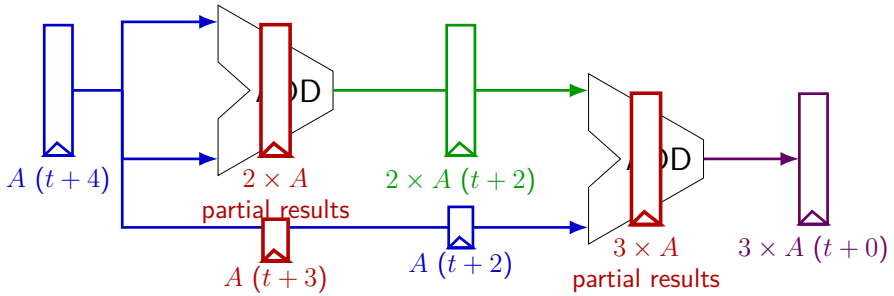


times three pipeline timing

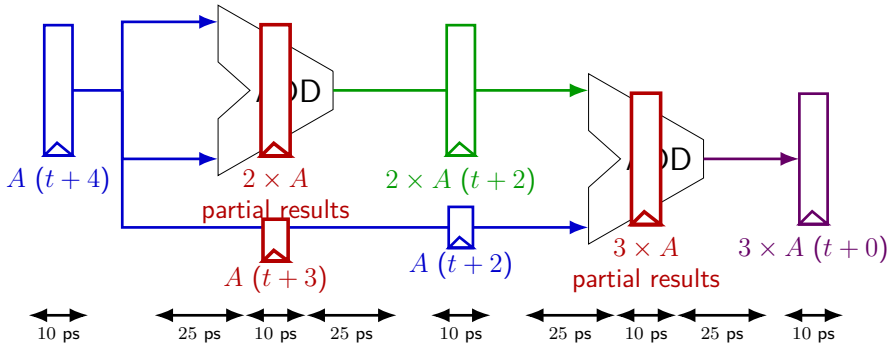


throughput: $\frac{1}{60\text{ ps}} \approx 16$ G operations/sec

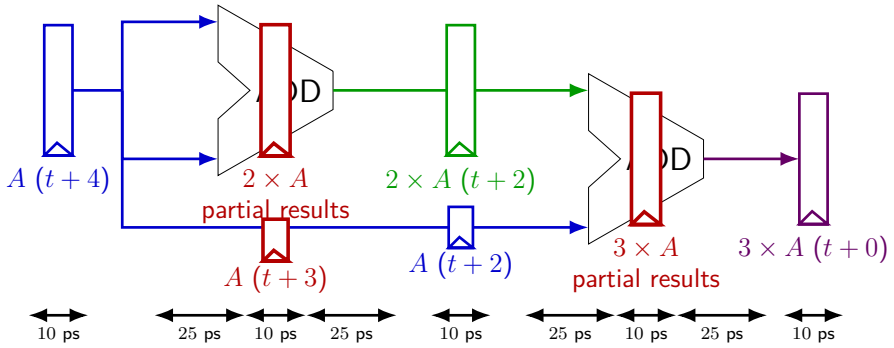
deeper pipeline



deeper pipeline

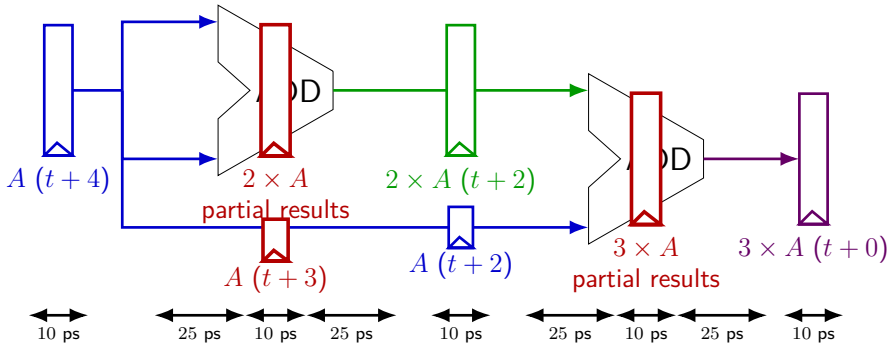


deeper pipeline



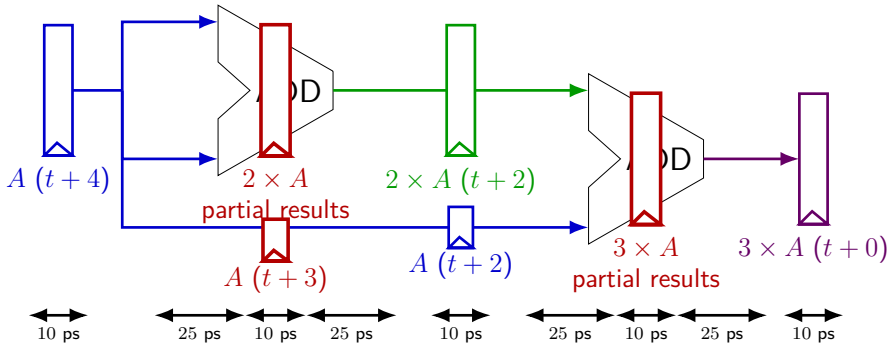
exercise: throughput now?

deeper pipeline



throughput: $\frac{1}{35\ \text{ps}} \approx 28\ \text{G ops/sec}$

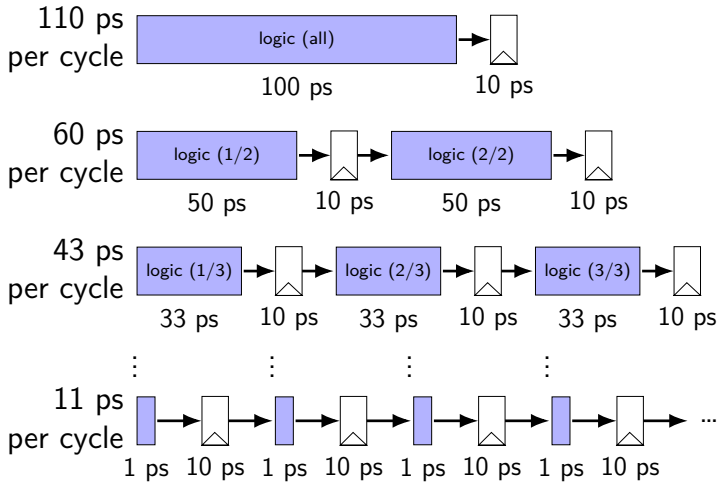
deeper pipeline



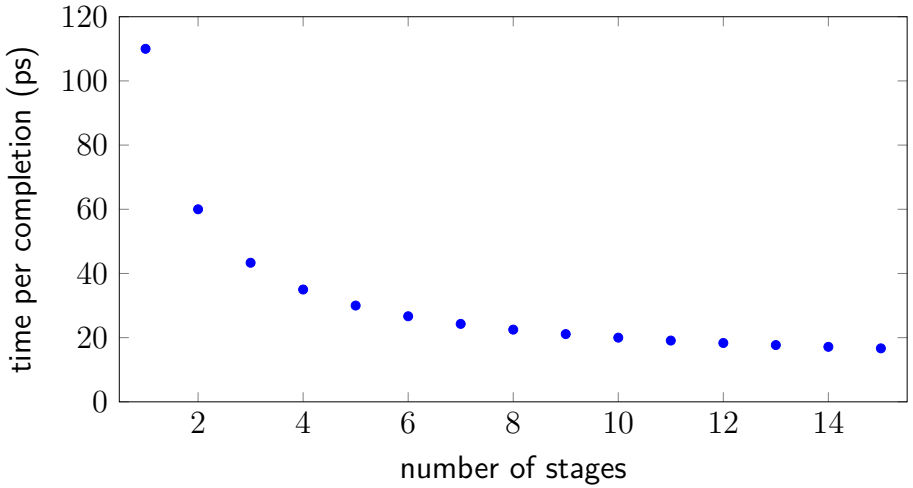
Problem: How much faster can we get?

Problem: Can we even do this?

