Constructive Computer Architecture

Superscalar in-order machines

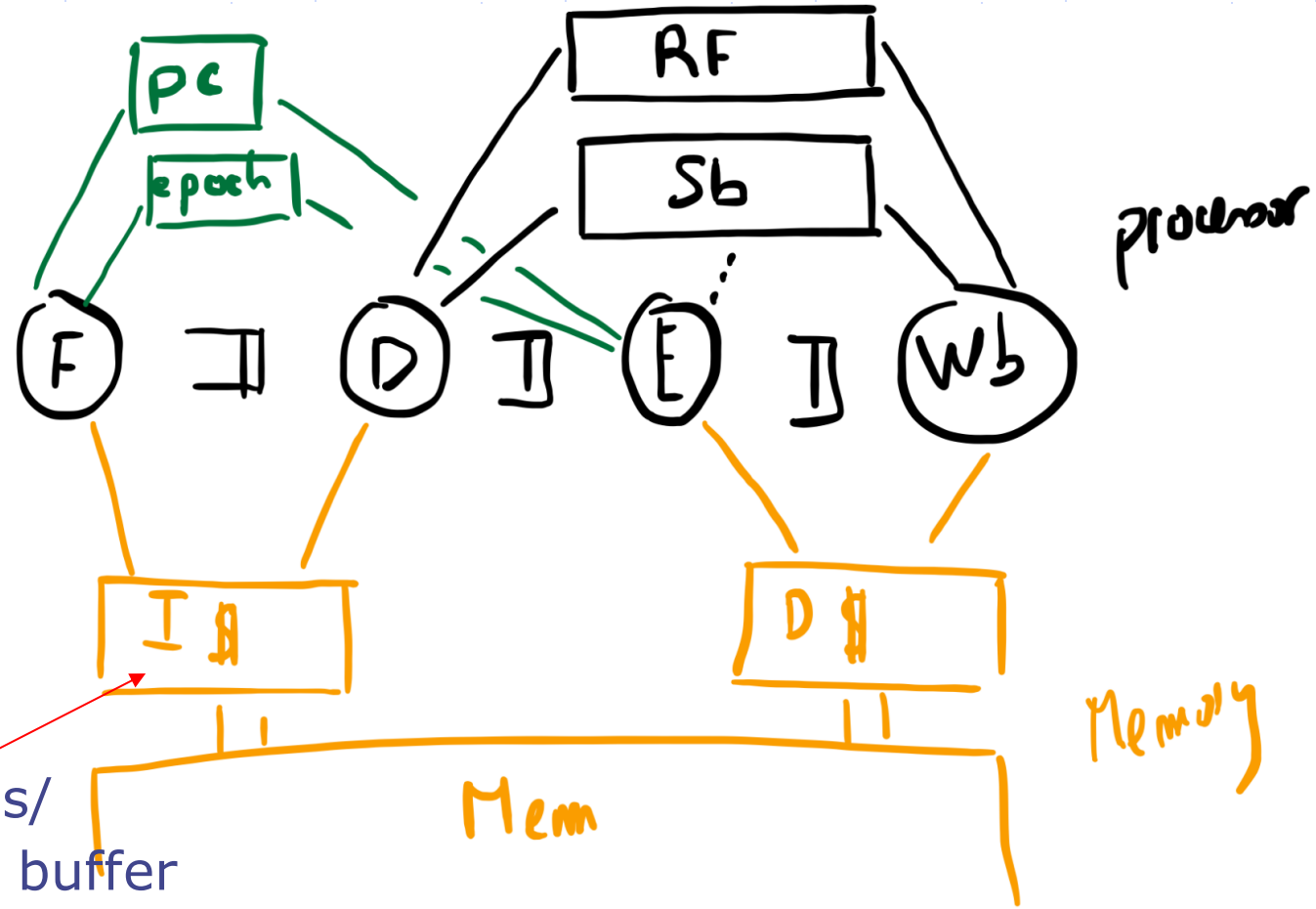
Constructive Computer Architecture

Superscalar in-order machines

Or, going wider

Thomas Bourgeat and Martin Chan

Starting point



No stores/
No store buffer

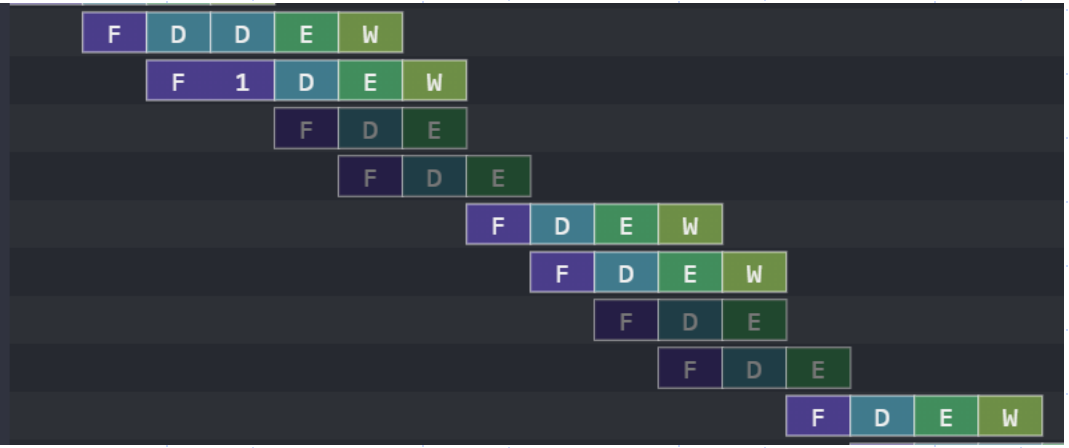
How to go faster?

- Increase number of useful cycles per program
  1. On branch misprediction (seen last lecture)
  2. On load miss
  3. When stalling for dependencies
- Improve program
- Speed up clock

Time =
Instructions/Program *
Cycles/Instruction *
Time/Cycle

Branch Mispredictions

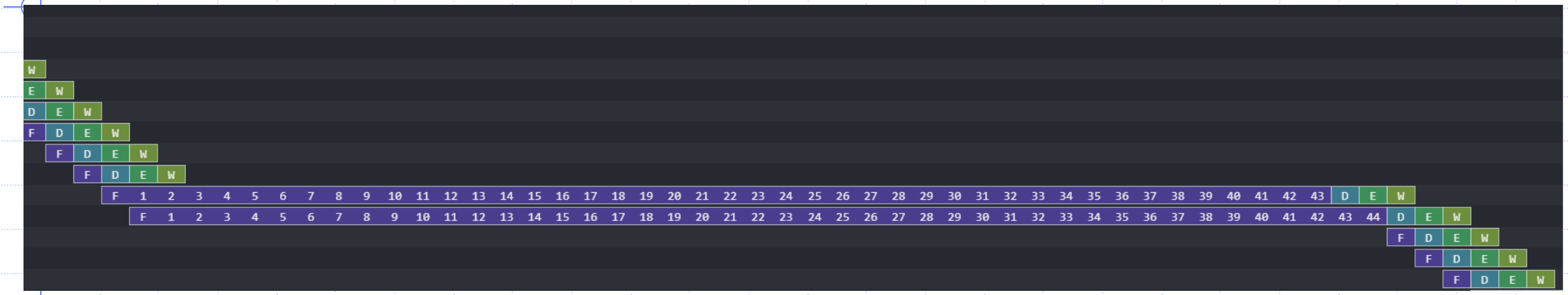
```
: 0x000010bc: addi    s1, s1, 1486
: 0x000010c0: j       pc + 0x8
 0x000010c4: lbu     a0, 0(s0)
 0x000010c8: addi    s0, s0, 1
: 0x000010c8: addi    s0, s0, 1
: 0x000010cc: jal     pc - 0xa0
 0x000010d0: bne     s0, s1, pc - 12
 0x000010d4: lw      ra, 12(sp)
: 0x0000102c: addi    sp, sp, -32
```



