last time

HCL details/built in components

HCL debug/interactive options

walkthrough of SEQ stages/needed MUXes

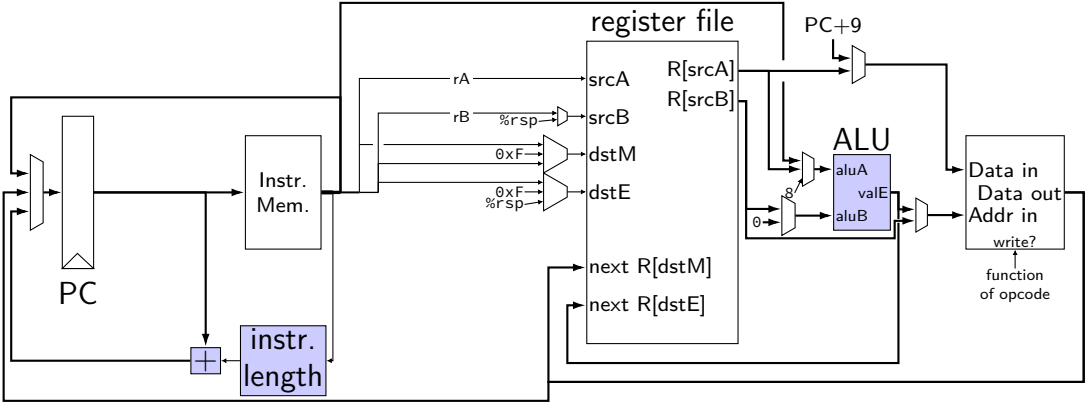
critical path

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

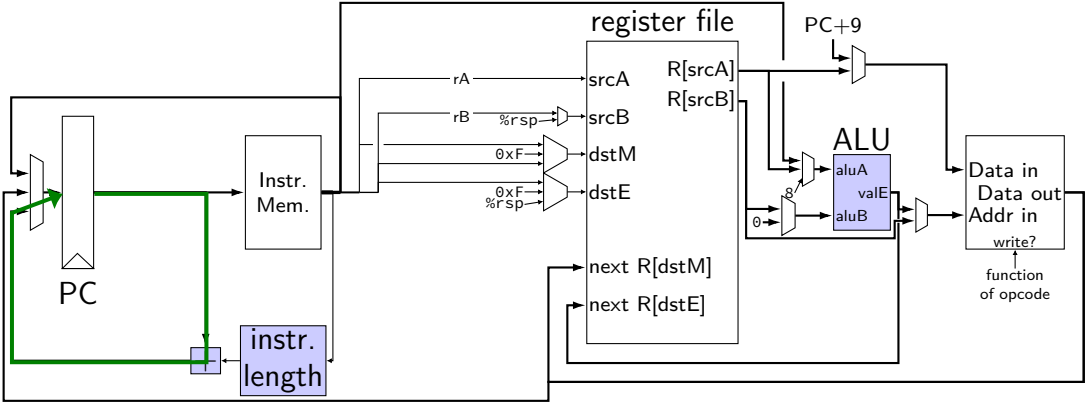
critical path: slowest of all these paths — determines cycle time

SEQ paths


register file
PC+9
rA
rB
%rsp
0xF
0xF
%rsp
srcA
srcB
dstM
dstE
R[srcA]
R[srcB]
next R[dstM]
next R[dstE]
ALU
aluA
valE
aluB
8
0
Instr.
Mem.
PC
instr.
length
Data in
Data out
Addr in
write?
function
of opcode