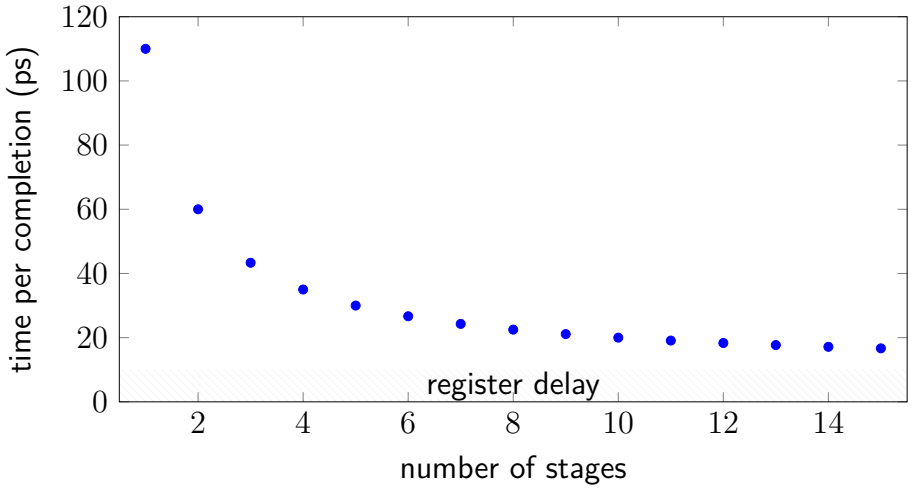
diminishing returns: register delays



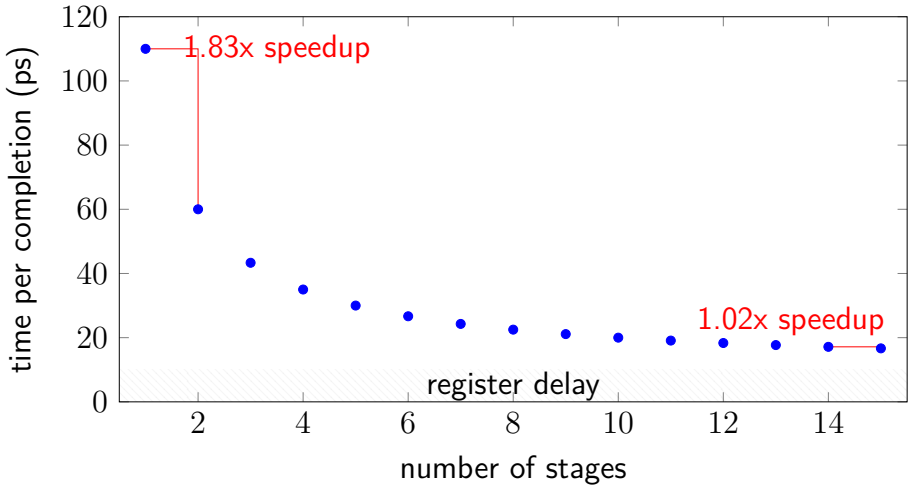
diminishing returns: register delays



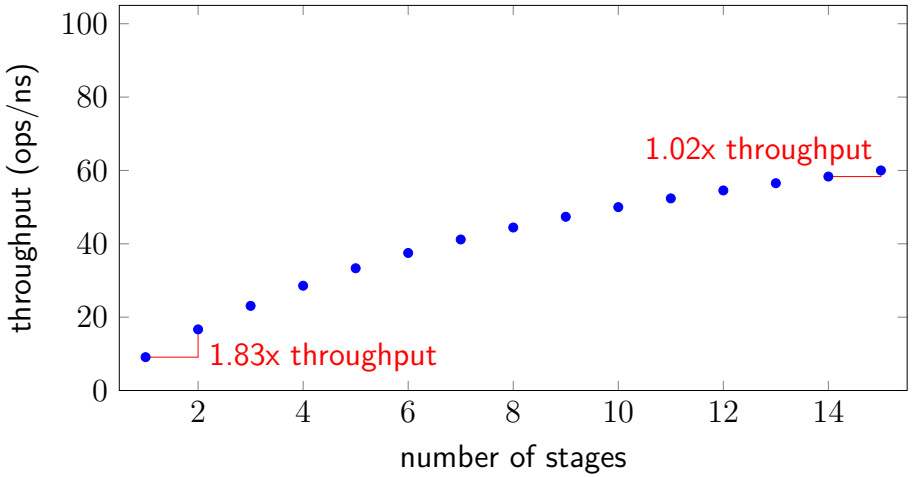
diminishing returns: register delays



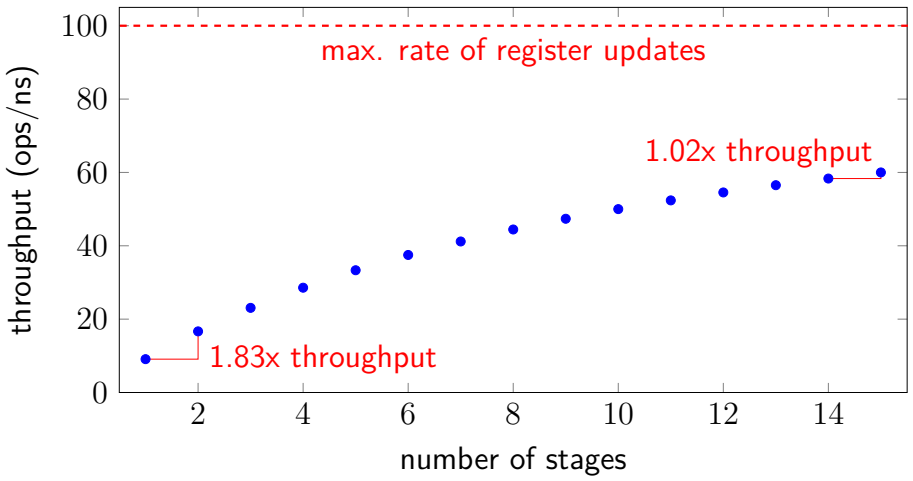
diminishing returns: register delays



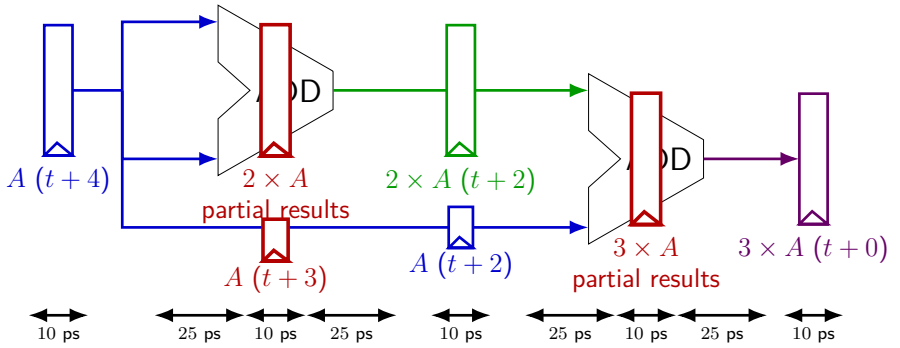
diminishing returns: register delays



diminishing returns: register delays



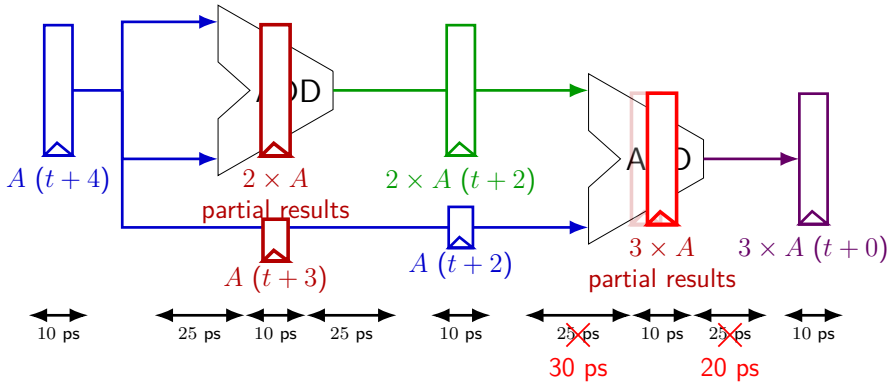
deeper pipeline



Problem: How much faster can we get?

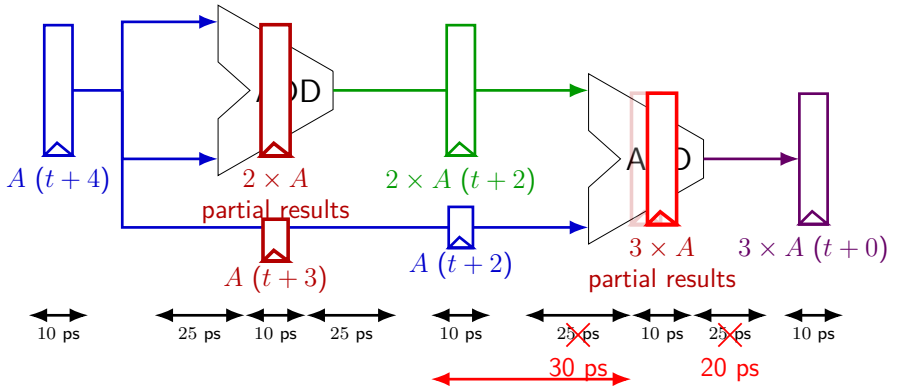
Problem: Can we even do this?

deeper pipeline



exercise: throughput now? (didn't split second add evenly)

deeper pipeline

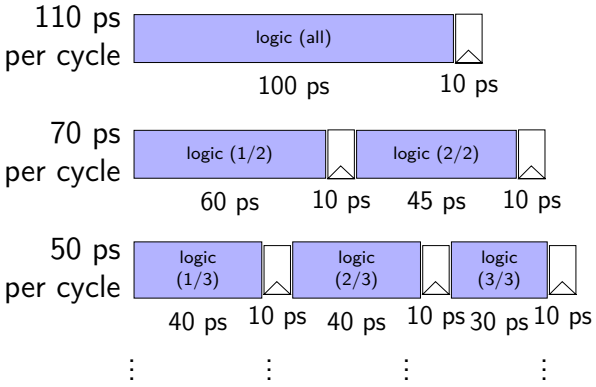


throughput: $\frac{1}{40\text{ ps}} \approx 25$ G ops/sec

diminishing returns: uneven split

Can we split up some logic (e.g. adder) arbitrarily?