- Mitigations:
  - Improve branch predictor (last lecture)
    - BHT, BTB, RAS, etc.
  - Faster recovery

All at a price

Instruction Miss



- Mitigations:
  - Prefetch
  - Increase cache size/replacement policy

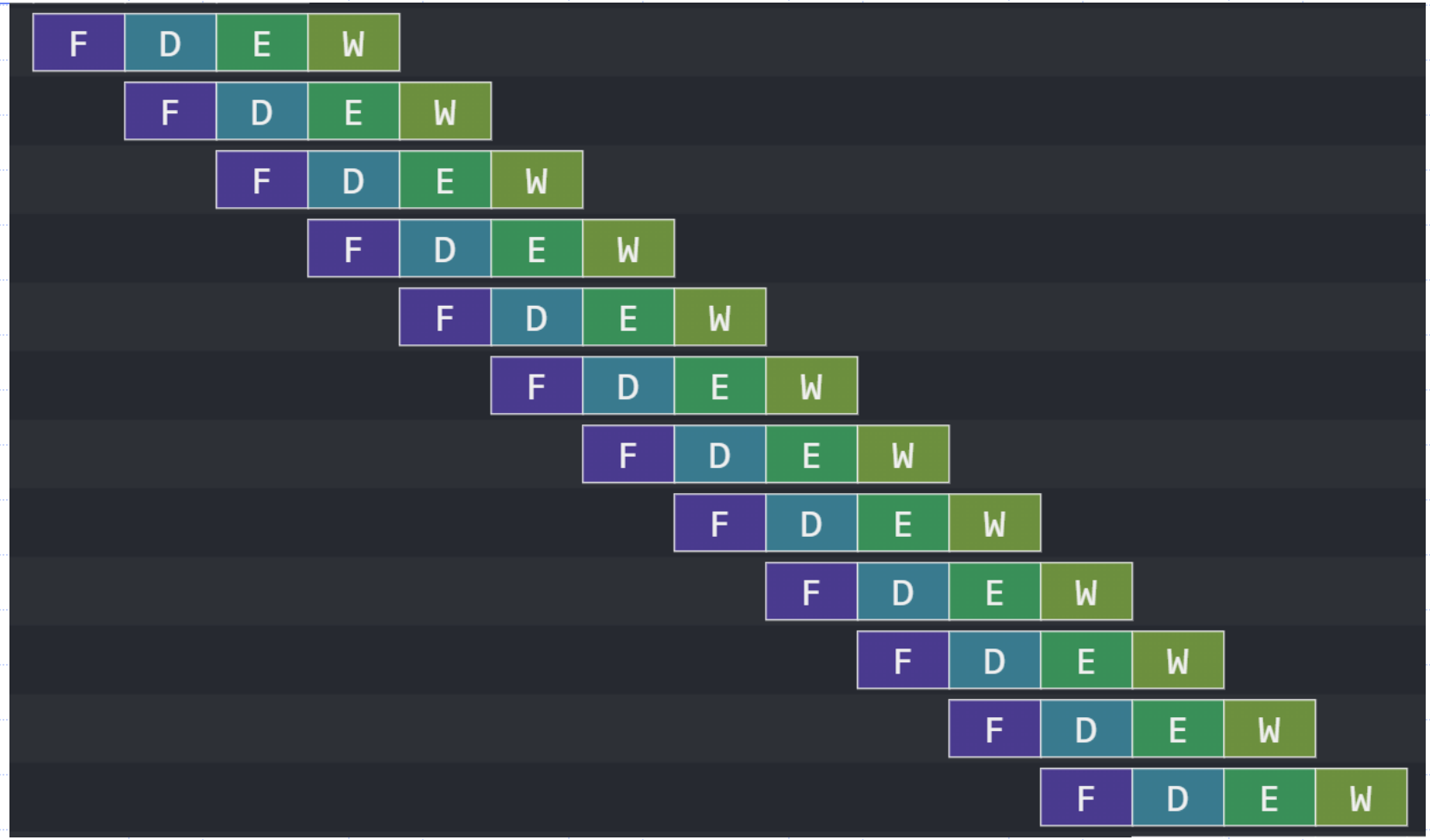
Data Cache Miss

- Mitigations:
  - Store buffer for stores (prioritize loads)
  - Nonblocking cache* (e.g., hit under miss)
  - Pipeline cache
  - Increase cache size/replacement policy
  - Data prefetch?

Data Dependency Stall

- Mitigations:
  - Add bypassing
  - Register renaming for false dependencies (not covered)

If we fix them all: best case scenario



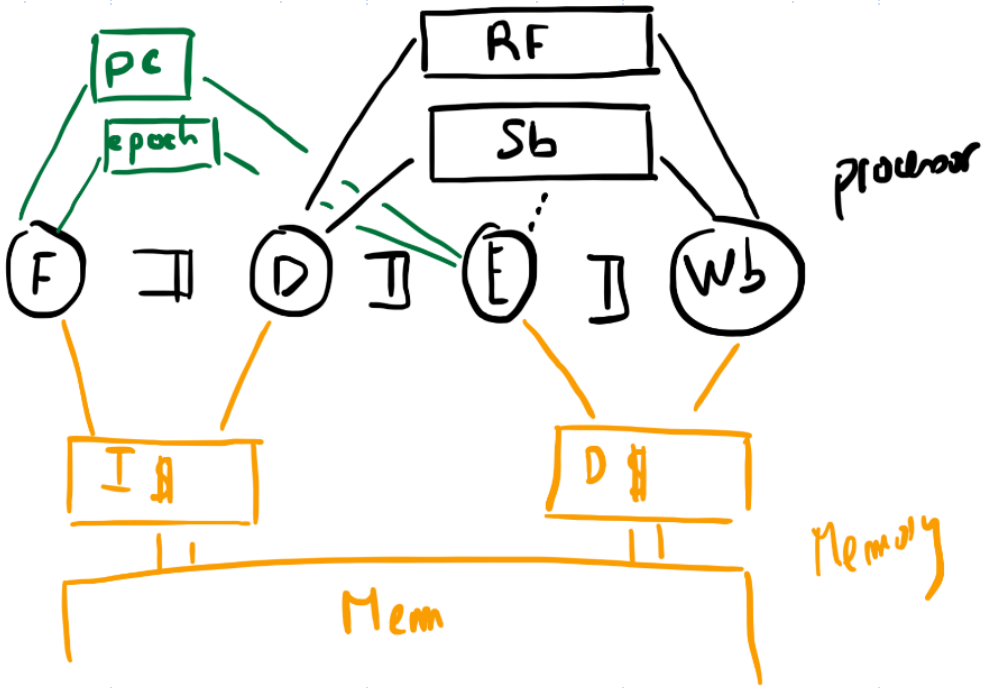
Alternative idea

- We have a single new instruction/cycle (IPC < 1)
  - Can we find more work to do?
- Idea:
  - Do two instructions simultaneously!