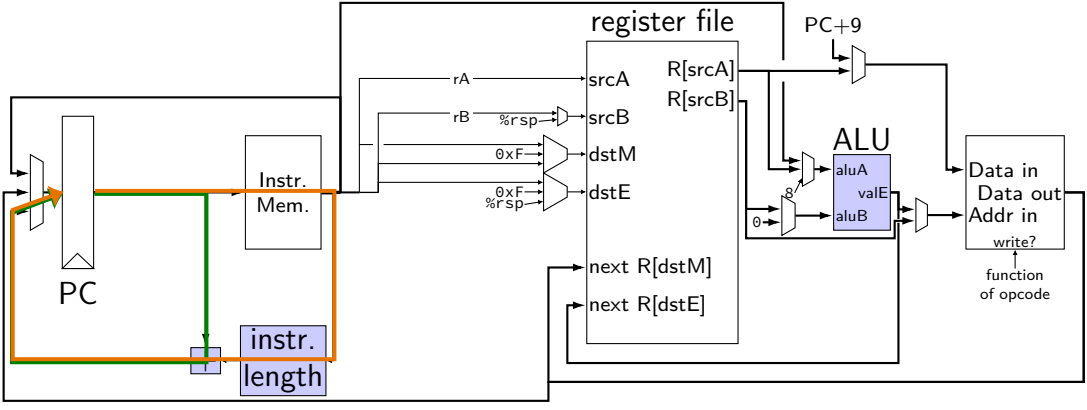
SEQ paths

path 1: 25 picoseconds



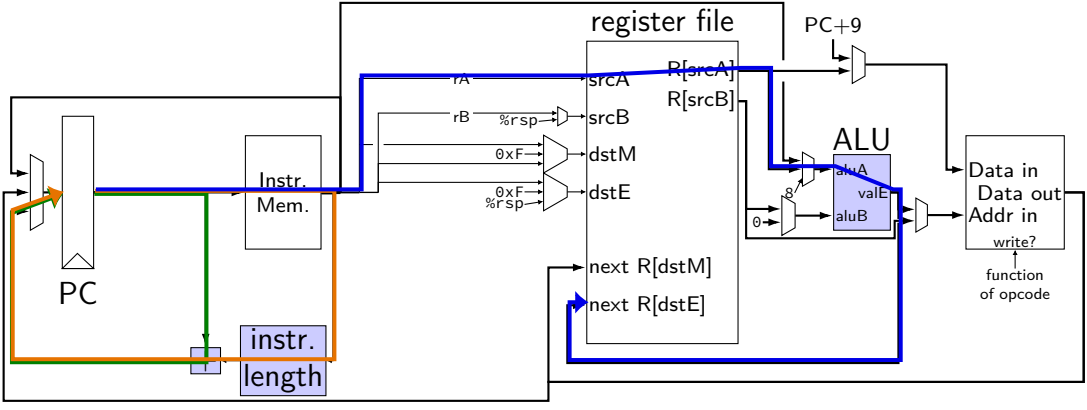
SEQ paths

path 1: 25 picoseconds path 2: 50 picoseconds



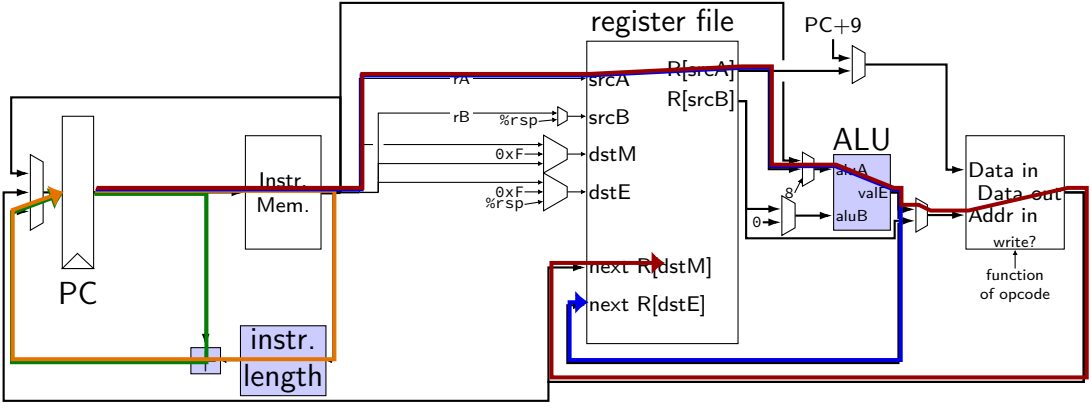
SEQ paths

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



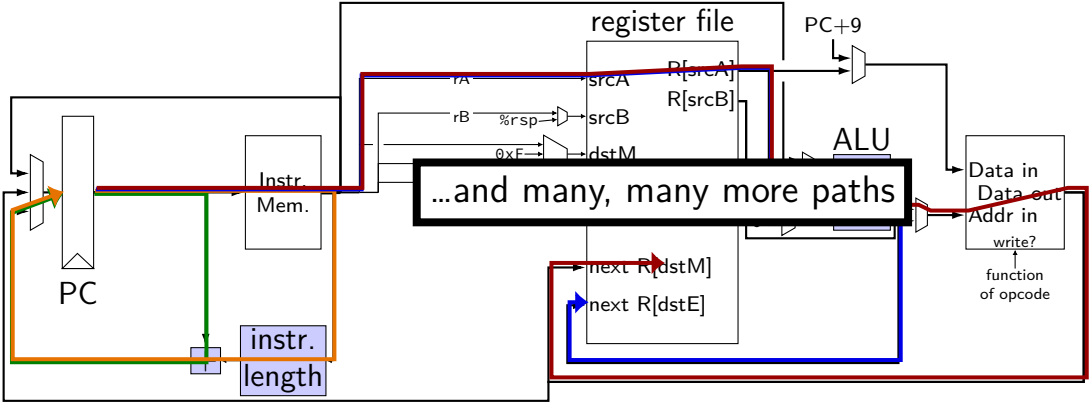
SEQ paths

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

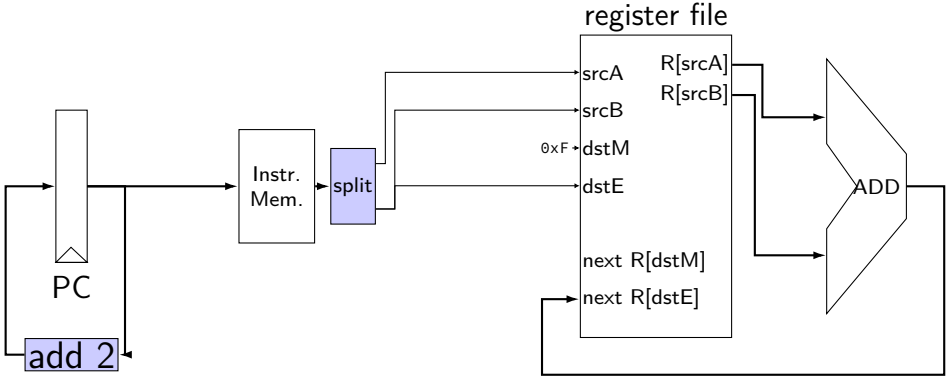


SEQ paths

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

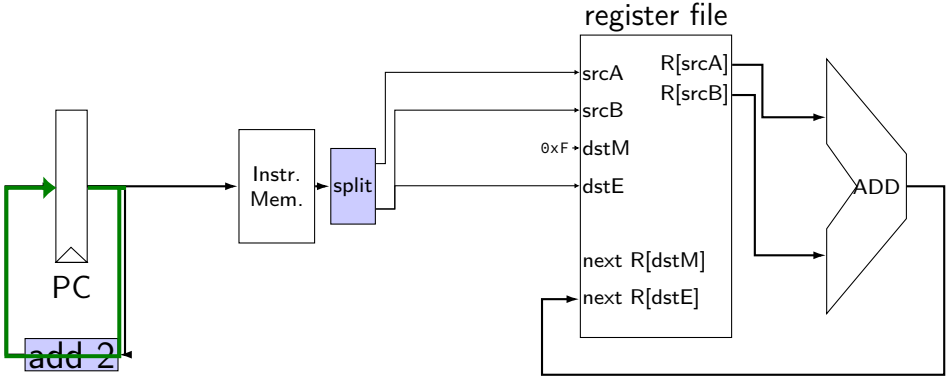


sequential addq paths



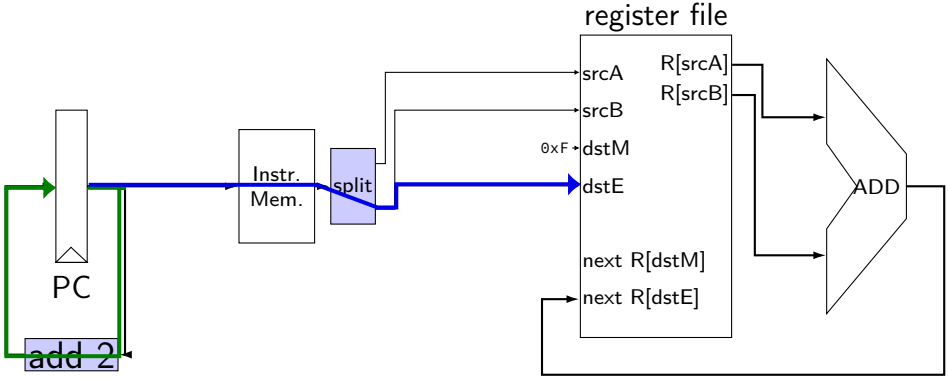
sequential addq paths

path 1: 25 picoseconds



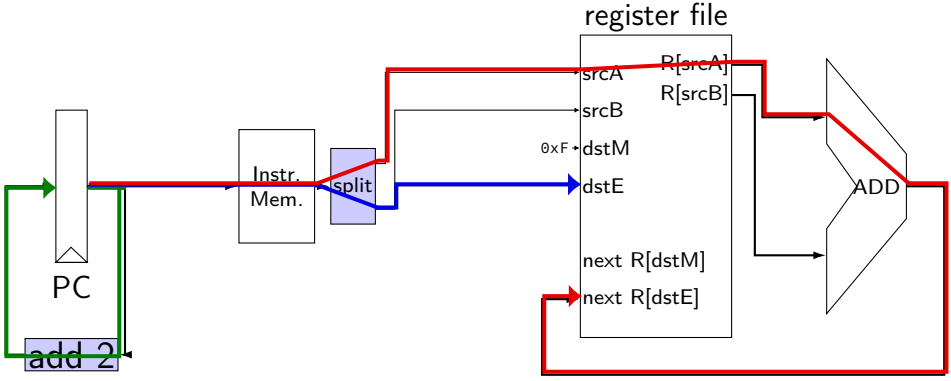
sequential addq paths

path 1: 25 picoseconds
path 2: 375 picoseconds



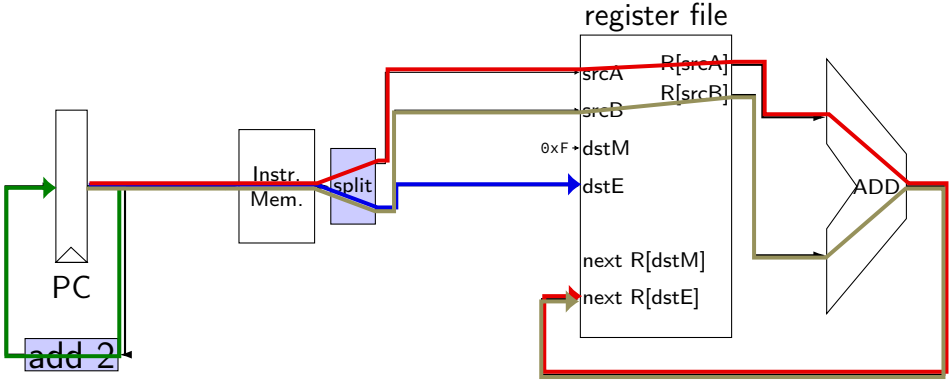
sequential addq paths

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



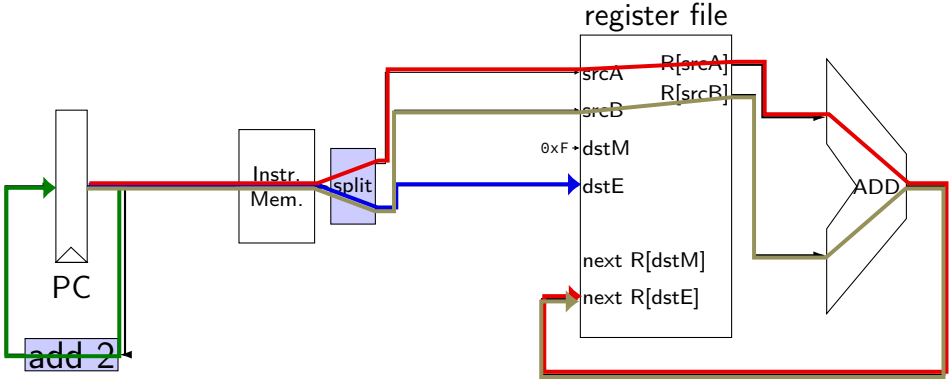
sequential addq paths

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