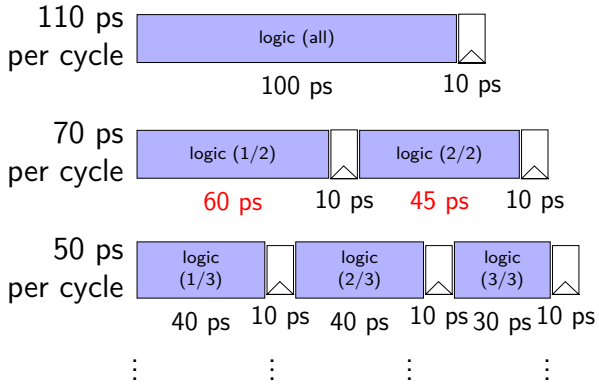
Probably not...



diminishing returns: uneven split

Can we split up some logic (e.g. adder) arbitrarily?

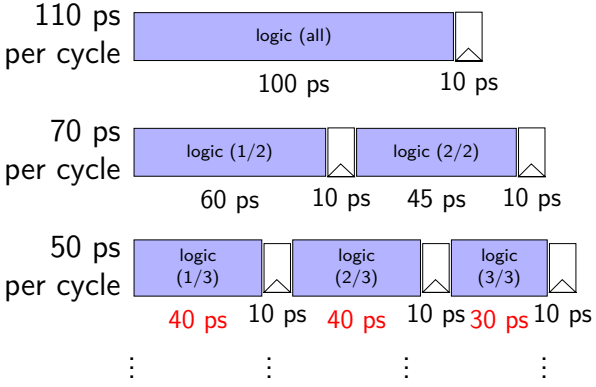
Probably not...



diminishing returns: uneven split

Can we split up some logic (e.g. adder) arbitrarily?

Probably not...



textbook SEQ 'stages'

conceptual order only

Fetch: read instruction memory

Decode: read register file

Execute: arithmetic (ALU)

Memory: read/write data memory

Writeback: write register file

PC Update: write PC register

textbook SEQ 'stages'

conceptual order only

Fetch: read instruction memory

Decode: read register file

Execute: arithmetic (ALU)

Memory: read/write data memory

Writeback: write register file

PC Update: write PC register

writes happen
at end of cycle

textbook SEQ 'stages'

conceptual order only

Fetch: read instruction memory

Decode: read register file

Execute: arithmetic (ALU)

Memory: read/write data memory

Writeback: write register file

PC Update: write PC register

reads — "magic"
like combinatorial logic
as values available

textbook stages

~~conceptual order only~~ pipeline stages

Fetch/PC Update: read instruction memory;
compute next PC

Decode: read register file

Execute: arithmetic (ALU)

Memory: read/write data memory

Writeback: write register file

textbook stages

~~conceptual order only~~ pipeline stages

Fetch/PC Update: read instruction memory;
compute next PC

Decode: read register file

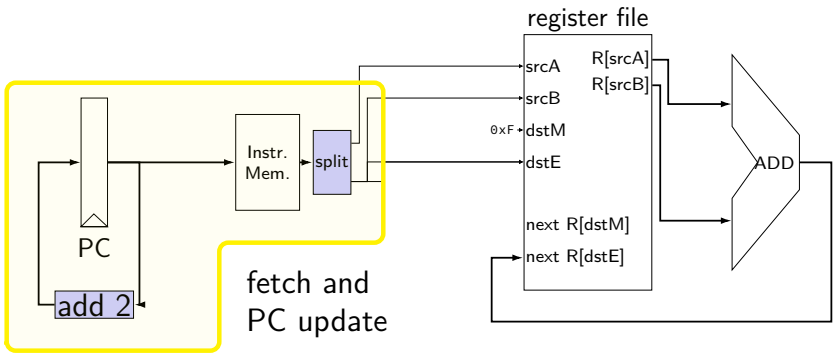
Execute: arithmetic (ALU)