Very rich idea: superscalar machines

High-level picture

- We will need to decode multiple times
- We will need to write multiple registers
- (and some more)

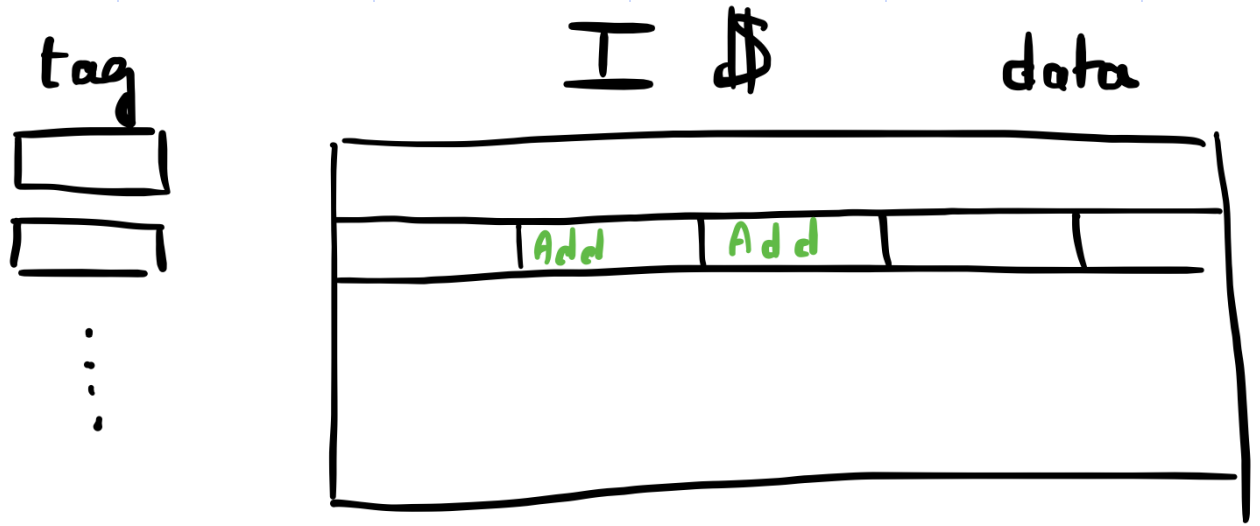


Outline

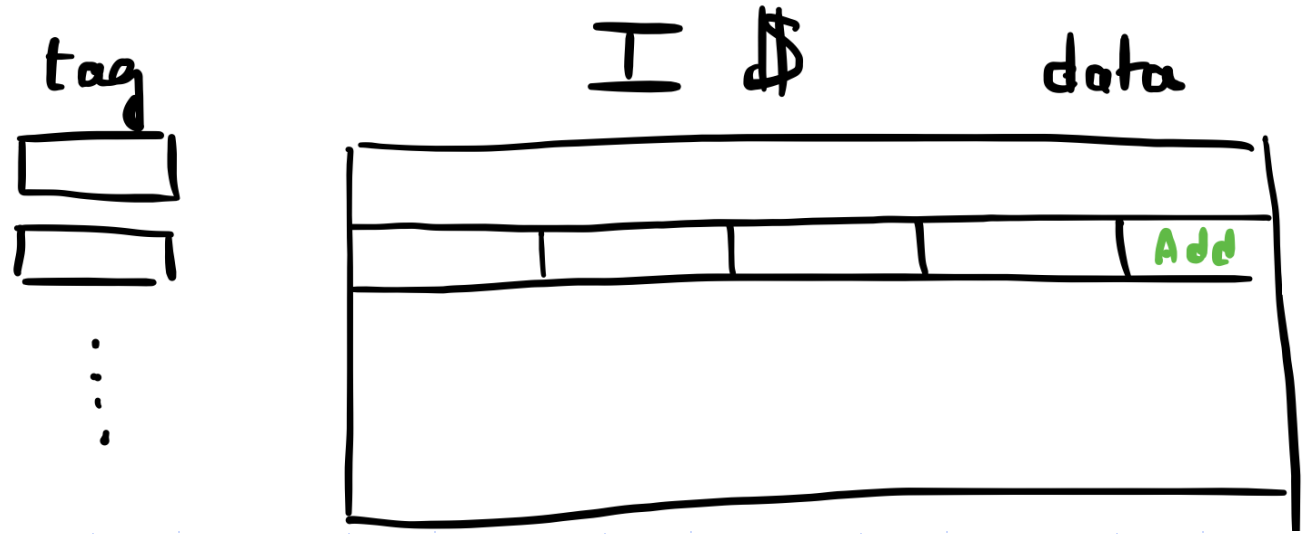
- Let’s consider all the stages
  - Fetch
  - Decode
  - Execute
  - Writeback
- What changes to the processor will be required for each stage?

Brainstorming superscalar Fetch

- Fetch 2 instructions:
  - Easy if instructions consecutive within one cache line of ICache