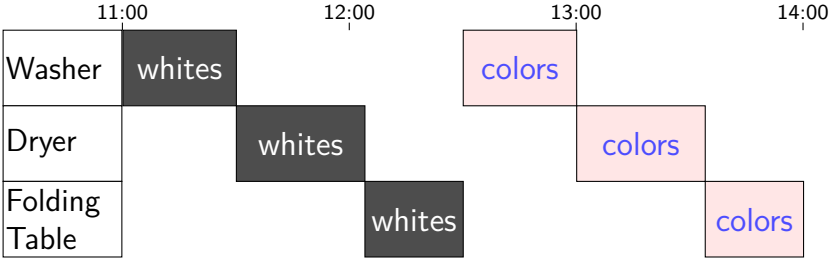


sequential addq paths

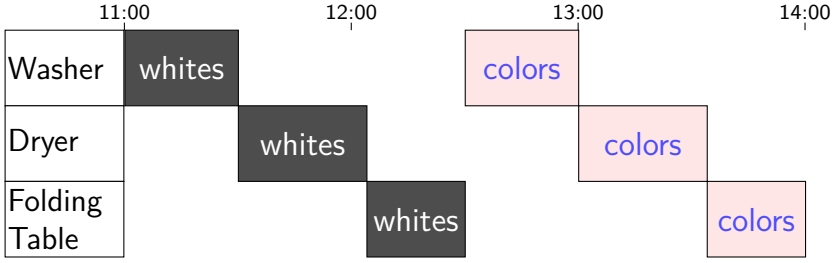
path 1: 25 picoseconds
path 2: 375 picoseconds
path 3: **500 picoseconds**
path 4: **500 picoseconds**
overall cycle time: **500 picoseconds** (longest path)

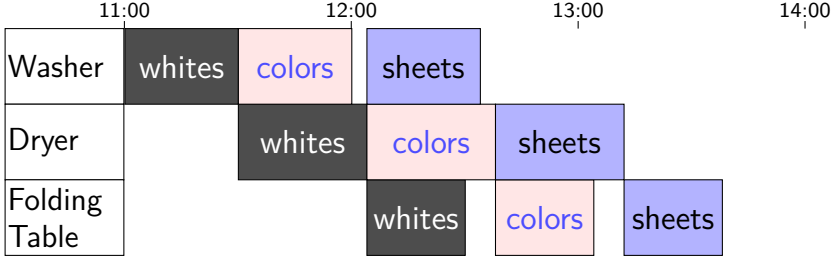


Human pipeline: laundry

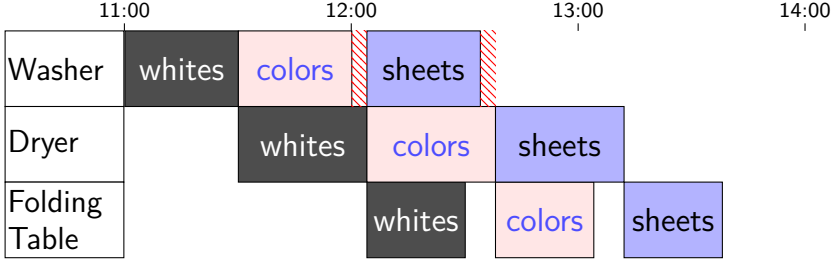


Human pipeline: laundry

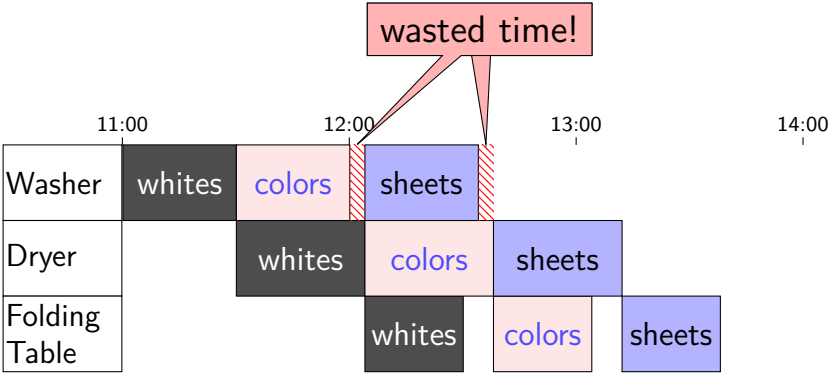




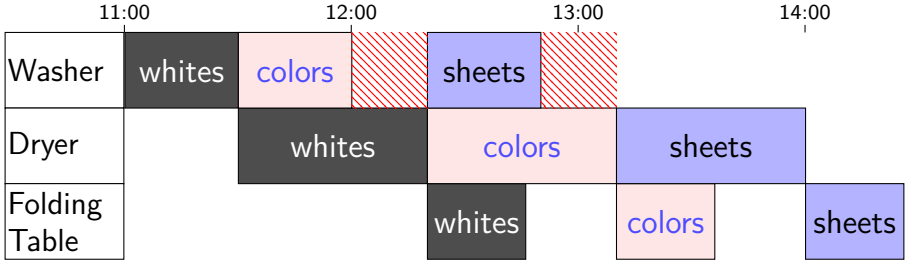
Waste (1)


11:00
12:00
13:00
14:00
Washer
Dryer
Folding Table
whites
colors
sheets
whites
colors
sheets
whites
colors
sheets

Waste (1)

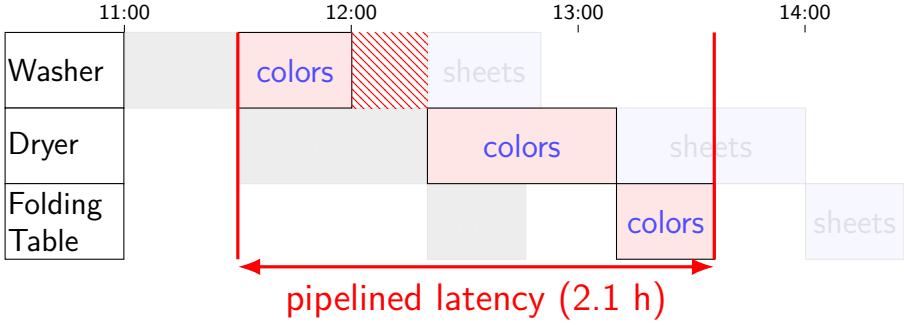

wasted time!
11:00
12:00
13:00
14:00
Washer
whites
colors
sheets
Dryer
whites
colors
sheets
Folding Table
whites
colors
sheets

Waste (2)

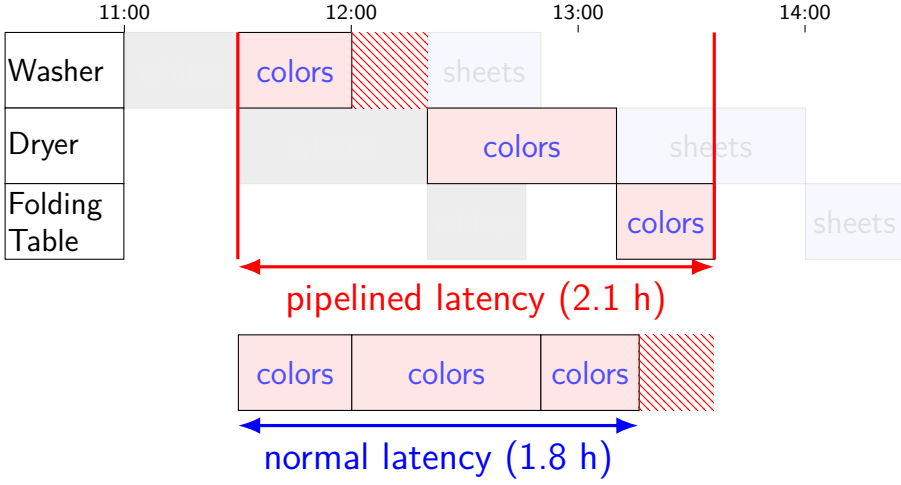

11:00
12:00
13:00
14:00
Washer
Dryer
Folding Table
whites
colors
sheets
whites
colors
sheets
whites
colors
sheets