Memory: read/write data memory

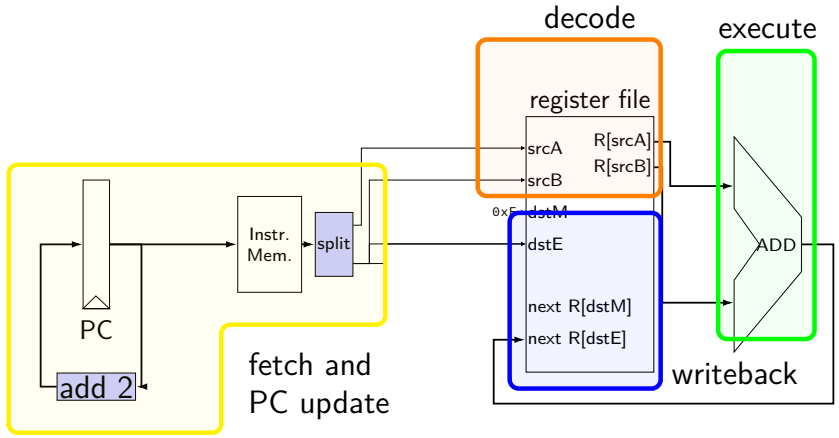
Writeback: write register file

5 stages
one instruction in each
compute next to start immediatelly

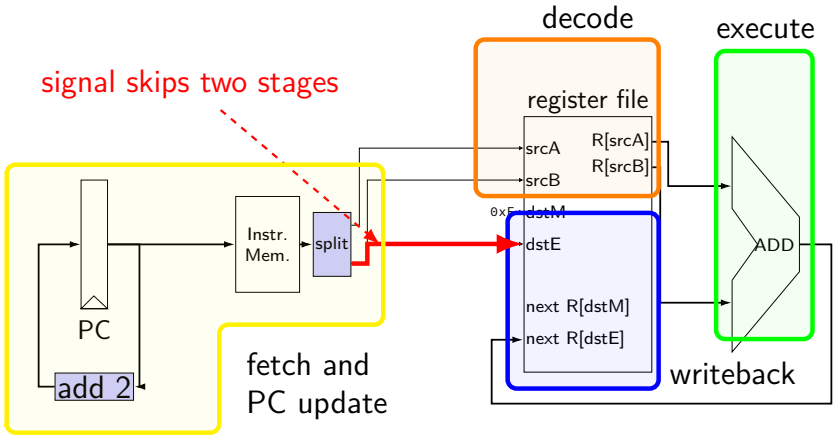
addq CPU



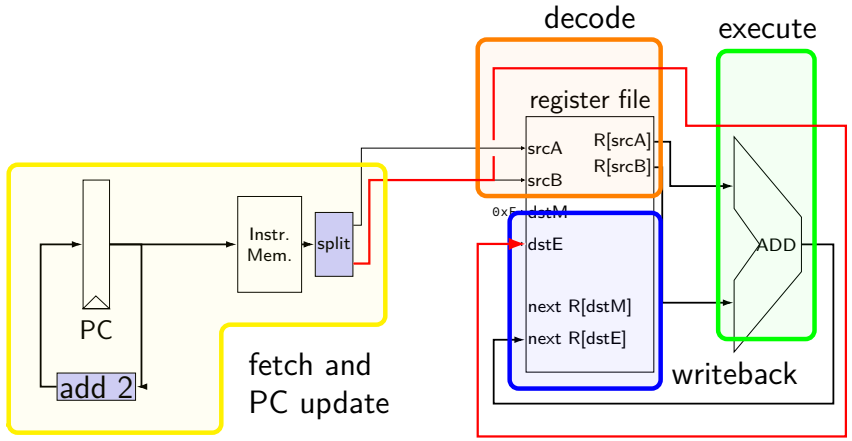
addq CPU



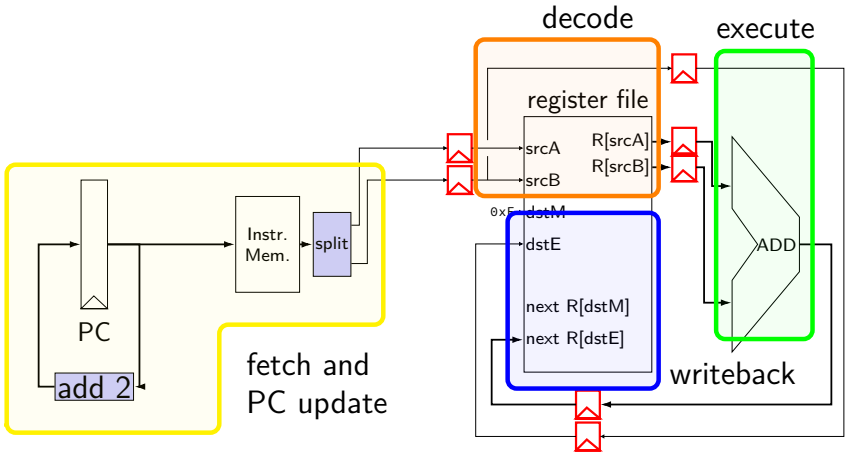
addq CPU



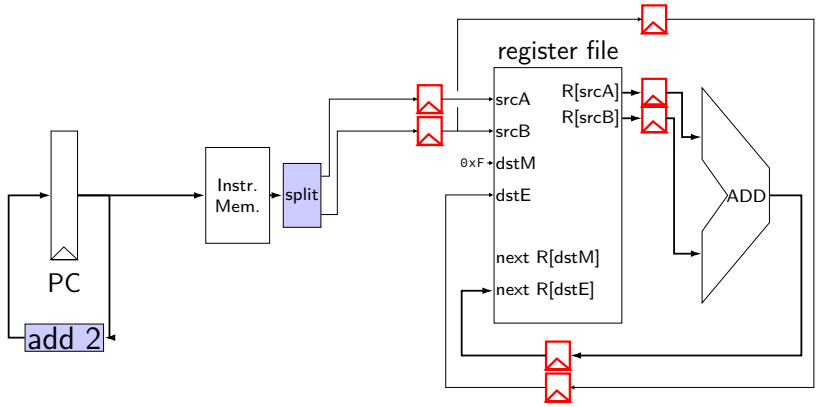
addq CPU



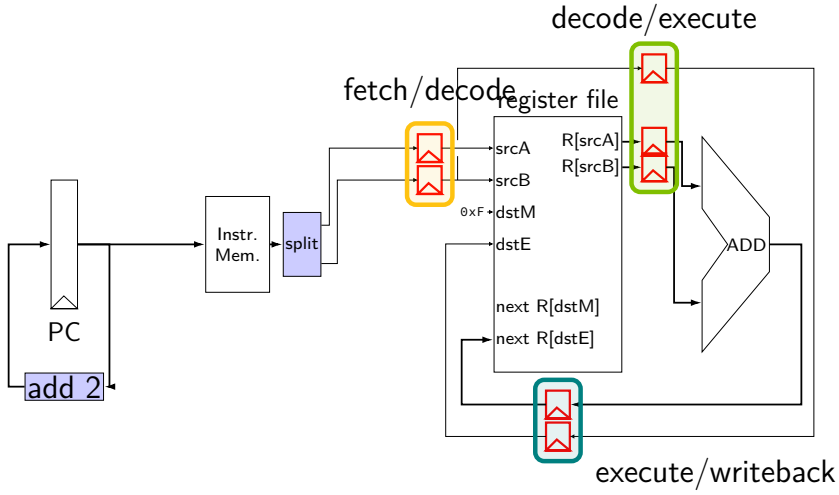
pipelined addq processor



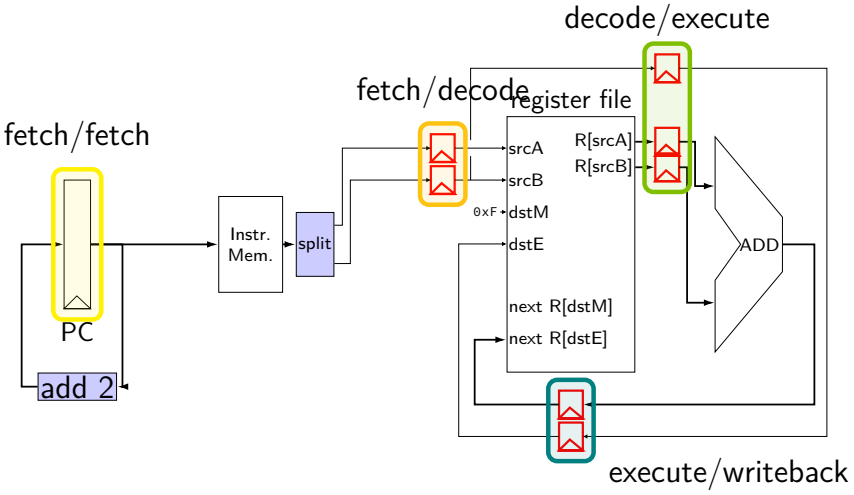
pipelined addq processor



pipelined addq processor

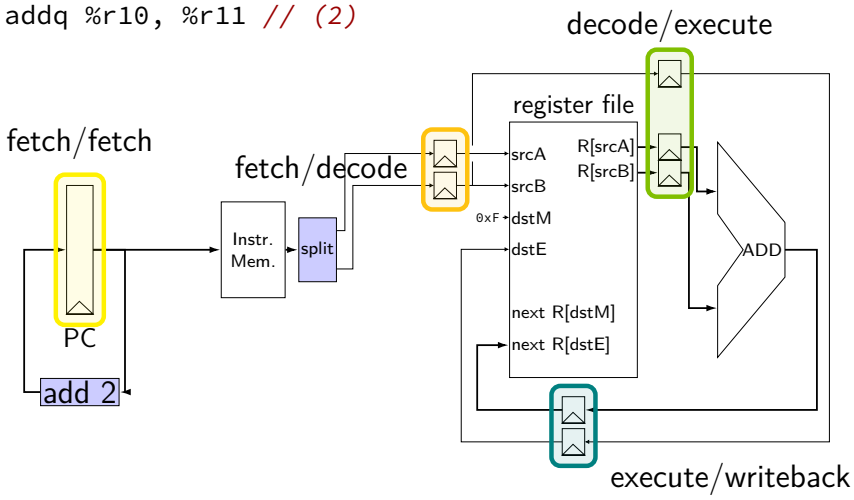


pipelined addq processor



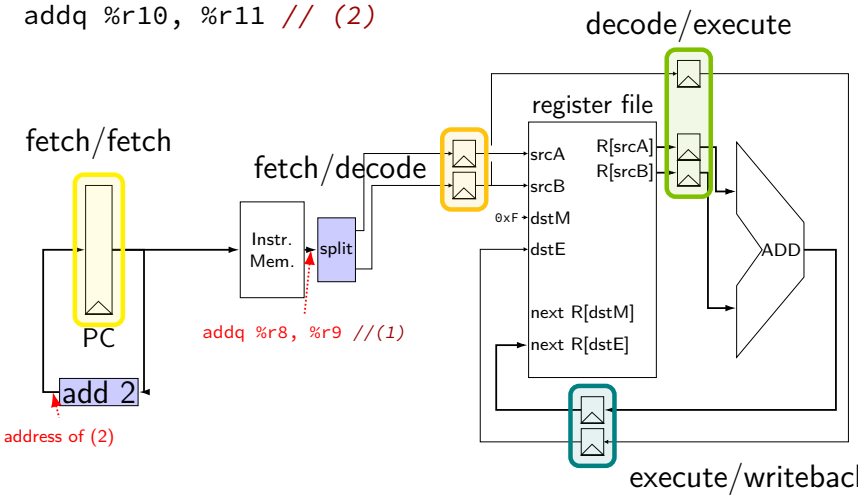
addq execution

```
addq %r8, %r9  // (1)
addq %r10, %r11 // (2)
```



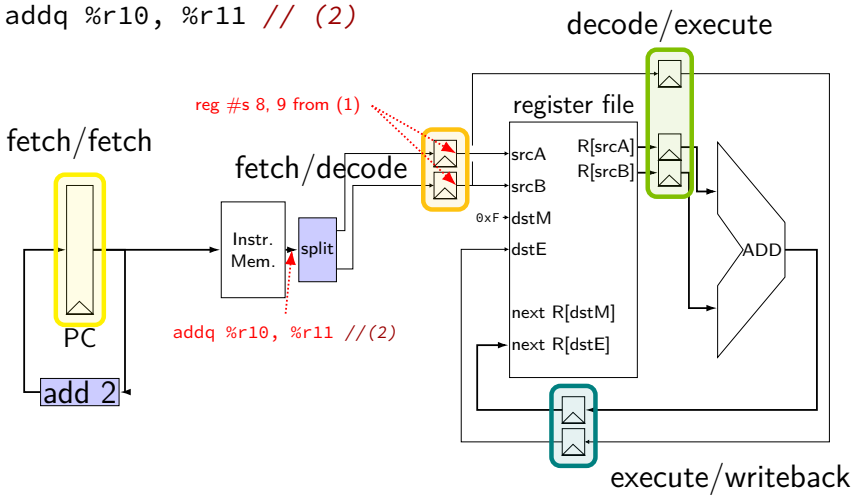
addq execution

```
addq %r8, %r9  // (1)
addq %r10, %r11 // (2)
```



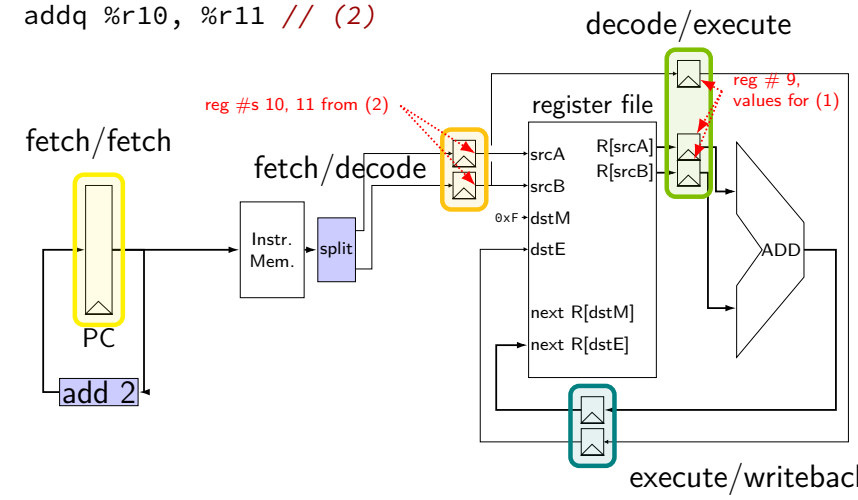
addq execution

```
addq %r8, %r9  // (1)
addq %r10, %r11 // (2)
```