Fetch – Problem at the boundary

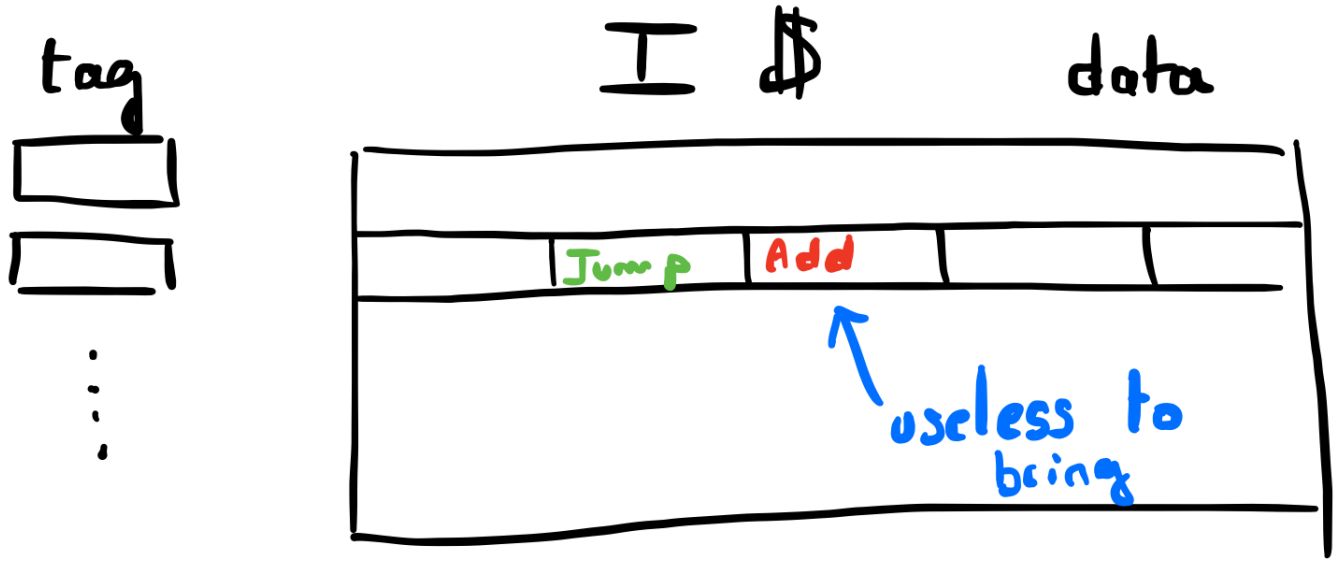


pc+4 not in the same cache line,
(maybe not even in cache)

Revised solution

- Idea:
  - ~~Always do two instructions simultaneously!~~
  - Try to do two instructions simultaneously, sometimes do only one if it is too hard (e.g. cache boundary)

Not always useful – that's ok



Summary ICache

```
typedef struct
  {Word ins1; Maybe#(Word) ins2;} OneOrTwoWords deriving (…);

interface Cache;
  method Action req(MemReq req);
  method ActionValue#(   OneOrTwoWords   ) resp();
  method ActionValue#(LReq) lineReq;
  method Action lineResp(Line r);
endinterface
```

Implementation:

When enqueuing into hitQ, enqueue pair of 2 words if you can

Fetch

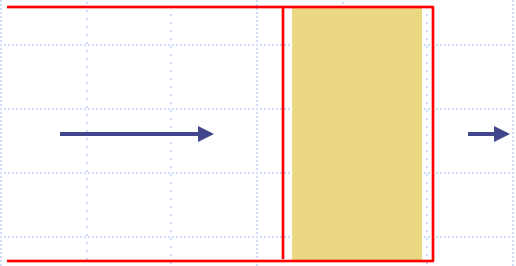
```
// OLD
let req = Mem {byte_en : 0, addr : pcf[1], data : 0};
toImem.enq(req);
f2d.enq(F2D{pc: pcf[1], ppc : pcf[1]
// NEW
let req = Mem {byte_en : 0, addr : pcf, data : 0};
toImem.enq(req); // Triggers a response with 1 or 2 inst
f2d.enq1(F2D{pc: pcf, ppc : pcf+4, epoch: epoch});
if (notLineBoundary(pcf)) begin
    f2d.enq2(F2D{pc: pcf+4, ppc : pcf+8, epoch: epoch});
    pcf <= pcf+8;
end else pcf <= pcf+4;
```

We need a queue where we can enqueue multiple times per rule

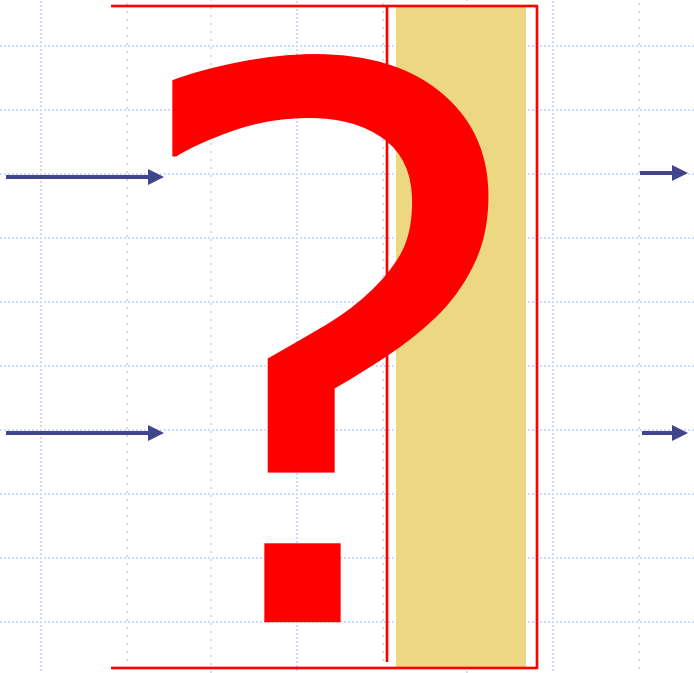
(An aside: you may want two predictor ports, or keep using just one)

Everyone's favorite kind of interface

Regular single FIFO



Superscalar FIFO



Superscalar FIFO- example

f == []

Cycle 0 “f.enq1(13), f.enq2(42)”

f==[13;42]

Cycle 1 “f.enq1(24), f.deq1()”

f==[42;24]

Cycle 2 “f.enq1(76), f.enq2(84), f.deq1(), f.deq2”

f==[76;84]

Interface for Superscalar FIFO

```
interface SuperFIFO#(type t);
    method Action enq1(t x);
    method Action enq2(t x); // implies enq1 called too
    method Action deq1;
    method Action deq2; // implies deq1 called too
    method t first1;
    method t first2; //…
endinterface
```

How to implement this interface?