Latency — Time for One

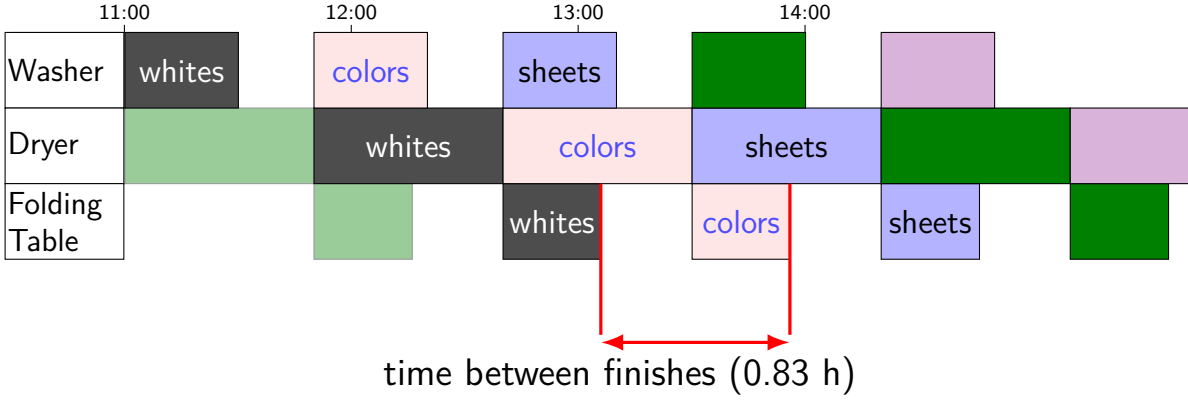
Latency — Time for One



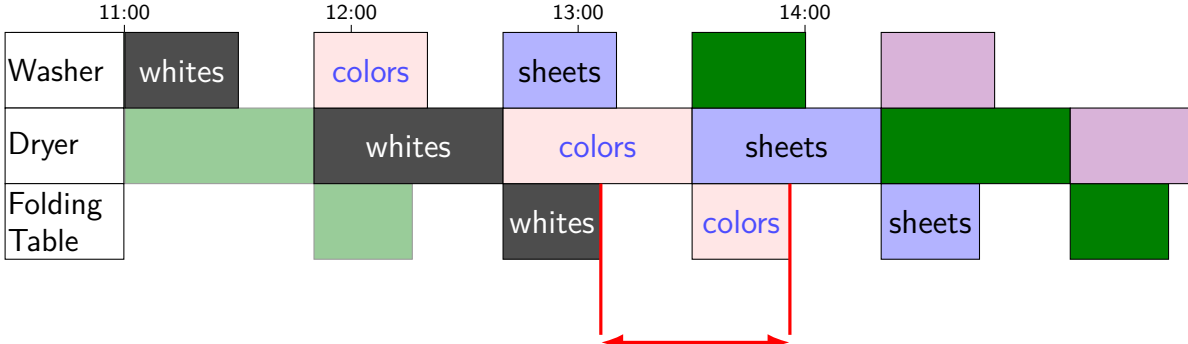
Latency — Time for One



Throughput — Rate of Many

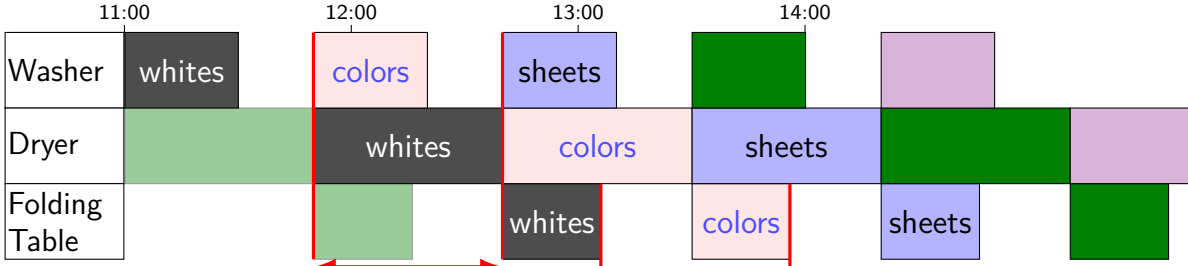


Throughput — Rate of Many



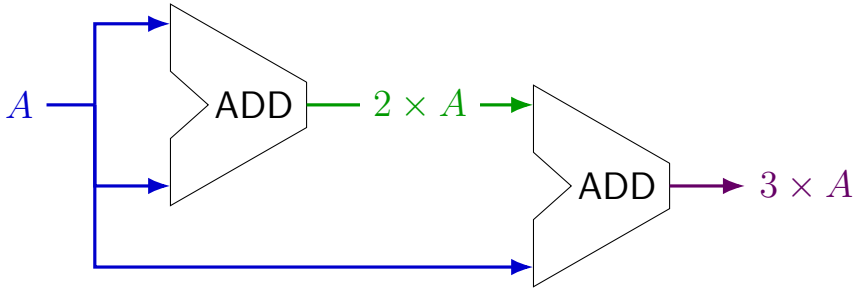
$$\frac{1 \text{ load}}{0.83\text{h}} = 1.2 \text{ loads/h}$$

Throughput — Rate of Many

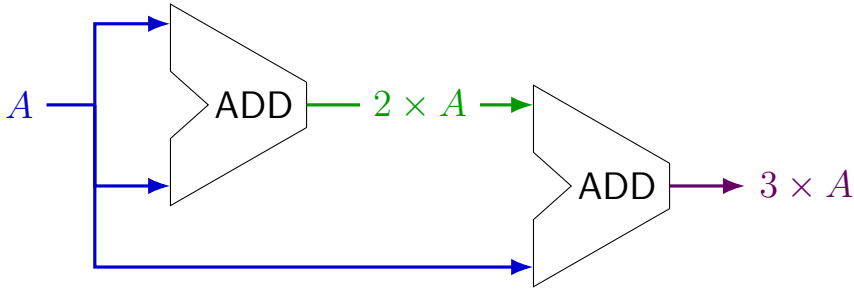


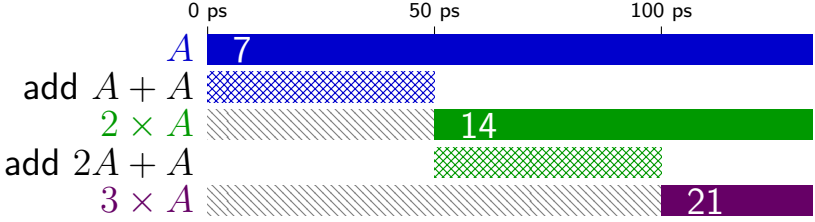
$$\frac{1 \text{ load}}{0.83\text{h}} = 1.2 \text{ loads/h}$$