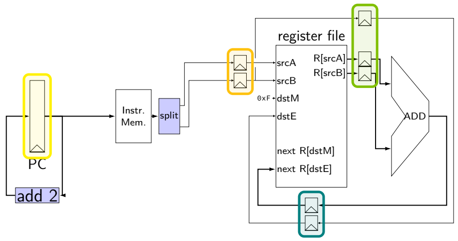
addq execution

```
addq %r8, %r9  // (1)
addq %r10, %r11 // (2)
```



addq processor timing

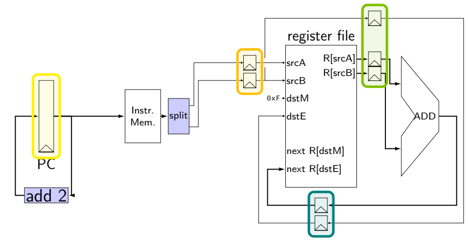
```
// initially %r8 = 800,
//           %r9 = 900, etc.
addq %r8, %r9
addq %r10, %r11
addq %r12, %r13
addq %r9, %r8
```



	fetch	fetch/decode		decode/execute			execute/writeback	
cycle	PC	rA	rB	R[srcA]	R[srcB]	dstE	next R[dstE]	dstE
0	0x0							
1	0x2	8	9					
2	0x4	10	11	800	900	9		
3	0x6	12	13	1000	1100	11	1700	9
4		9	8	1200	1300	13	2100	11
5				1700	800	8	2500	13
6							2500	8

addq processor timing

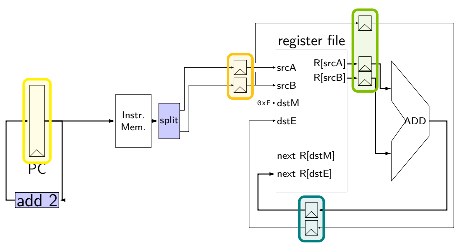
```
// initially %r8 = 800,
//           %r9 = 900, etc.
addq %r8, %r9
addq %r10, %r11
addq %r12, %r13
addq %r9, %r8
```



	fetch	fetch/decode		decode/execute			execute/writeback	
cycle	PC	rA	rB	R[srcA]	R[srcB]	dstE	next R[dstE]	dstE
0	0x0							
1	0x2	8	9					
2	0x4	10	11	800	900	9		
3	0x6	12	13	1000	1100	11	1700	9
4		9	8	1200	1300	13	2100	11
5				1700	800	8	2500	13
6							2500	8

addq processor timing

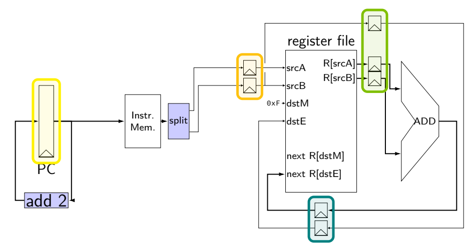
```
// initially %r8 = 800,
//           %r9 = 900, etc.
addq %r8, %r9
addq %r10, %r11
addq %r12, %r13
addq %r9, %r8
```



	fetch	fetch/decode		decode/execute			execute/writeback	
cycle	PC	rA	rB	R[srcA]	R[srcB]	dstE	next R[dstE]	dstE
0	0x0							
1	0x2	8	9					
2	0x4	10	11	800	900	9		
3	0x6	12	13	1000	1100	11	1700	9
4		9	8	1200	1300	13	2100	11
5				1700	800	8	2500	13
6							2500	8

addq processor timing

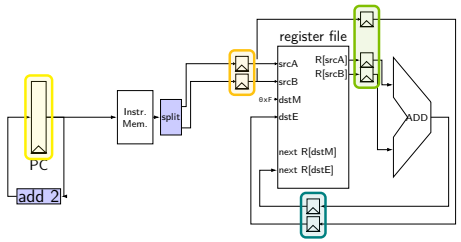
```
// initially %r8 = 800,
//           %r9 = 900, etc.
addq %r8, %r9
addq %r10, %r11
addq %r12, %r13
addq %r9, %r8
```



	fetch	fetch/decode		decode/execute			execute/writeback	
cycle	PC	rA	rB	R[srcA]	R[srcB]	dstE	next R[dstE]	dstE
0	0x0							
1	0x2	8	9					
2	0x4	10	11	800	900	9		
3	0x6	12	13	1000	1100	11	1700	9
4		9	8	1200	1300	13	2100	11
5				1700	800	8	2500	13
6							2500	8

addq processor timing

```
// initially %r8 = 800,
//           %r9 = 900, etc.
addq %r8, %r9
addq %r10, %r11
addq %r12, %r13
addq %r9, %r8
```

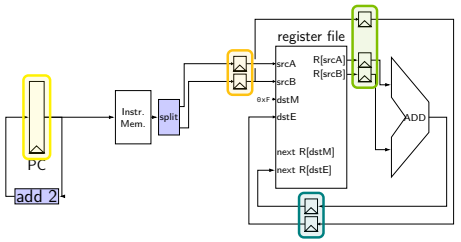


	fetch	fetch/decode		decode/execute			execute/writeback	
cycle	PC	rA	rB	R[srcA]	R[srcB]	dstE	next R[dstE]	dstE
0	0x0							
1	0x2	8	9					
2	0x4	10	11	800	900	9		
3	0x6	12	13	1000	1100	11	1700	9
4		9	8	1200	1300	13	2100	11
5				1700	800	8	2500	13
6							2500	8

addq processor performance

example delays:

path	time
add 2	80 ps
instruction memory	200 ps
register file read	125 ps
add	100 ps
register file write	125 ps



no pipelining: 1 instruction per 550 ps

add up everything but add 2 (critical (slowest) path)

pipelining: 1 instruction per 200 ps + pipeline register delays

slowest path through stage + pipeline register delays

latency: 800 ps + pipeline register delays (4 cycles)

critical path

every path from state output to state input needs enough time

output — may change on rising edge of clock

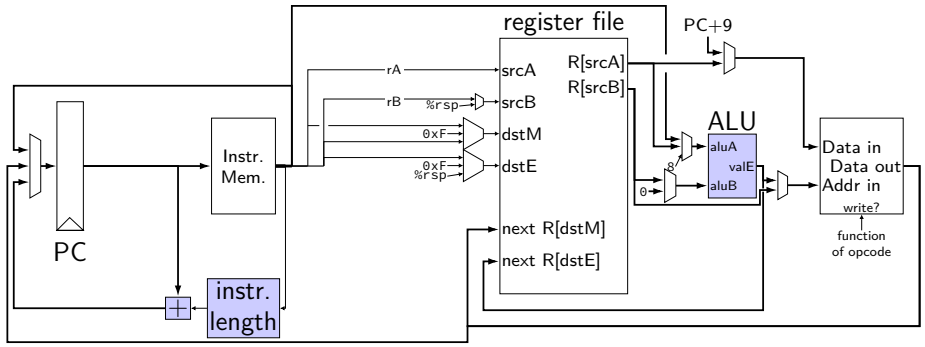
input — must be stable sufficiently before rising edge of clock