Sketch – Rotating FIFOs

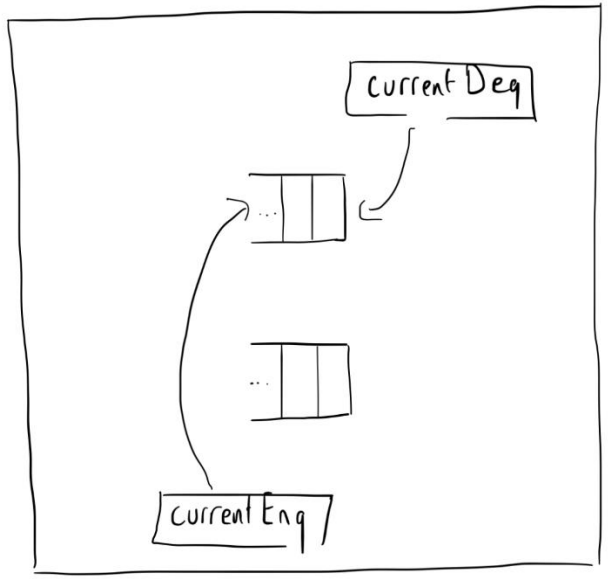
2 internal queues

Track which queue holds the oldest value: 1 bit reg currentDeq

Track which queue should be enqueued into next: currentEnq

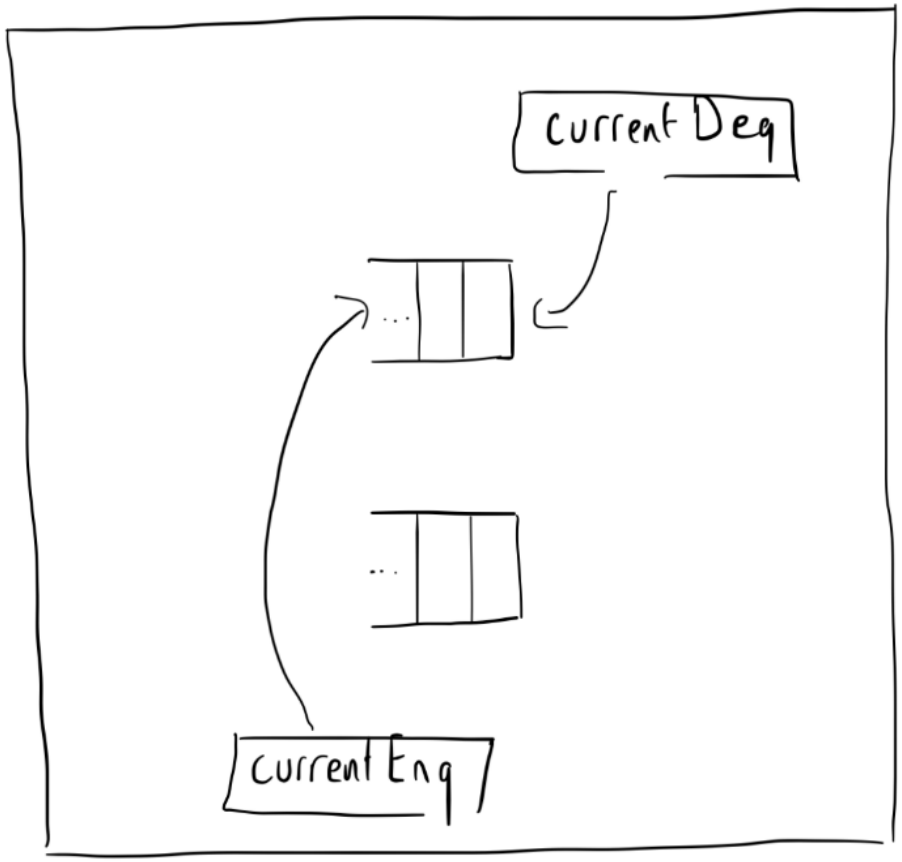
The two enqueue push into FIFO (currentEnq, currentEnq+1)

The two dequeue pull from FIFO (currentDeq, currentDeq + 1)



Rotating Superscalar FIFO

f == []

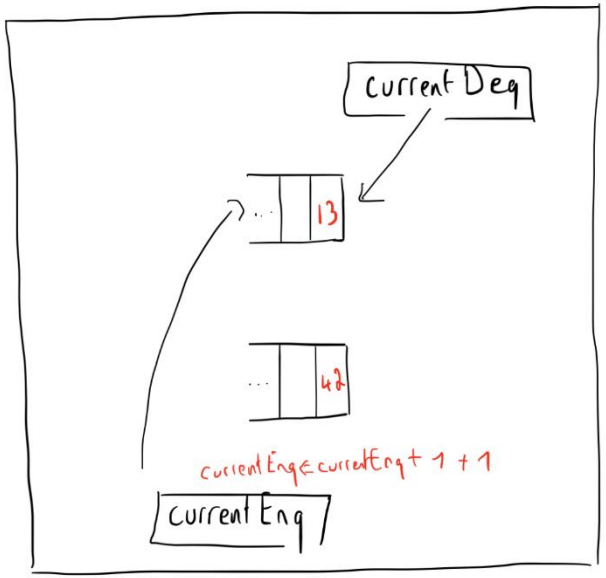


Rotating Superscalar FIFO

f == []

Cycle 0 “f.enq1(13), f.enq2(42)”

f==[13;42]



Rotating Superscalar FIFO

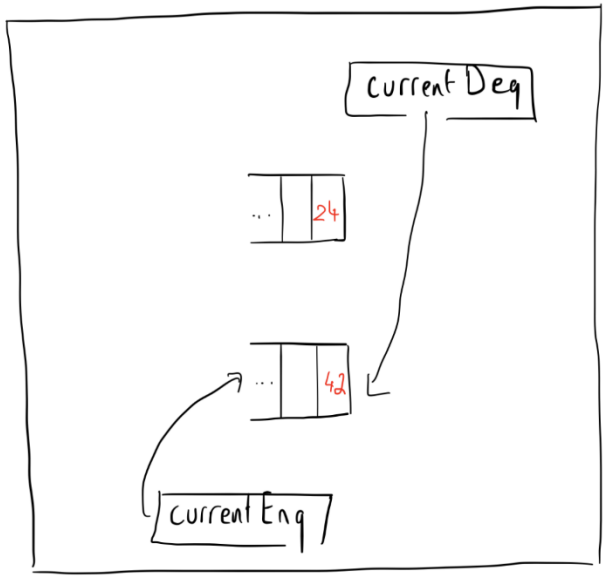
f == []

Cycle 0 “f.enq1(13), f.enq2(42)”

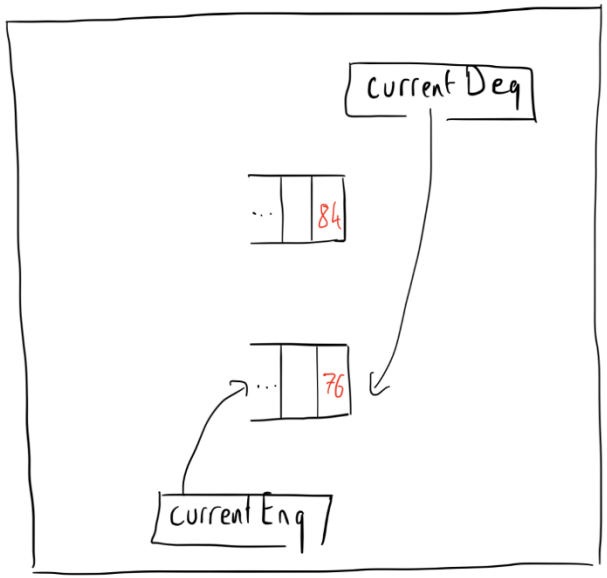
f==[13;42]

Cycle 1 “f.enq1(24), f.deq1()”

f==[42;24]



Rotating Superscalar FIFO



f == []

Cycle 0 “f.enq1(13), f.enq2(42)”

f==[13;42]

Cycle 1 “f.enq1(24), f.deq1()”

f==[42;24]