times three circuit

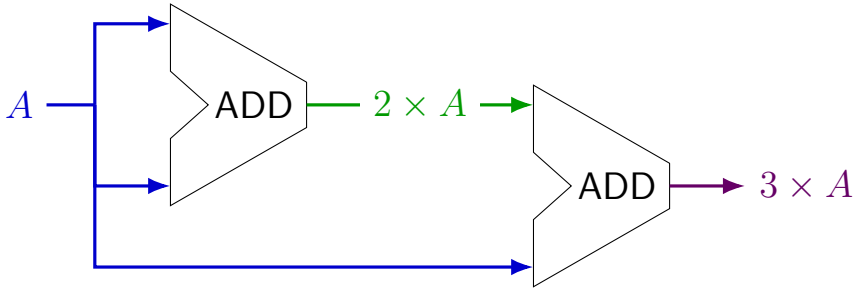


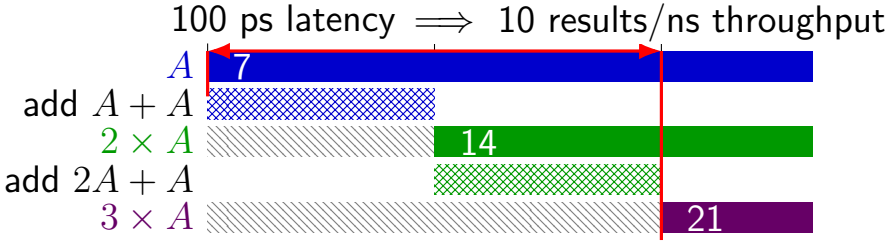
times three circuit



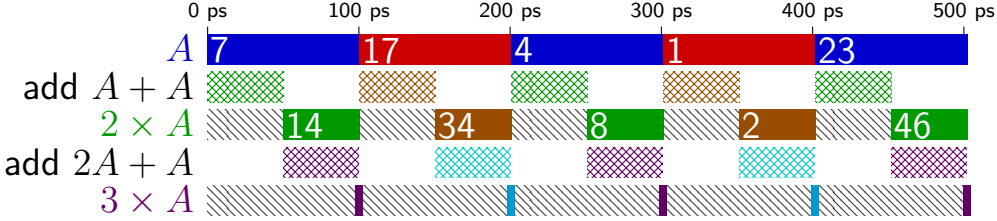


times three circuit

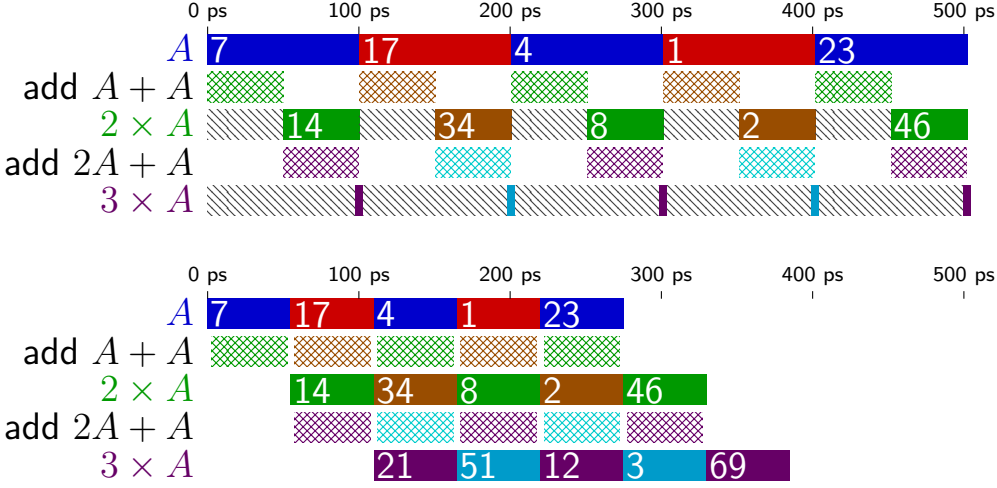




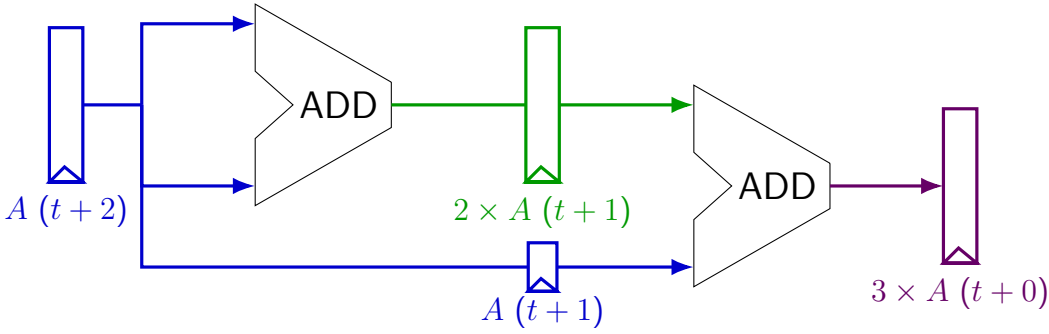
times three and repeat



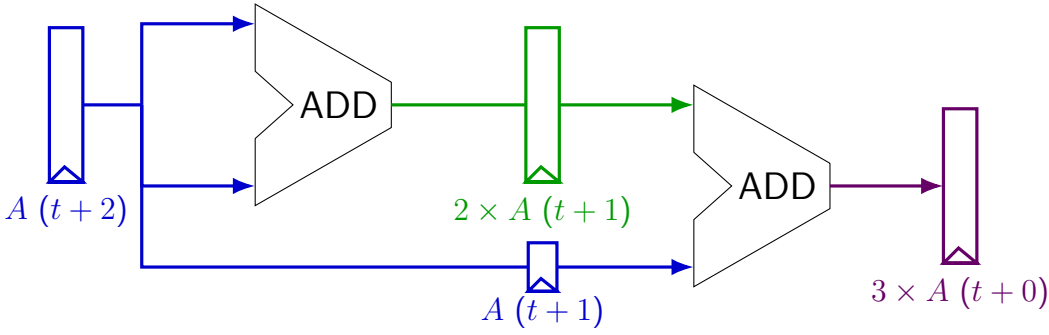
times three and repeat

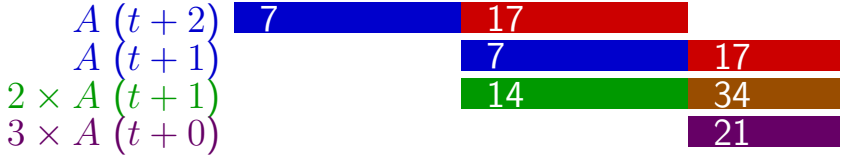


pipelined times three

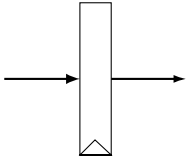


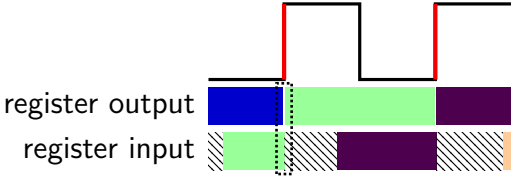
pipelined times three



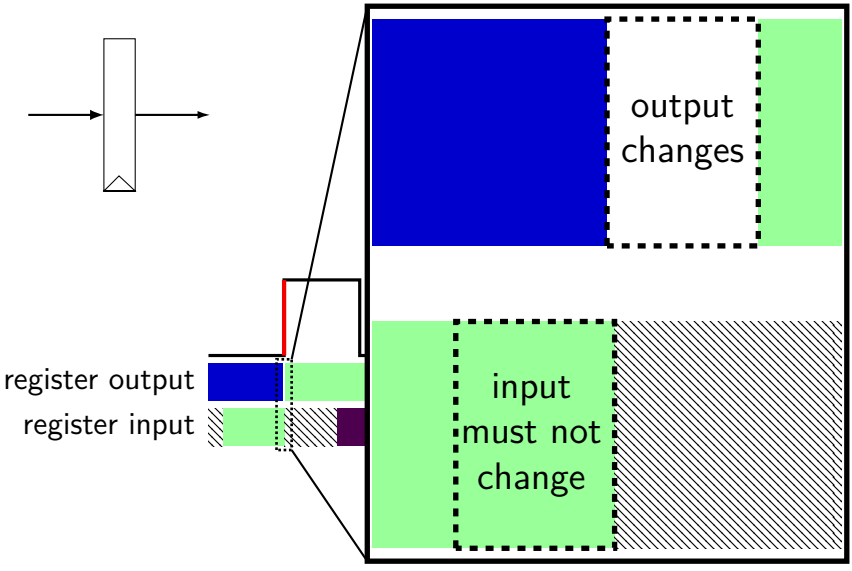


register tolerances

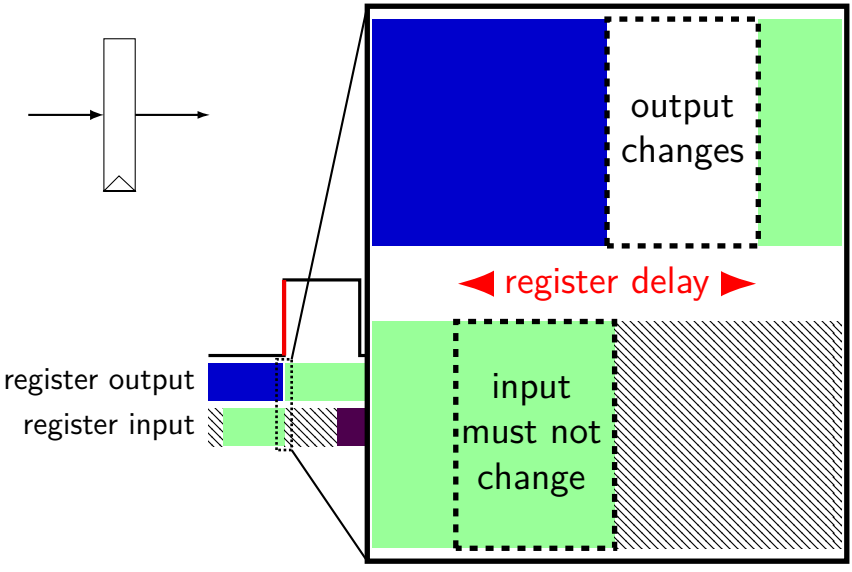




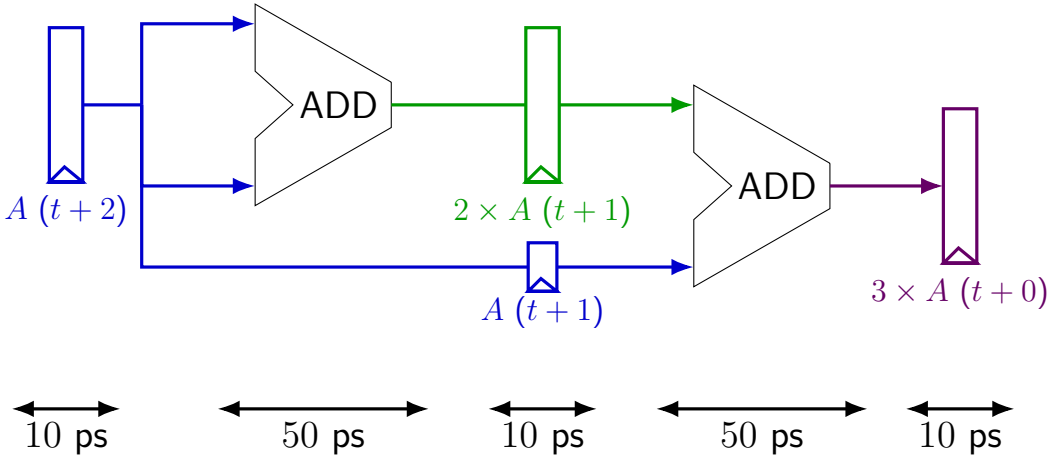
register tolerances



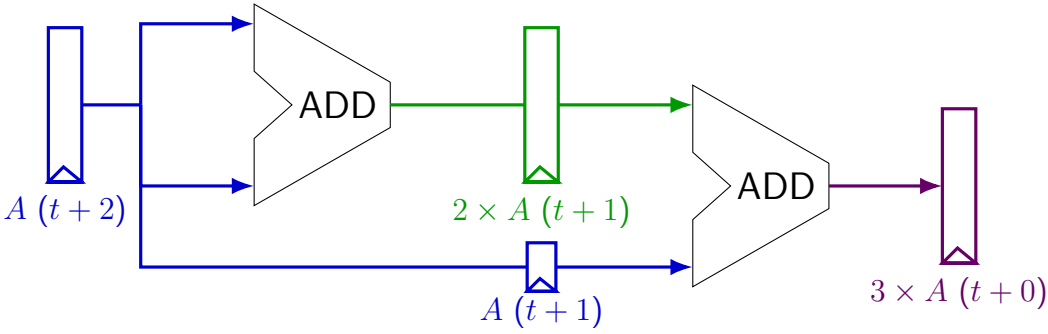
register tolerances

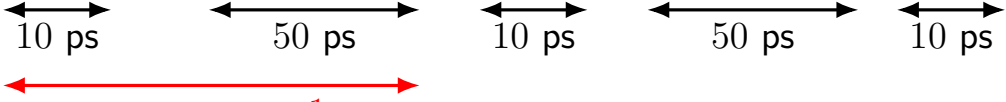


times three pipeline timing



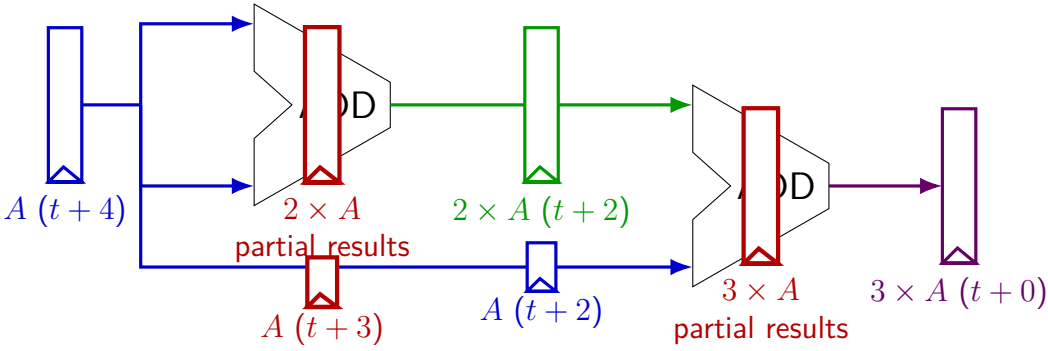
times three pipeline timing



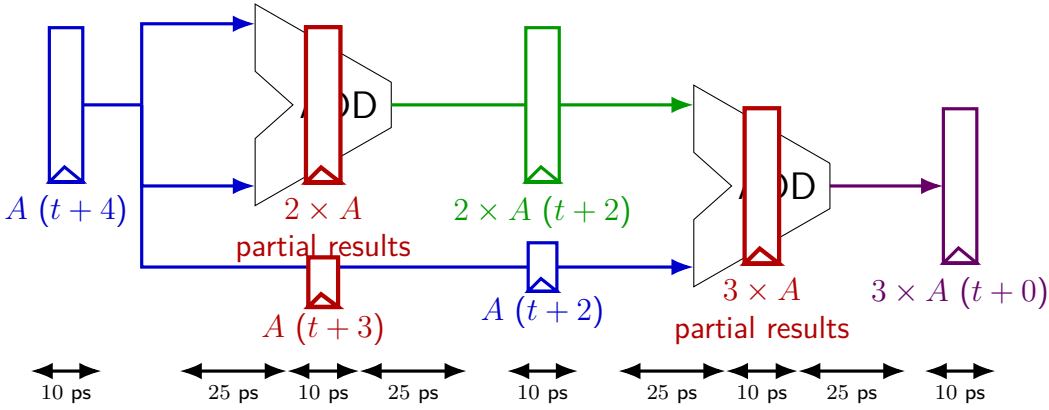


throughput: $\dfrac{1}{60\ \text{ps}} \approx 16$ G operations/sec

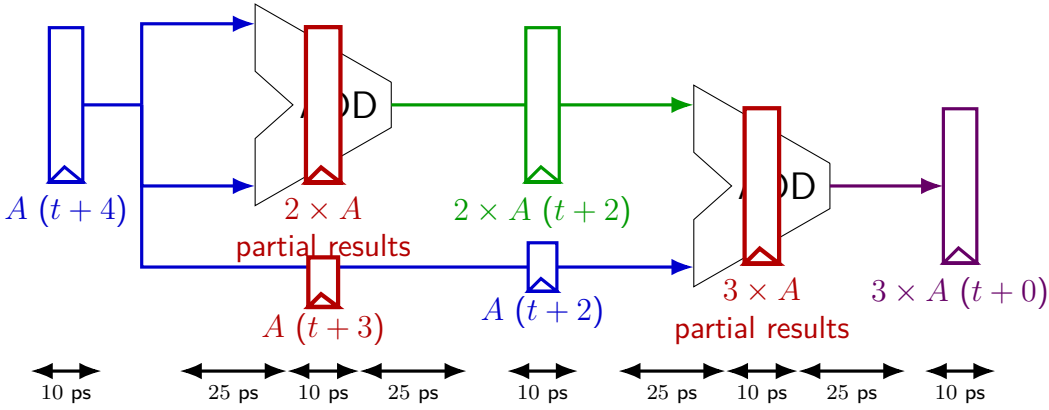
deeper pipeline



deeper pipeline

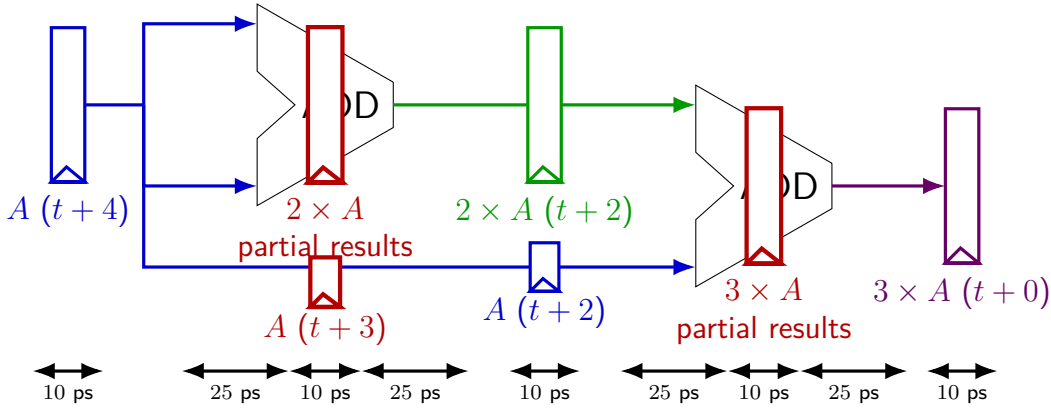


deeper pipeline



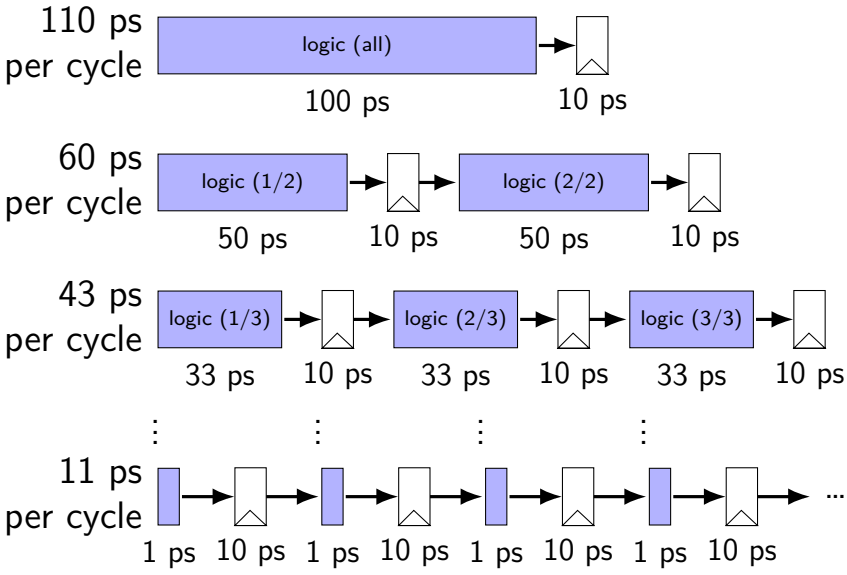
exercise: throughput now?