critical path: slowest of all these paths — determines cycle time

times three: slowest part of ALU ended up mattering

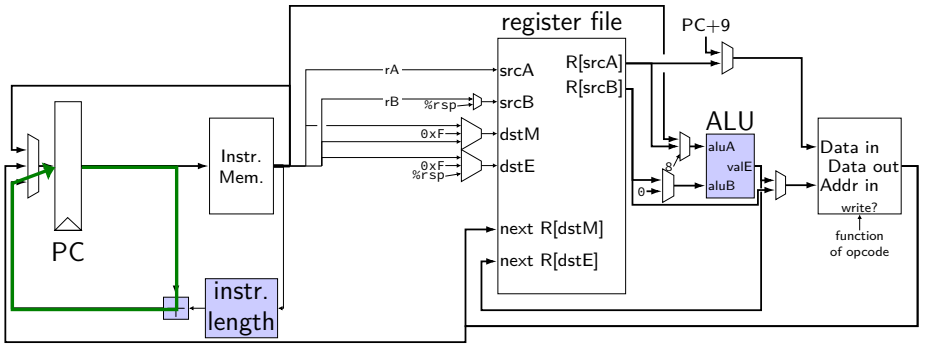
matters with or without pipelining

SEQ paths



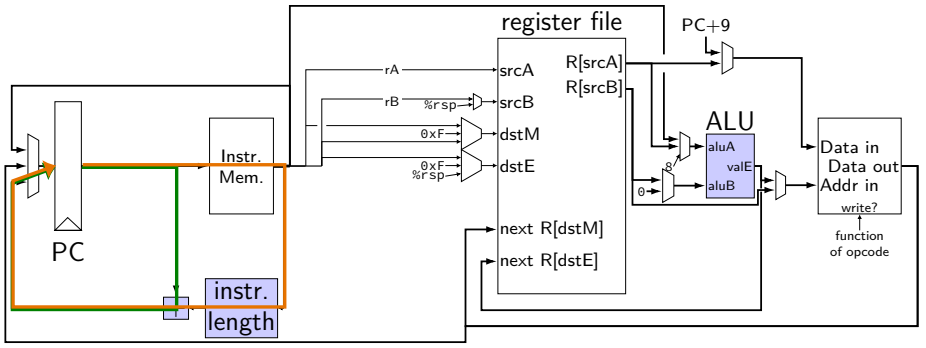
SEQ paths

path 1: 25 picoseconds



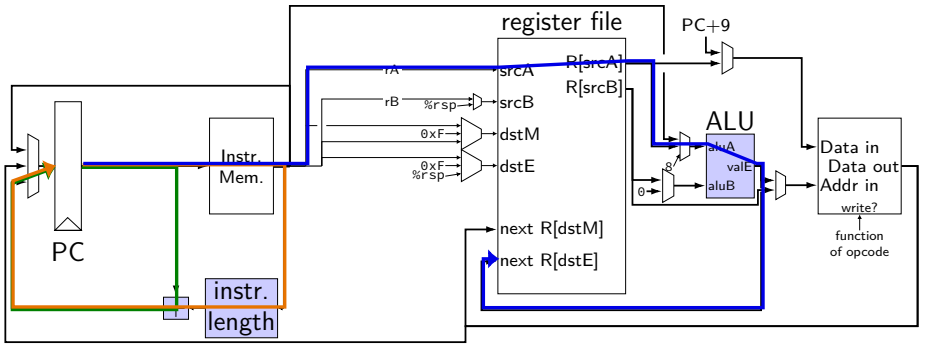
SEQ paths

path 1: 25 picoseconds path 2: 50 picoseconds



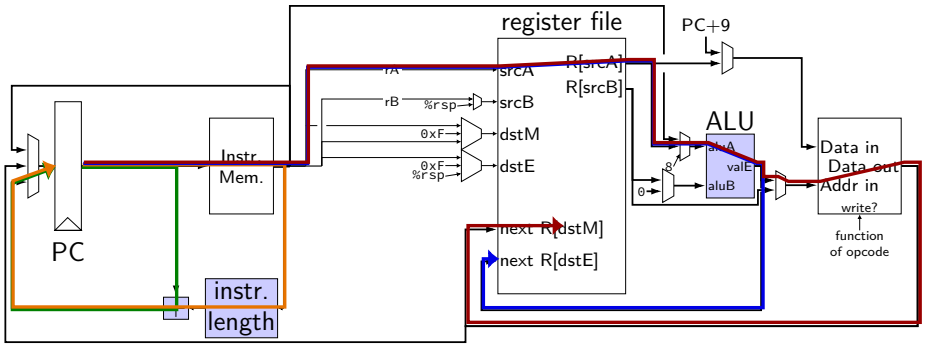
SEQ paths

path 1: 25 picoseconds path 2: 50 picoseconds
path 3: 400 picoseconds



SEQ paths

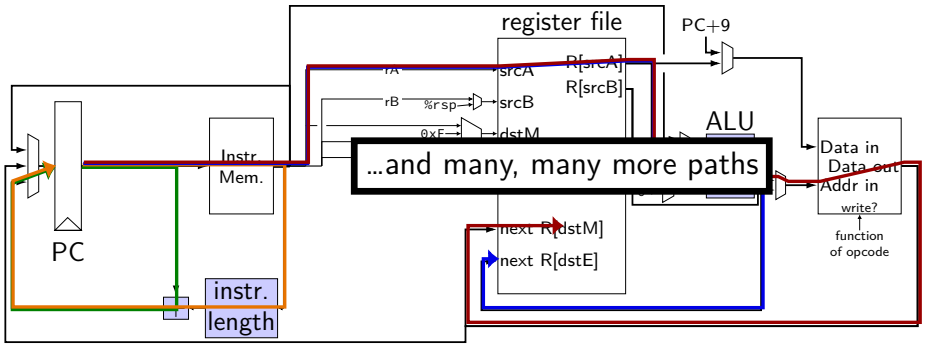
path 1: 25 picoseconds path 2: 50 picoseconds
path 3: 400 picoseconds path 4: 900 picoseconds
... ...



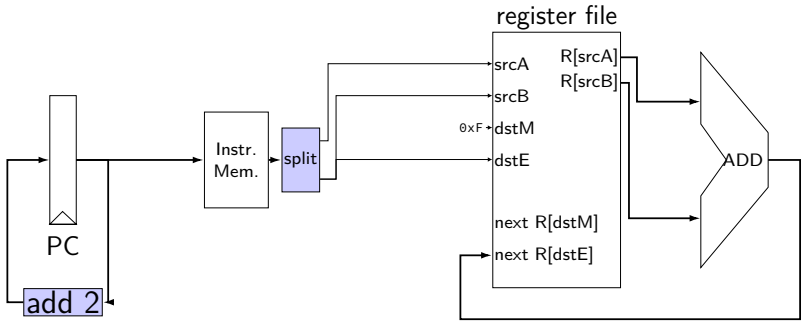
SEQ paths

path 1: 25 picoseconds
path 3: 400 picoseconds
...

path 2: 50 picoseconds
path 4: 900 picoseconds
...

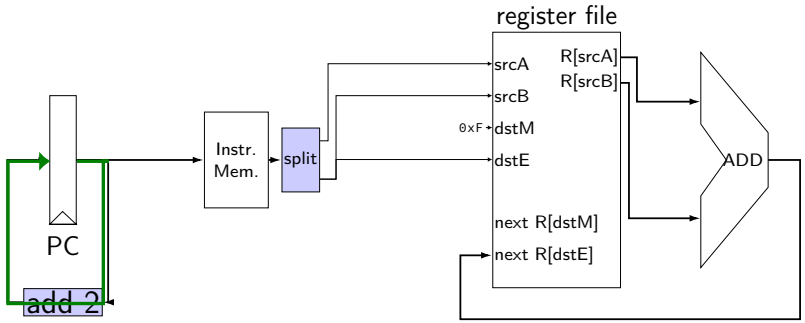


sequential addq paths



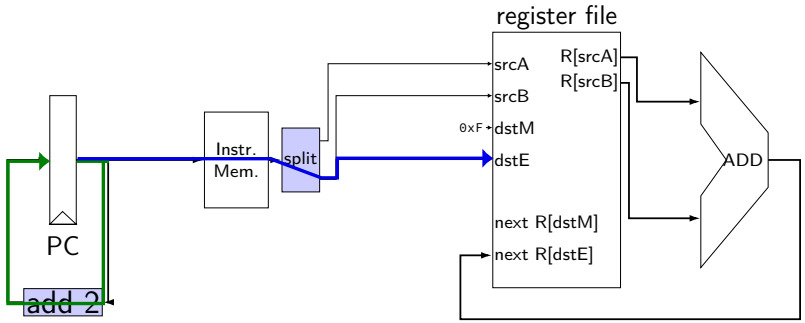
sequential addq paths

path 1: 25 picoseconds



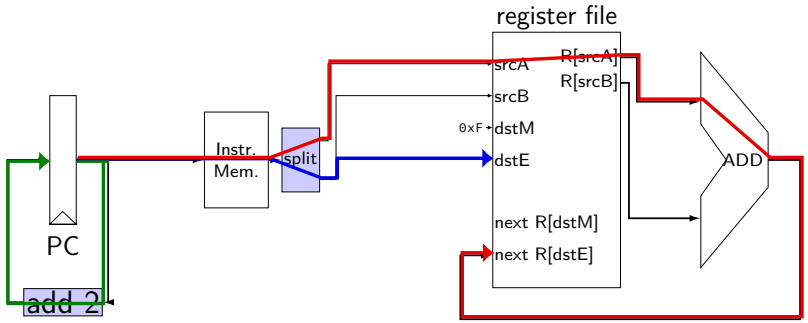
sequential addq paths

path 1: 25 picoseconds
path 2: 375 picoseconds



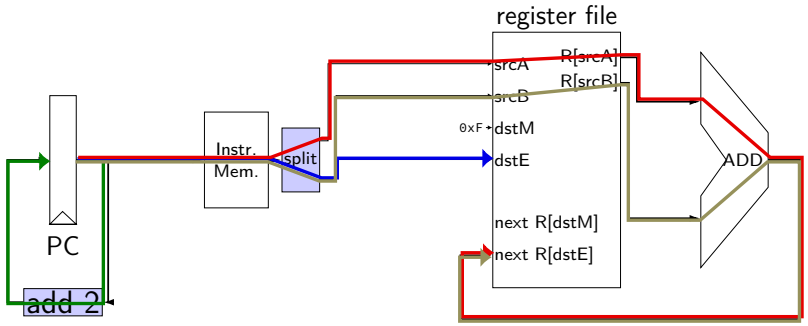
sequential addq paths

path 1: 25 picoseconds
path 2: 375 picoseconds
path 3: 500 picoseconds



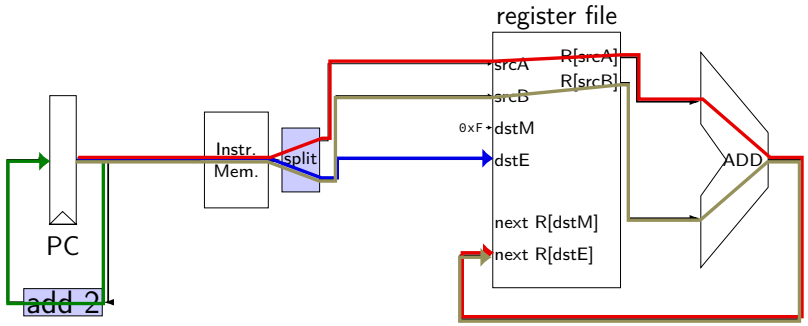
sequential addq paths

path 1: 25 picoseconds
path 2: 375 picoseconds
path 3: 500 picoseconds
path 4: 500 picoseconds