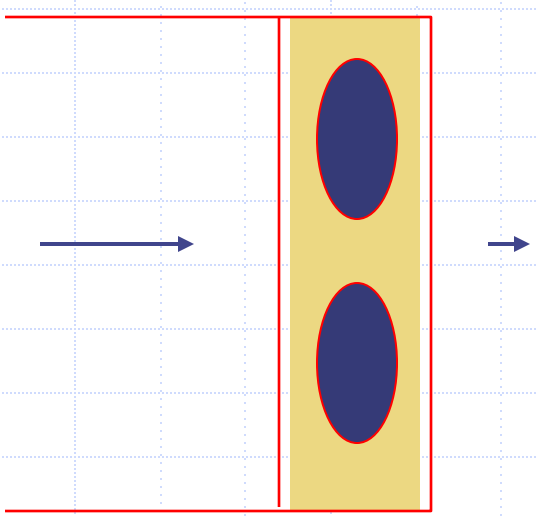
Cycle 2 “f.enq1(76), f.enq2(84), f.deq1(), f.deq2”

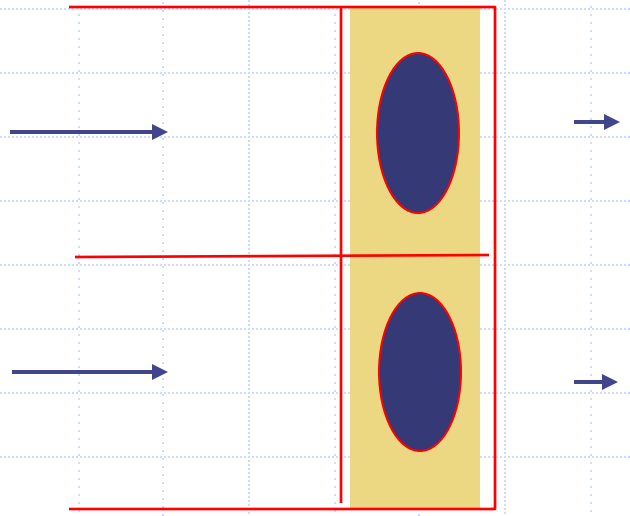
f==[76;84]

Code in appendix “SuperFIFO.bsv”

Subtle FIFO difference

- FIFO with single enq, deq for two tokens vs.
  FIFO with two enq, deq for one token each

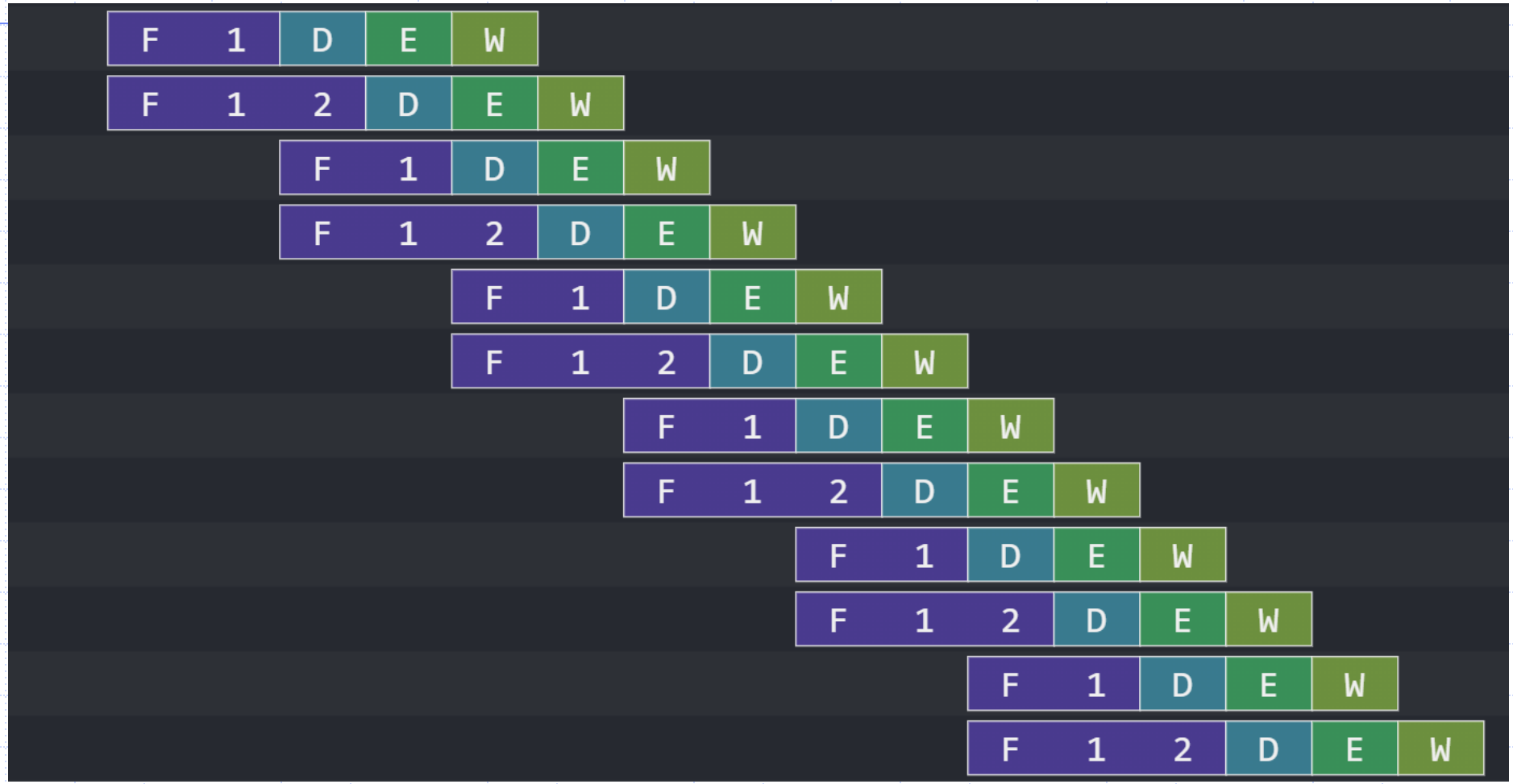




Wrap OneOrTwoToken iMem FIFO with SuperFIFO

Fetch only

Useful?



Decode

- Requirements:
  - Dequeue two elements from f2d (ins1,ins2)
  - Push multiple times into d2e
  - Duplicate decoding logic
  - Needs more register file and scoreboard reads/write
  - Handle ins1 write a register read by ins2
  - Decode (ins1, ins2), outstanding dependency on ins2, ins1 ok. Stall?