deeper pipeline

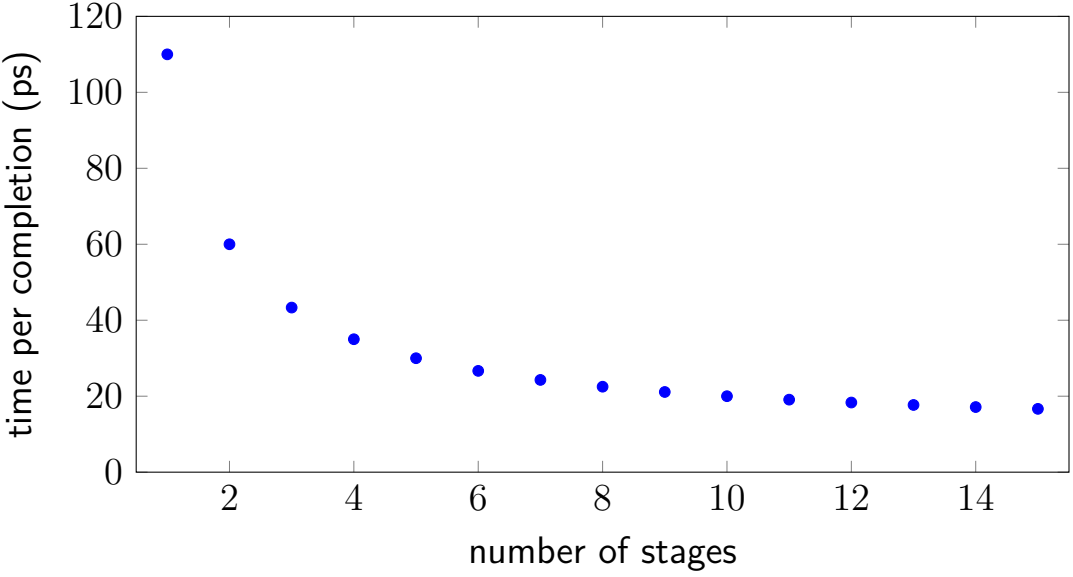


throughput: $\dfrac{1}{35\ \text{ps}} \approx 28$ G ops/sec

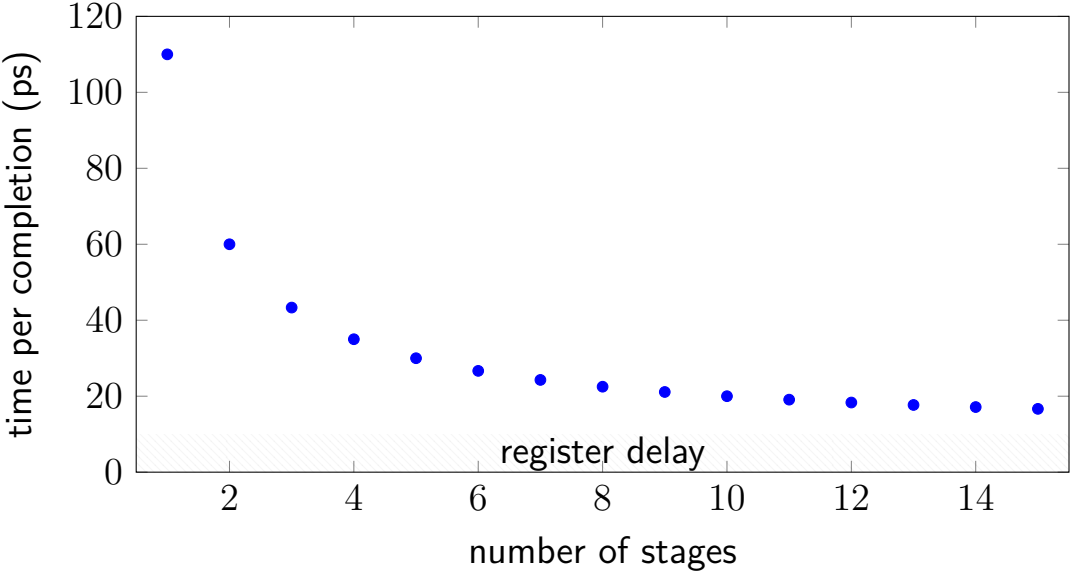
diminishing returns: register delays



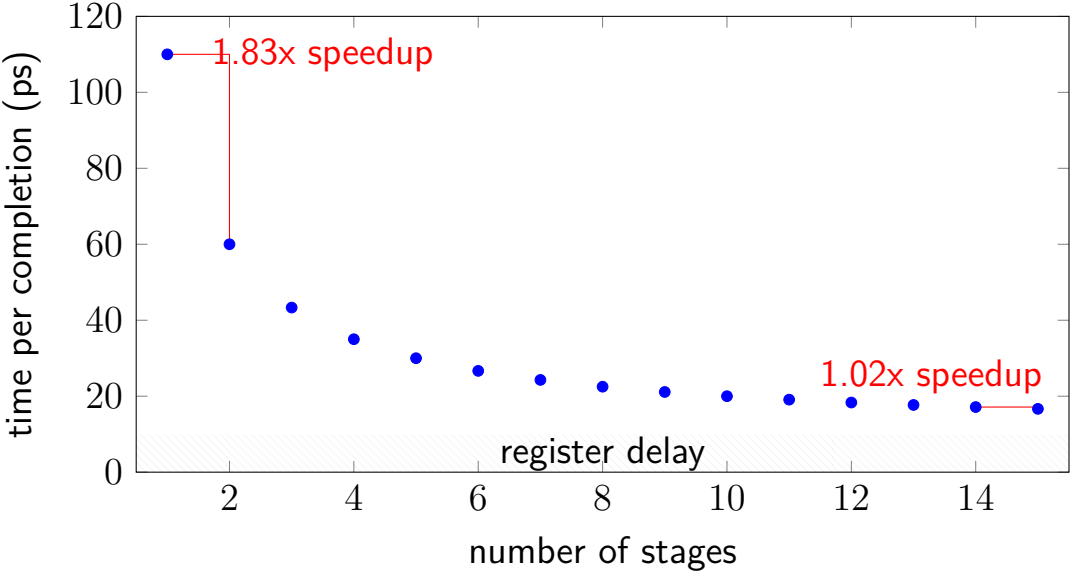
diminishing returns: register delays



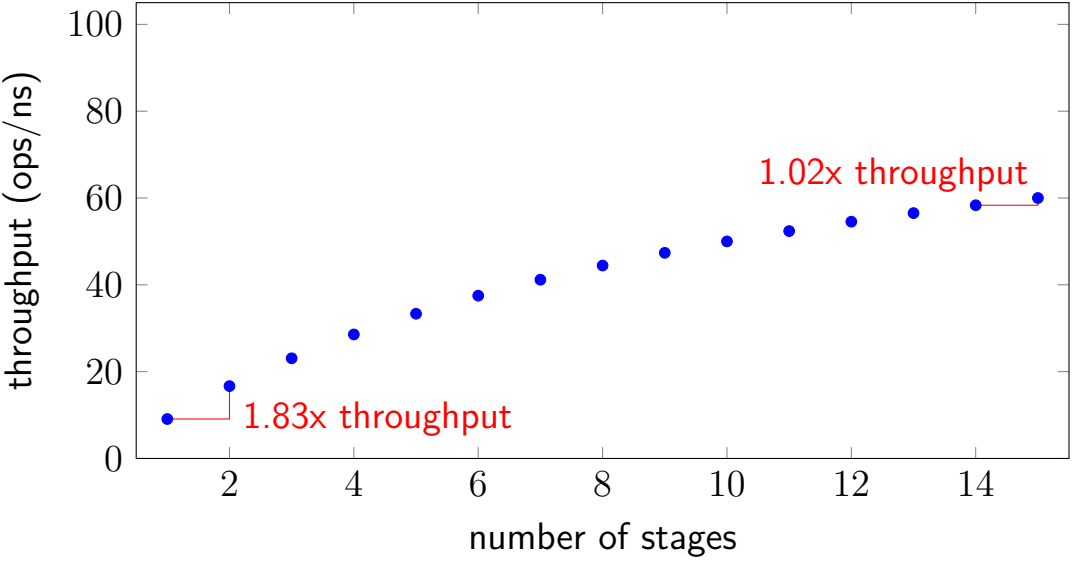
diminishing returns: register delays



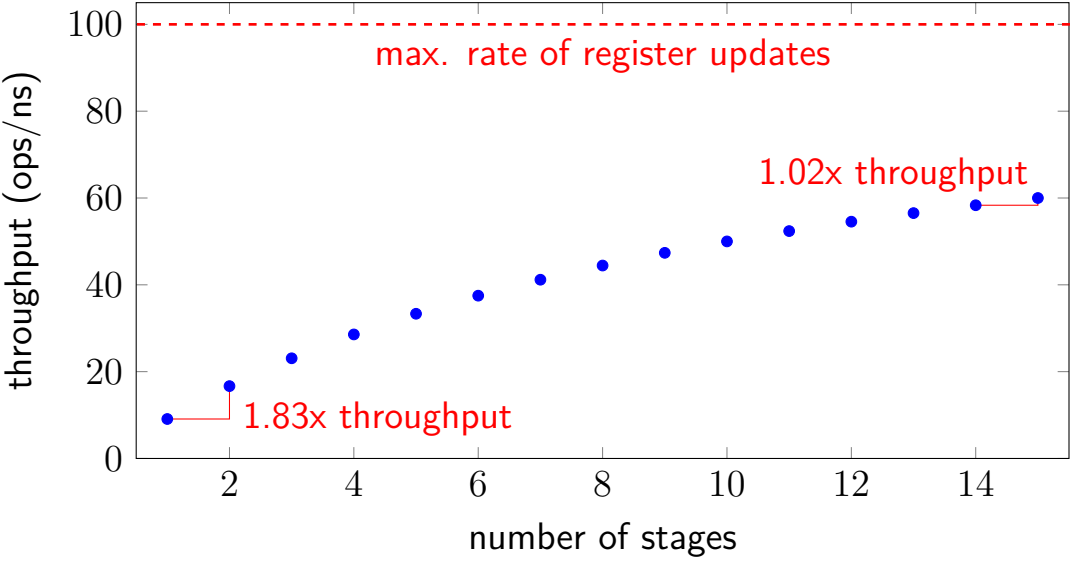
diminishing returns: register delays



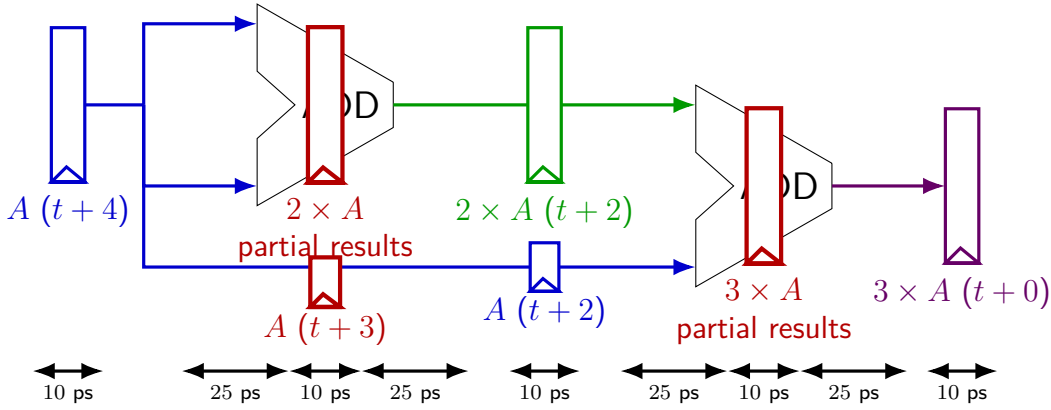
diminishing returns: register delays



diminishing returns: register delays



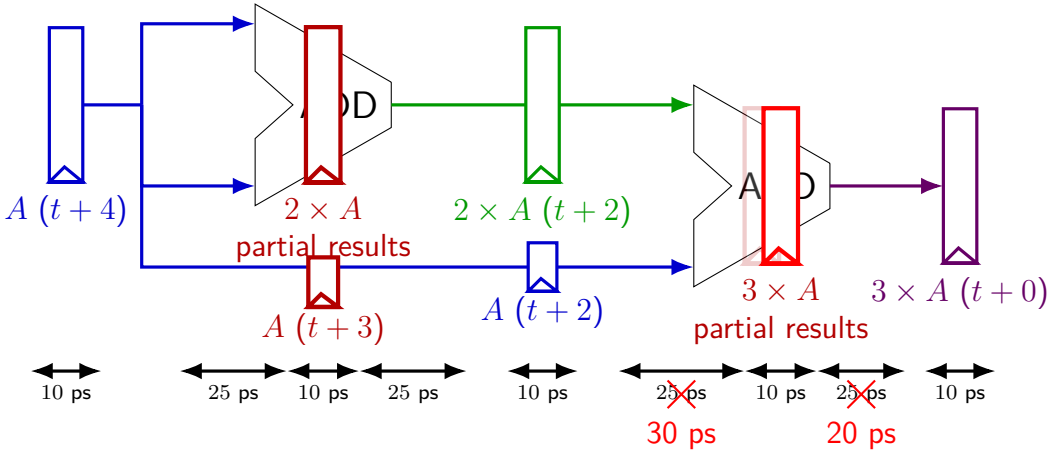
deeper pipeline



Problem: How much faster can we get?

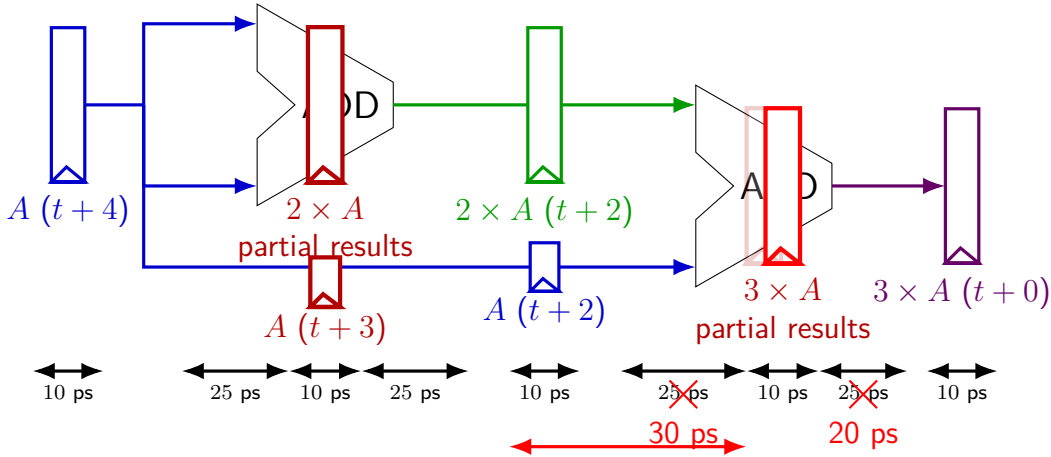
Problem: Can we even do this?

deeper pipeline



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

deeper pipeline

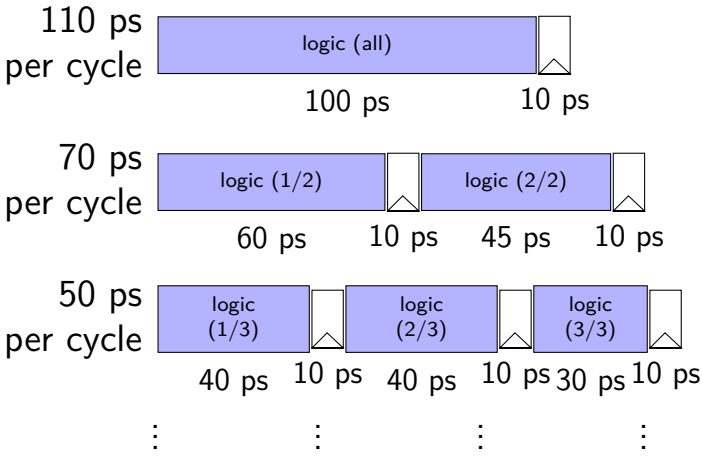


throughput: $\frac{1}{40 \text{ ps}} \approx 25$ G ops/sec

diminishing returns: uneven split

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

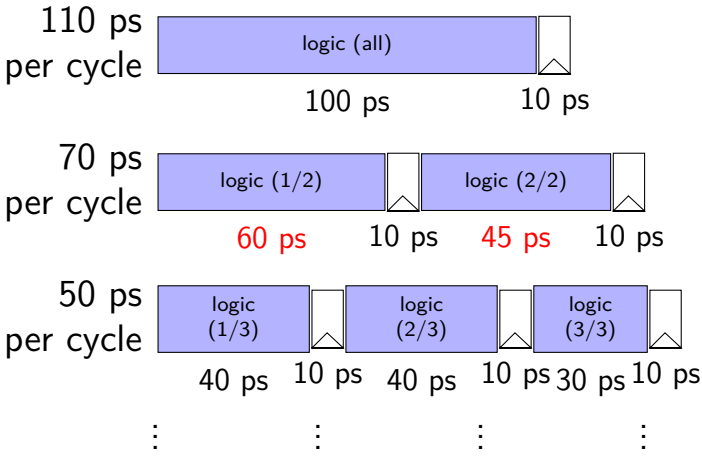
Probably not...



diminishing returns: uneven split

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

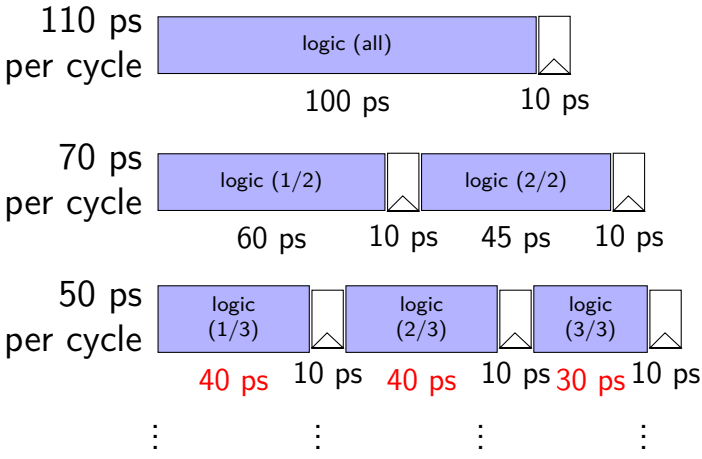
Probably not...



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

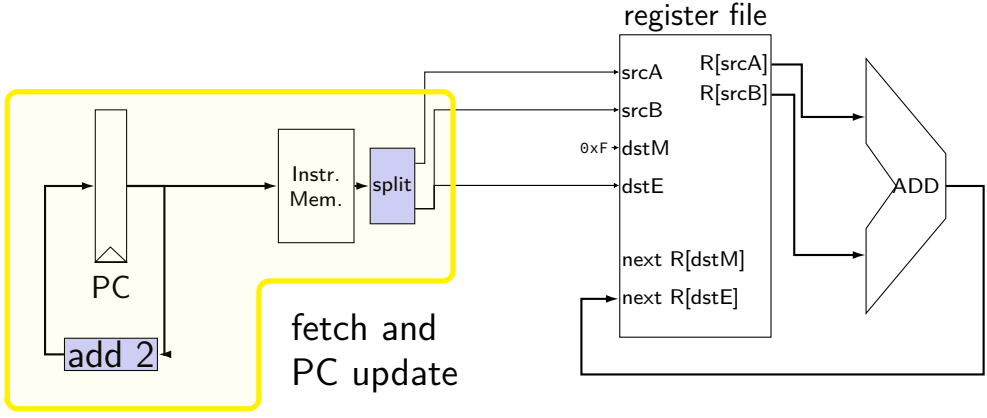
Execute: arithmetic (ALU)