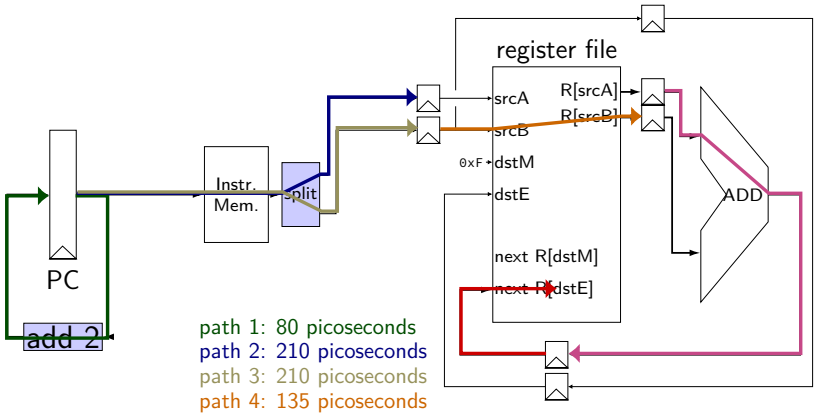


sequential addq paths

path 1: 25 picoseconds
path 2: 375 picoseconds
path 3: **500 picoseconds**
path 4: **500 picoseconds**
overall cycle time: **500 picoseconds** (longest path)

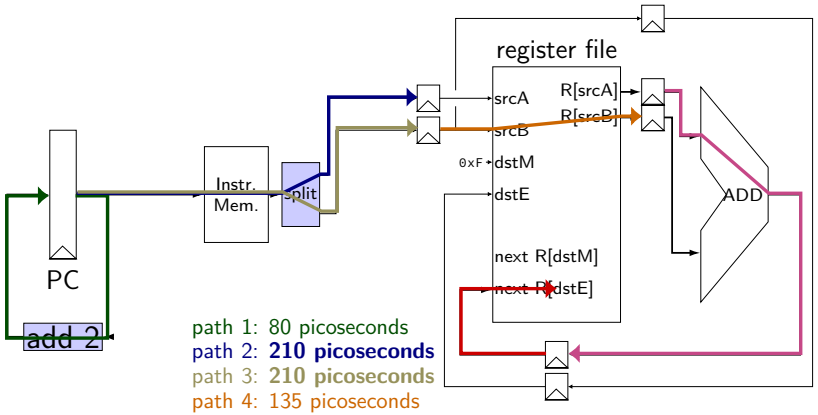


pipelined addq paths



path 1: 80 picoseconds
path 2: 210 picoseconds
path 3: 210 picoseconds
path 4: 135 picoseconds
path 5: 110 picoseconds
path 6: 135 picoseconds
...
overall cycle time: 210 picoseconds

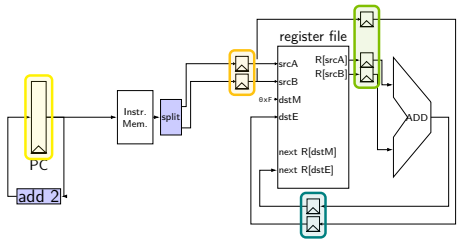
pipelined addq paths



path 1: 80 picoseconds
path 2: **210 picoseconds**
path 3: **210 picoseconds**
path 4: 135 picoseconds
path 5: 110 picoseconds
path 6: 135 picoseconds
...
overall cycle time: **210 picoseconds**

exercise

path	time
add 2	50 ps
instruction memory	200 ps
register file read	125 ps
add	100 ps
register file write	125 ps



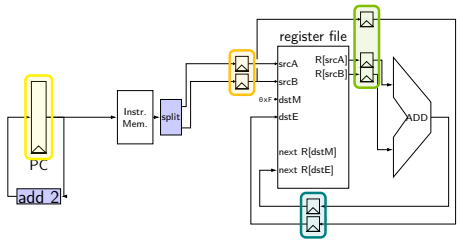
pipeline register delay: 10ps

how will throughput improve if we double the speed of the instruction memory?

A. 2.00x **B.** 1.70x to 1.99x
C. 1.60x to 1.69x **D.** 1.50x to 1.59x
E. less than 1.50x

exercise

path	time
add 2	50 ps
instruction memory	200 ps
register file read	125 ps
add	100 ps
register file write	125 ps



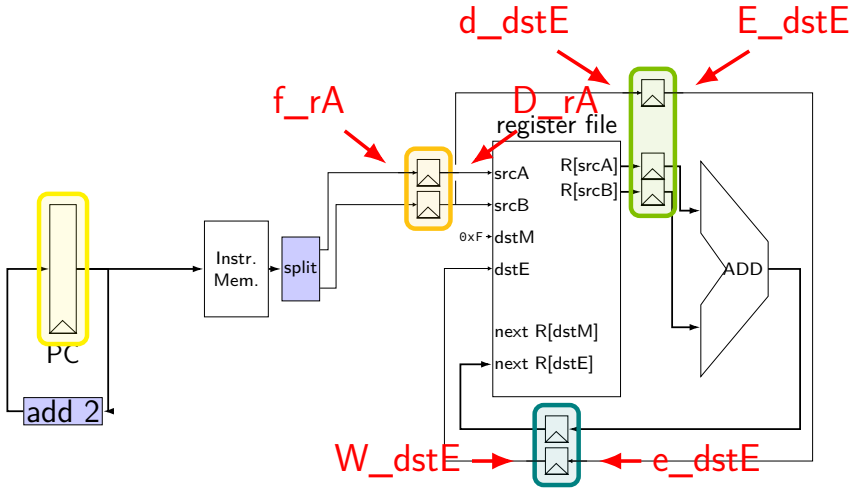
pipeline register delay: 10ps

how will throughput improve if we double the speed of the instruction memory?

A. 2.00x **B.** 1.70x to 1.99x
C. 1.60x to 1.69x **D.** 1.50x to 1.59x
E. less than 1.50x

$$\frac{1}{135} \div \frac{1}{210} = 1.56\text{x} \text{ — D}$$

pipeline register naming convention



pipeline register naming convention

f — fetch sends values here

D — decode receives values here

d — decode sends values here

...

addq HCL

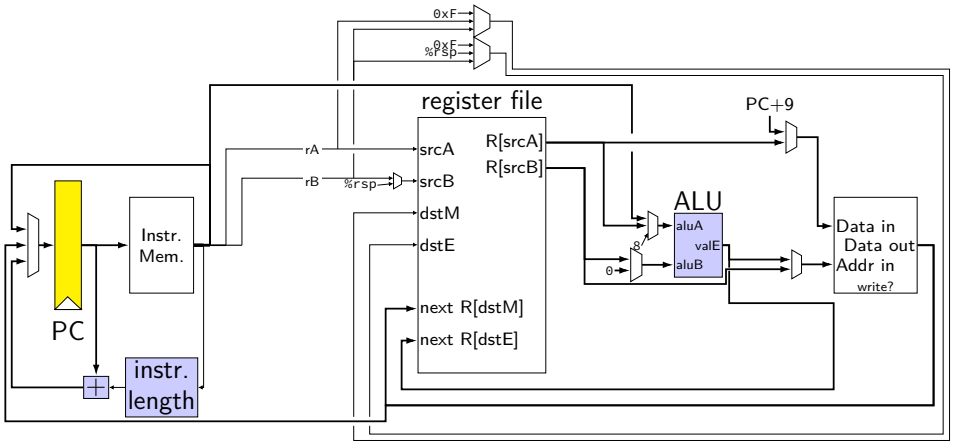
```
...
/* f: from fetch */
f_rA = i10bytes[12..16];
f_rB = i10bytes[12..16];

/* fetch to decode */
/* f_rA -> D_rA, etc. */
register fD {
    rA : 4 = REG_NONE;
    rB : 4 = REG_NONE;
}
```