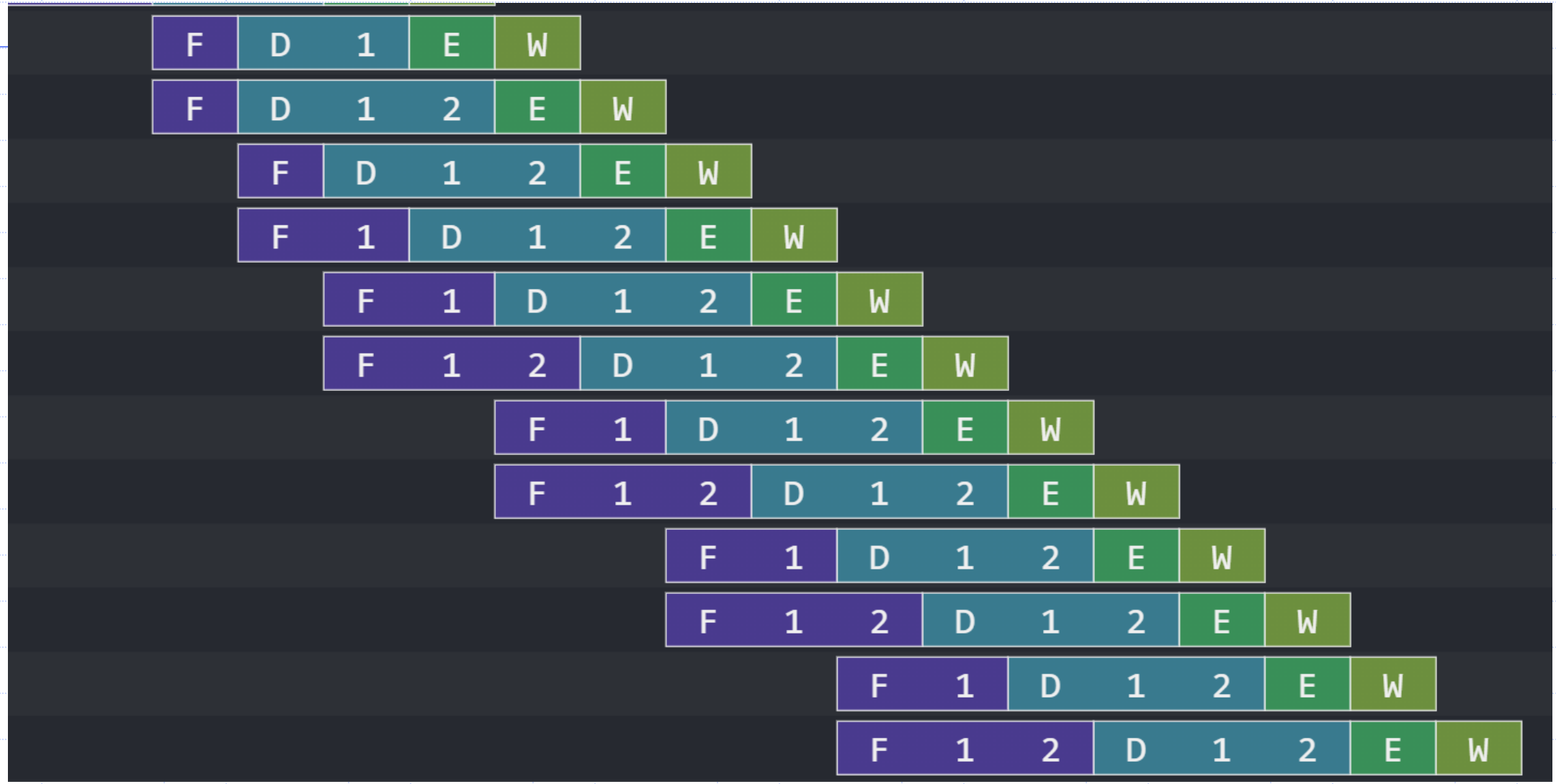
Partial stalling in Decode

```
if (noDependency(ins1,sb) && noDependency(ins2,sb) &&
      noDependencyBetween(ins1, ins2)) begin
       d2e.enq1("ins1");
       d2e.enq2("ins2");
       f2d.deq2;
end else if (noDependency(ins1,sb)) begin
       d2e.enq1("ins1");
       f2d.deq1;
end // otherwise stall
```

Is the second case for correctness or performance?

+ Decode

(Not the full picture)



Check your Decode

- Some bugs might not show up in every program!
- Think about how things may break
- Write or find RISC-V unit tests
  - Check for each of the cases we discussed
    - I1 not in SB, I2 not in SB, I2 independent from I1
    - I1 not in SB, I2 not in SB, but I2 depends on I1
    - etc.
- Key is thinking about dependencies

Execute

- Execute 2 instructions:
  - Duplicate ALU? Possible – some area tradeoff
  - Duplicate memory? Much harder/impossible
  - New kind of structural hazards:
    - ins1 and ins2 are memory instructions
    - More generally, any ins1 and ins2 using the same resource
      - (multiplier, floating point unit, etc.)

Execute – ALU, Memory, Control

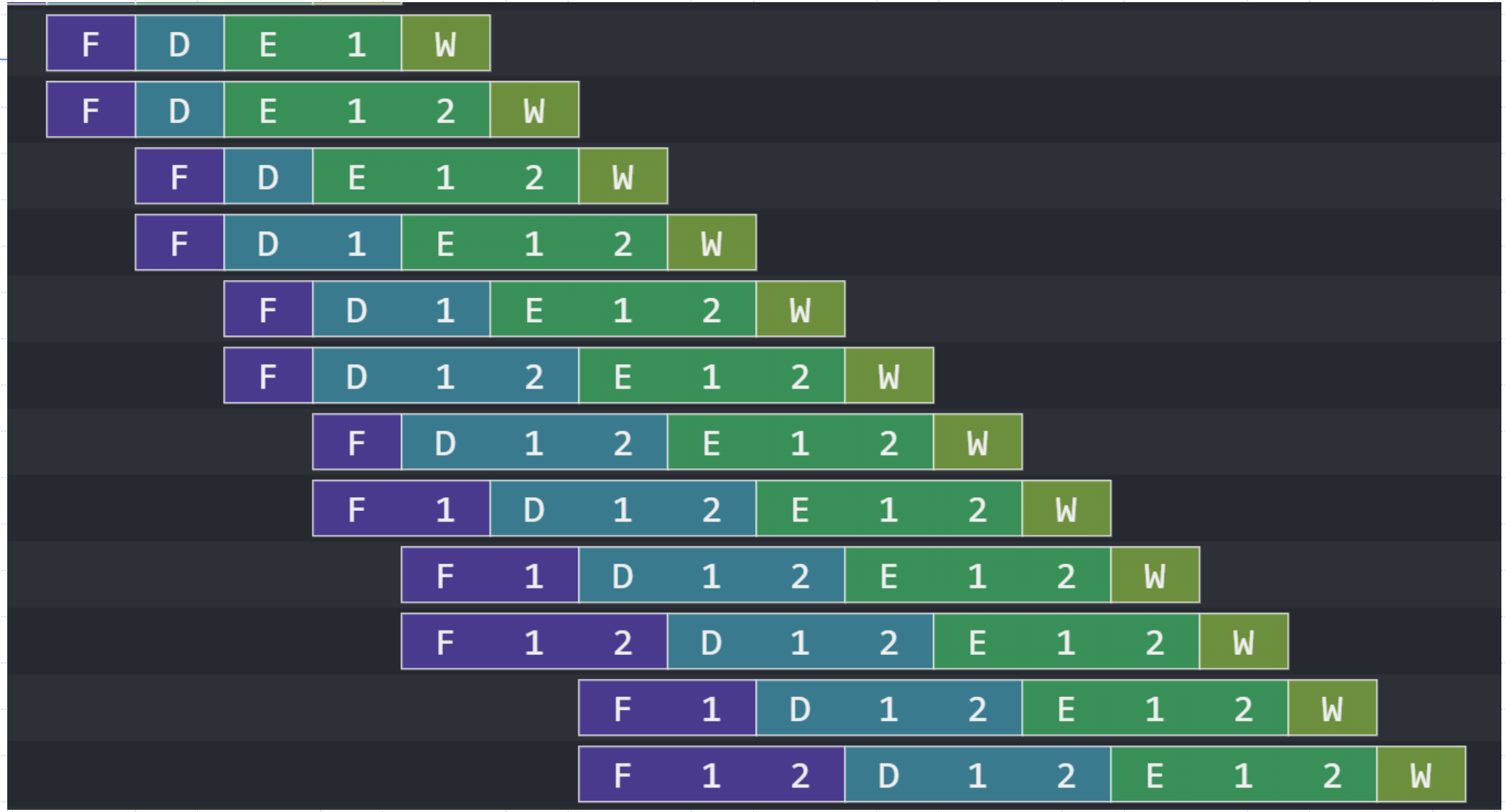
- Many combinations in the sequence (ins1,ins2):
  - Misprediction and both incorrect?
  - Misprediction then correct instruction?
- Easiest approximation:
  - Do control (br/jmp) one-by-one
  - Do memory one-by-one
  - Do arithmetic two-by-two
  - *Check* squashing two-by-two, then try again