Memory: read/write data memory

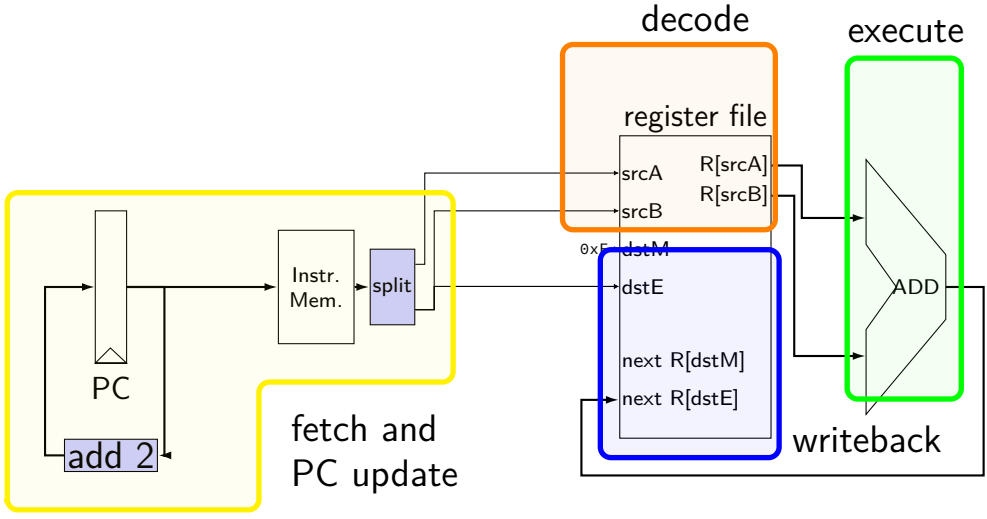
Writeback: write register file

5 stages
one instruction in each
compute next to start immediatelly

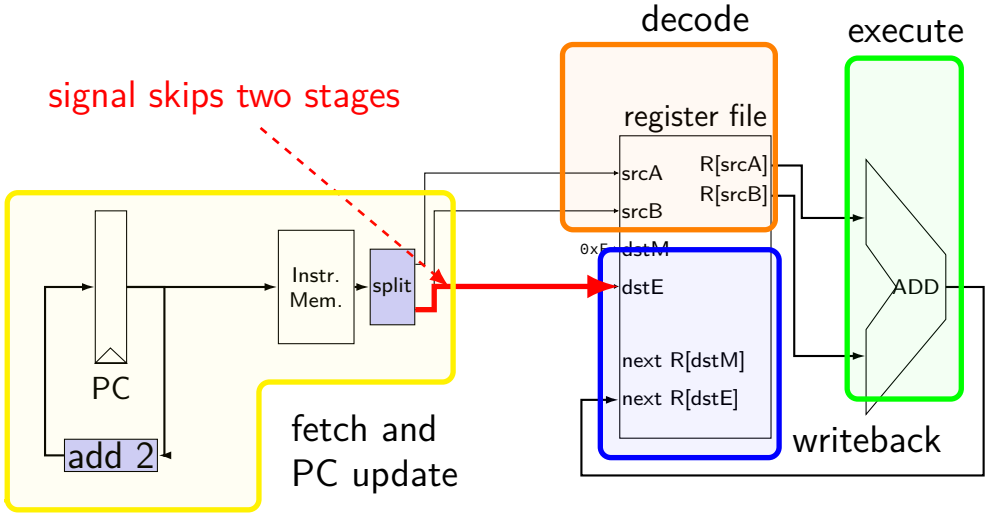
addq CPU


register file
srcA
srcB
0xF
dstM
dstE
R[srcA]
R[srcB]
next R[dstM]
next R[dstE]
ADD
Instr.
Mem.
split
PC
add 2
fetch and
PC update

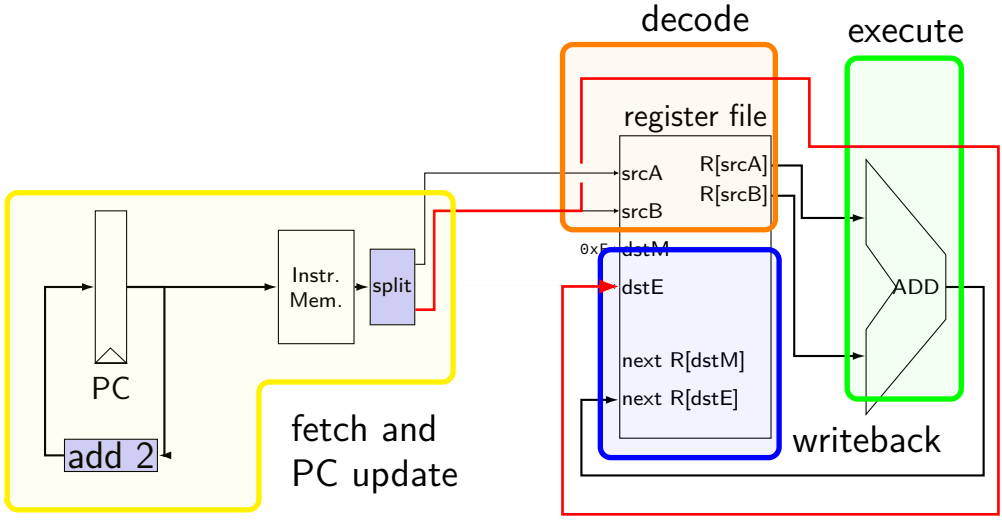
addq CPU


decode
execute
register file
srcA
srcB
R[srcA]
R[srcB]
0xF
dstM
dstE
next R[dstM]
next R[dstE]
ADD
Instr.
Mem.
split
PC
add 2
fetch and
PC update
writeback

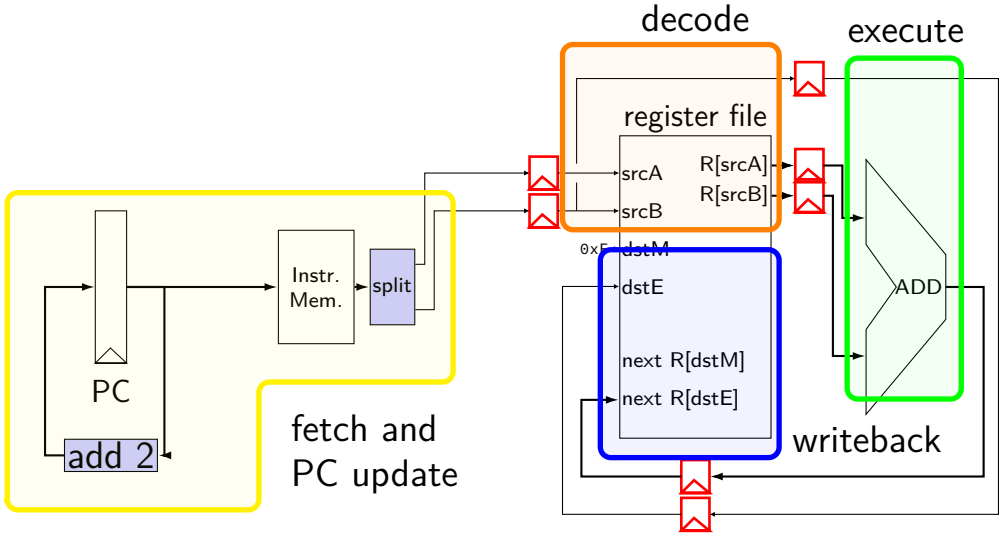
addq CPU


decode
execute
signal skips two stages
register file
srcA
srcB
R[srcA]
R[srcB]
0xF
dstM
dstE
next R[dstM]
next R[dstE]
ADD
Instr.
Mem.
split
PC
add 2
fetch and
PC update
writeback

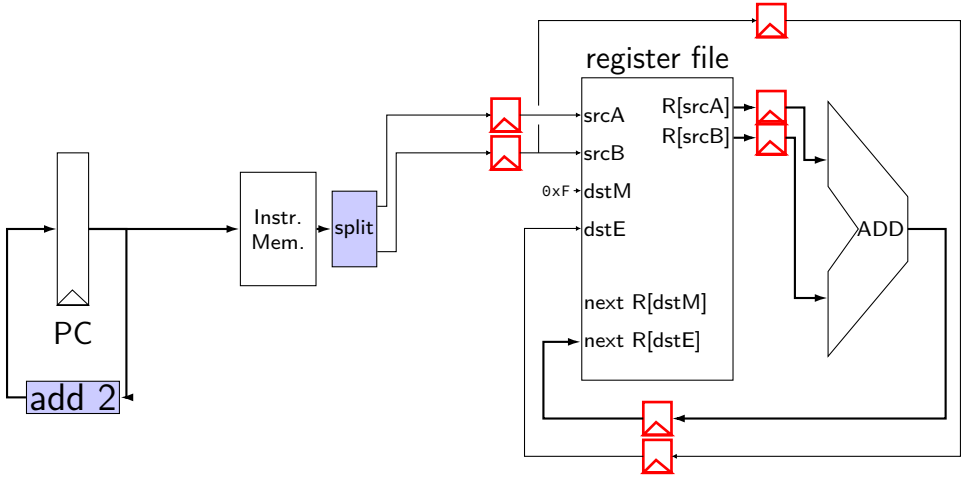
addq CPU


decode
execute
register file
srcA
srcB
R[srcA]
R[srcB]
0xF
dstM
dstE
next R[dstM]
next R[dstE]
ADD
writeback
Instr.
Mem.
split
PC
add 2
fetch and
PC update