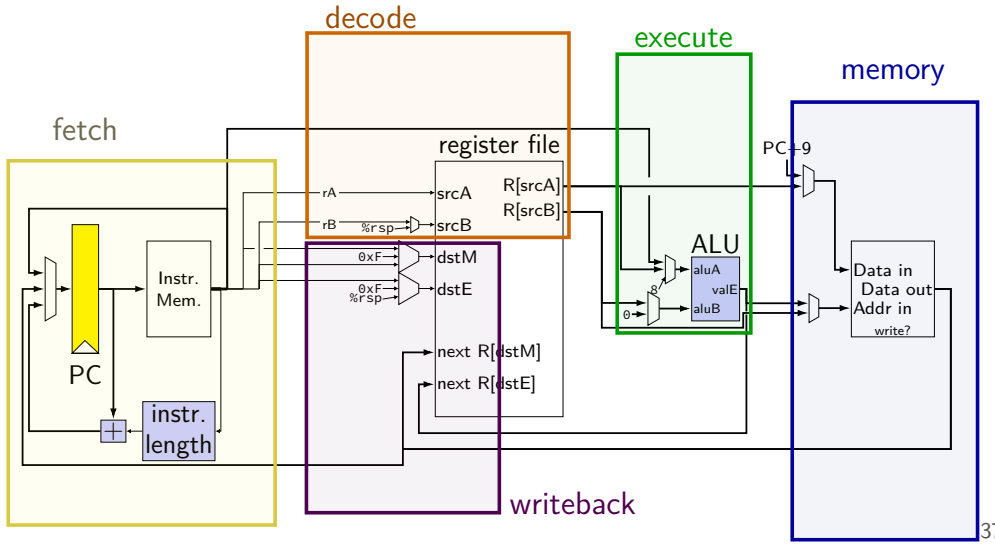
```
/* D: to decode
   d: from decode */
d_dstE = D_rB;
/* use register file: */
reg_srcA = D_rA;
d_valA = reg_outputA;
...

/* decode to execute */
register dE {
    dstE : 4 = REG_NONE;
    valA : 64 = 0;
    valB : 64 = 0;
}
```

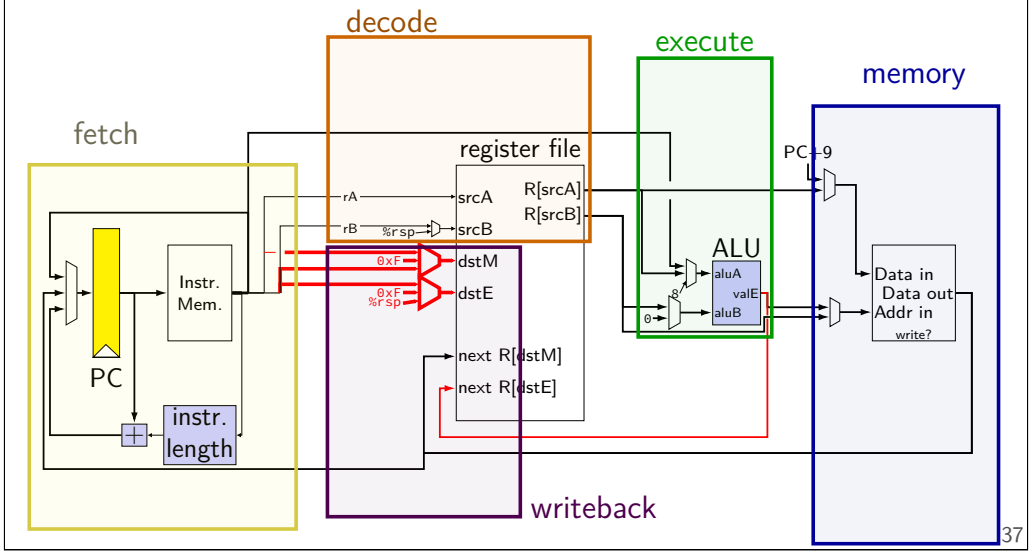

SEQ without stages



SEQ with stages

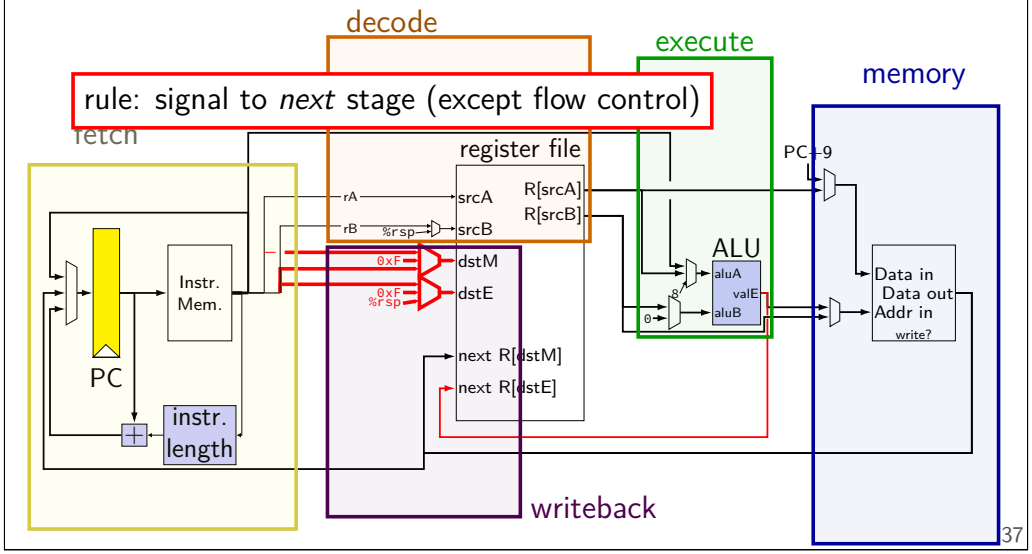


SEQ with stages

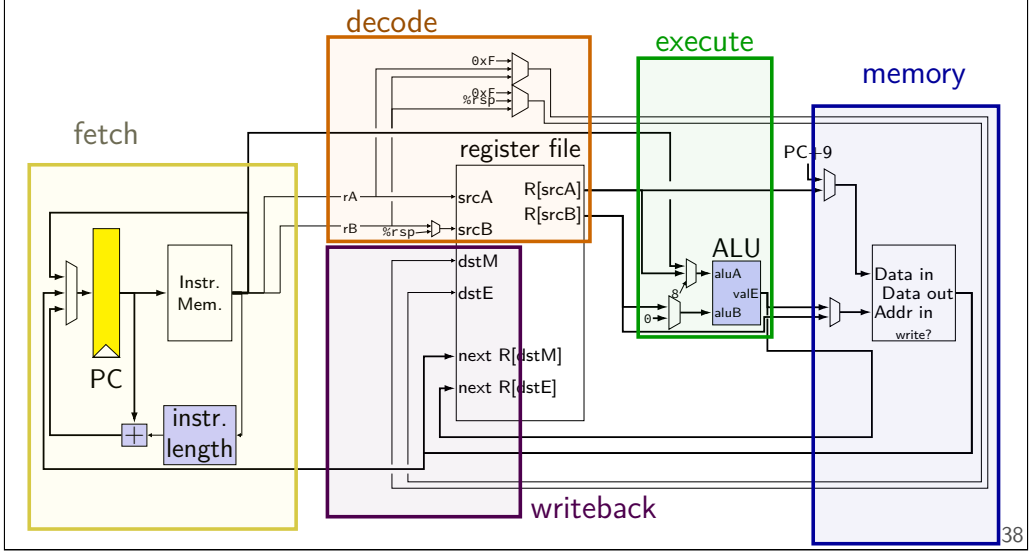


SEQ with stages

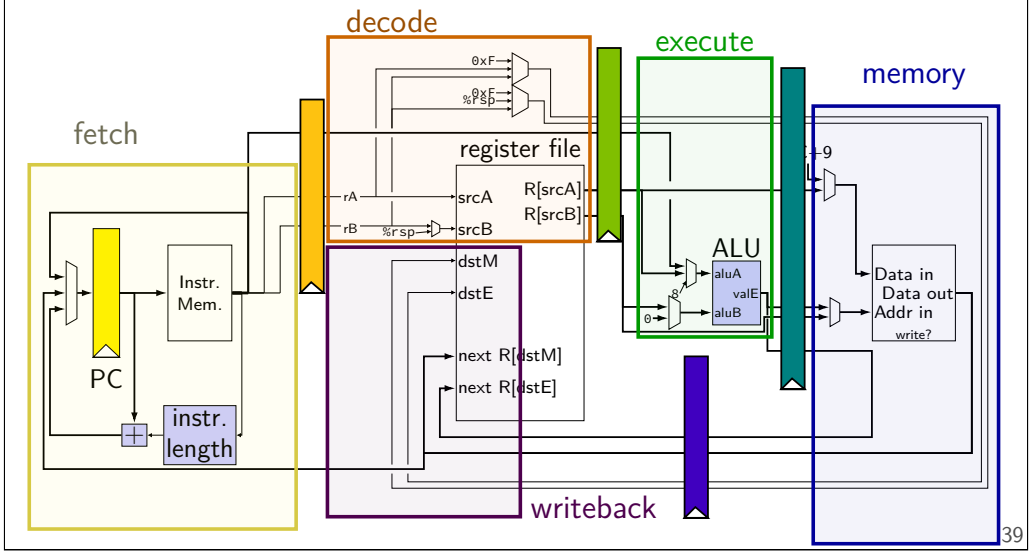
rule: signal to *next* stage (except flow control)



SEQ with stages (actually sequential)



adding pipeline registers



adding pipeline registers