Simple Execute

```
let ins1 = d2e.first1(); let ins2 = d2e.first2();
d2e.deq1(); // at least process ins1
if (ins1.epoch != epoch[0] && ins2.epoch != epoch[0]) begin
  d2e.deq2(); // invalid ins1 and ins2
  squash ins1 AND ins2D
end else if (ins1.epoch != epoch[0]) begin
  squash ins1, don't touch ins2, try again next cycle
end
else if (isALU(ins1) && isALU(ins2) && ins2.epoch == epoch[0]) begin
   d2e.deq2();
   execute both ins1 and ins2 (duplicate ALU circuit)
end else begin
   be modest and only process ins1 (as in original design)
end
```

+ Execute

(Also not the full picture)



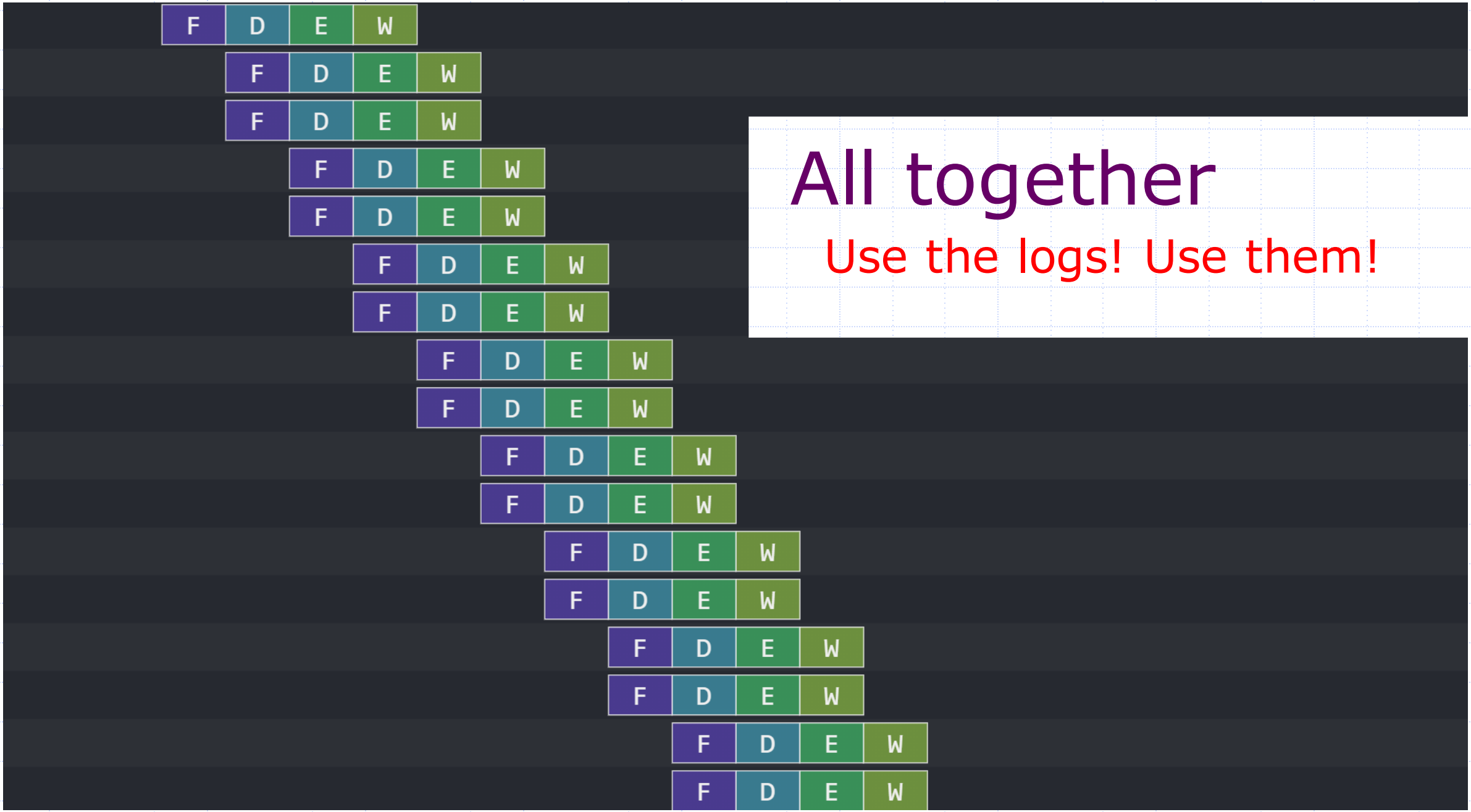
Check your Execute

- Some bugs might not show up in every program!
- Think about how things may break
  - dMem and e2w mismatch
- Write or find RISC-V unit tests
  - Check for each of the cases we discussed
    - Some things are tricky with control and memory
    - Draw a 3x3 matrix

Brainstorming superscalar Writeback

- Writeback 2 instructions:
  - More ports for the register file
  - More ports for the scoreboard
- Structural hazard for memory/MMIO instructions
- What if the two instructions write the same register?
  - Is it a problem?
  - Easy safe solution: EHR with extra ports (RF and SB)

In reality, register file ports can become expensive, and so can cache ports



All together

Use the logs! Use them!

Teaser - next week and next lab

- So far, processor runs one program
- Why not making it run two programs?
  - pc_x, rf_x[_]
  - pc_y, rf_y[_]
  - (but shared memory, shared other stuff)
- The two programs are independent, unlikely to miss/branch/stall simultaneously.
- We call this Simultaneous Multithreading (SMT)