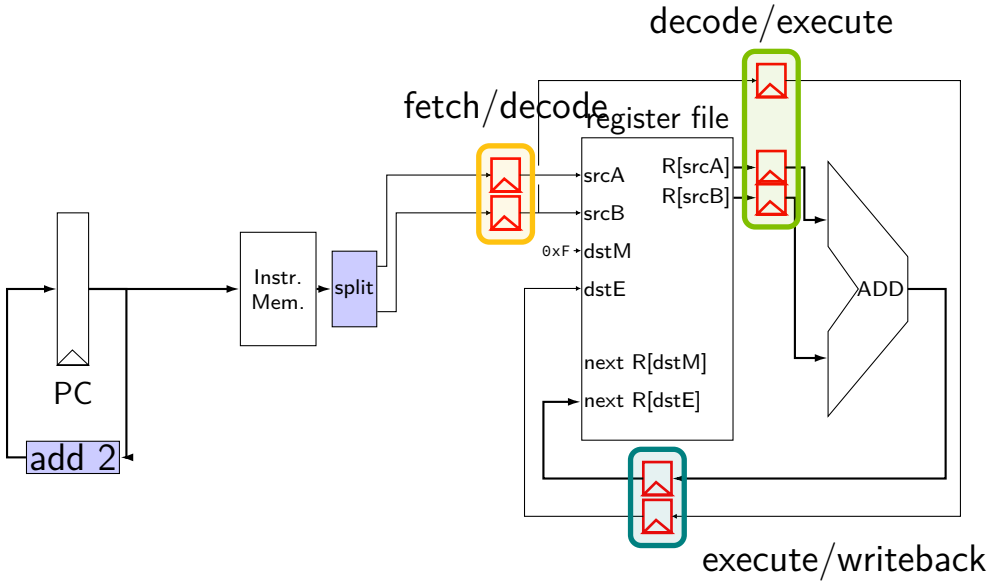
pipelined addq processor



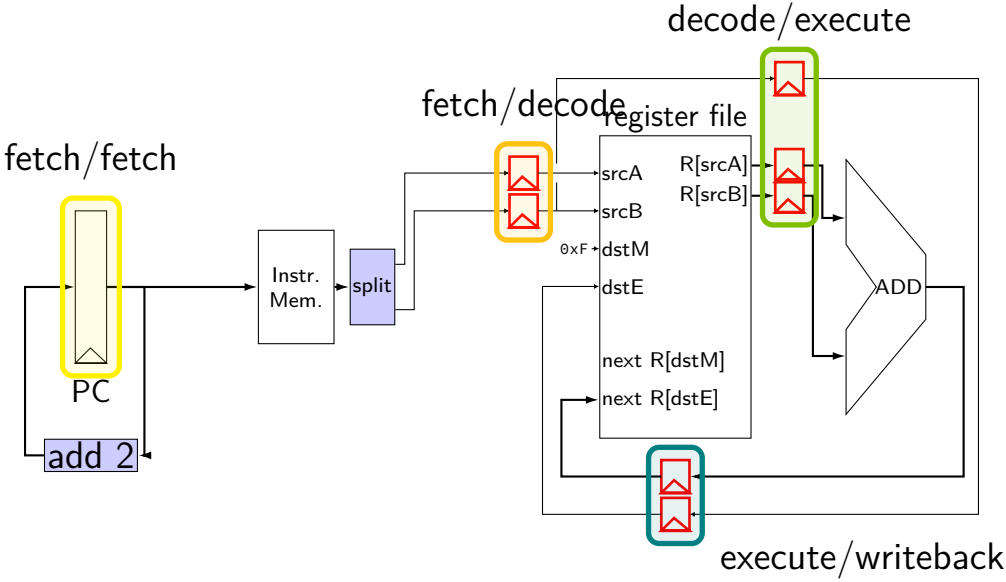
pipelined addq processor



pipelined addq processor

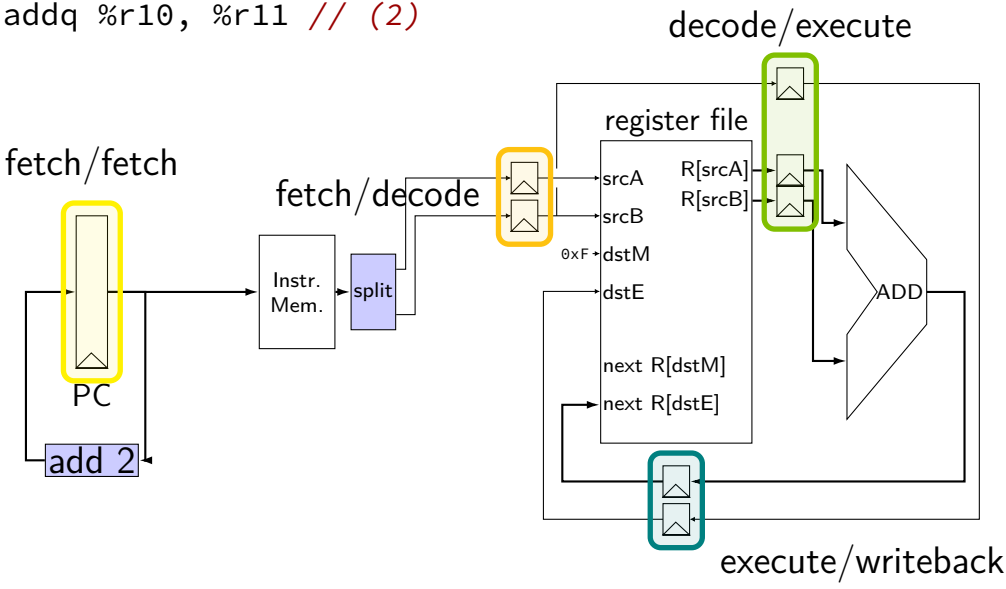


pipelined addq processor



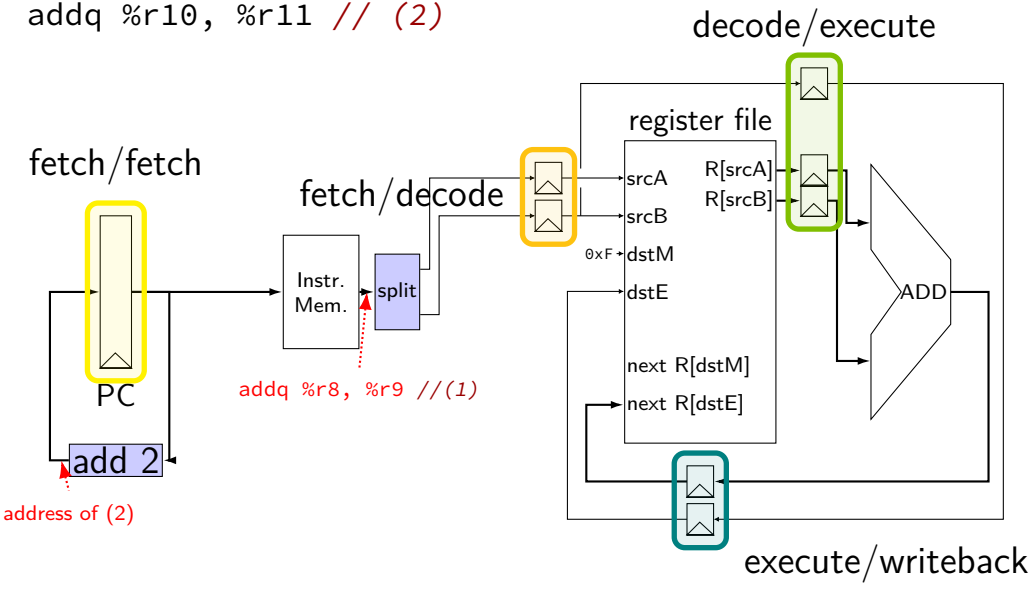
addq execution

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



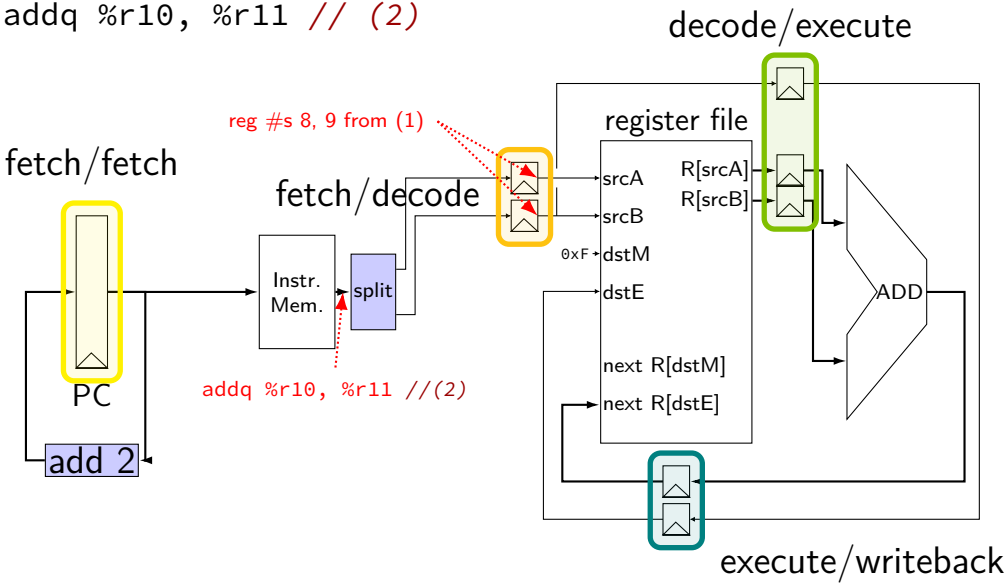
addq execution

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



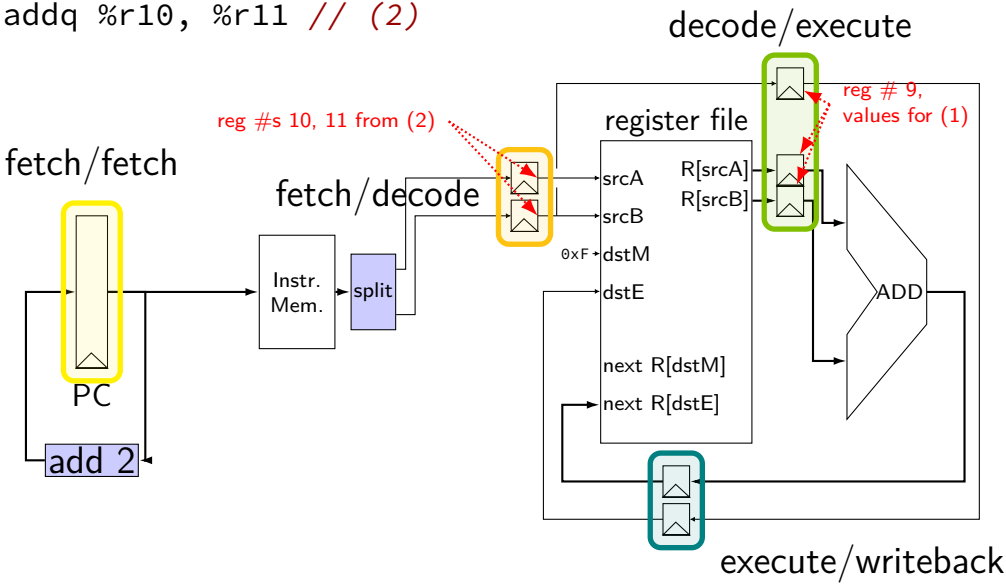
addq execution

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



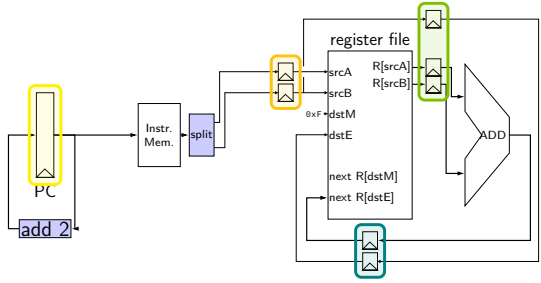
addq execution

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



addq processor timing

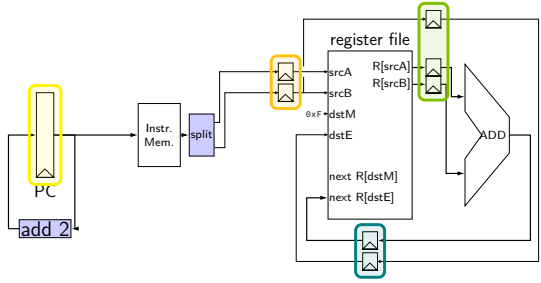
```
// initially %r8 = 800,
//           %r9 = 900, etc.
addq %r8, %r9
addq %r10, %r11
addq %r12, %r13
addq %r9, %r8
```



	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

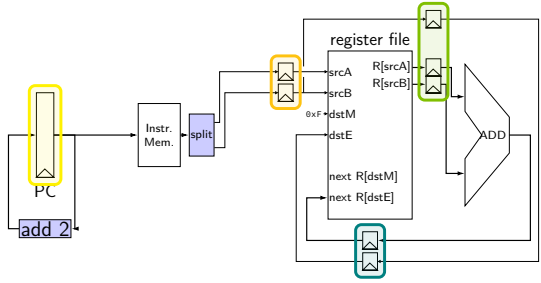
```
// initially %r8 = 800,
//           %r9 = 900, etc.
addq %r8, %r9
addq %r10, %r11
addq %r12, %r13
addq %r9, %r8
```



	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

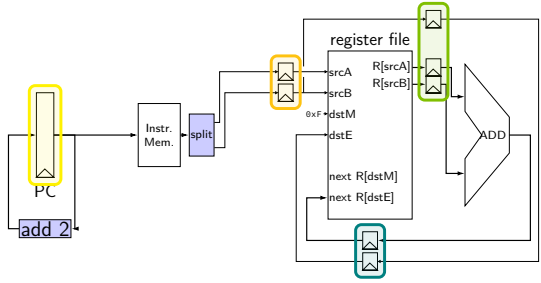
```
// initially %r8 = 800,
//           %r9 = 900, etc.
addq %r8, %r9
addq %r10, %r11
addq %r12, %r13
addq %r9, %r8
```



	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

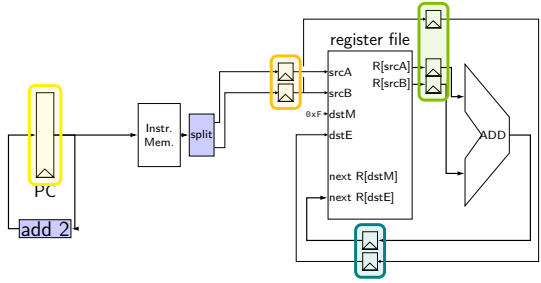
```
// initially %r8 = 800,
//           %r9 = 900, etc.
addq %r8, %r9
addq %r10, %r11
addq %r12, %r13
addq %r9, %r8
```



	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
```

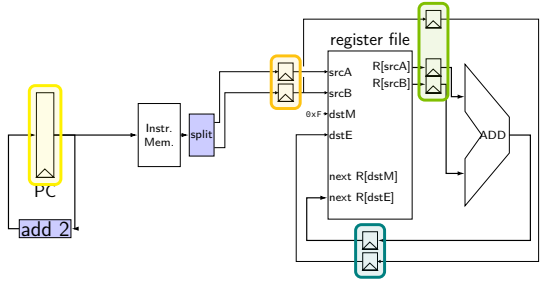


	fetch	fetch/decode		decode/execute			execute/writeback	
cycle	PC	rA	rB	R[srcA]	R[srcB]	dstE	next R[dstE]	dstE
0	0x0							
1	0x2	8	9					
2	0x4	10	11	800	900	9		
3	0x6	12	13	1000	1100	11	1700	9
4		9	8	1200	1300	13	2100	11
5				1700	800	8	2500	13
6							2500	8

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)