mp_pipeline

Part 3

Announcements

- Patches:
  - https://courses.grainger.illinois.edu/ece411/sp2024/project.html
  - Patch 1: 2/5/24 (ID: 1d54a44e7d96115c7895c78cbb6192ca91bc7106)
  - Patch 2: 2/11/24 (ID: b756db28f44f72dca5f9e54aa3f5e53130d7e852)
  - Patch 3: 2/13/24 (ID: 969ca8b02b3d4868783c2c2885bd651445f90518)
- Mp_cache releasing later this week

C code testing

- Test more complex operations using C!
  - `make run_top_tb PROG=../testcode/example.c`
- Function calls -> jal/jalr
- Array accesses or pointer dereferencing -> loads/stores
  - Use volatile to make sure these don’t get optimized to reg insts!
  - Use different types to test different sized loads and stores
    - int, short, char, unsigned
  - Casting and pointer arithmetic
- Check sim/bin/*.dis to see the instructions generated

Checkpoint 2 Recap

You have so far accomplished:

- A full pipelined processor! 🥳
  - With idealized memory
- Data hazard handling
  - Forwarding
- Control hazard handling
  - Pipeline flushing

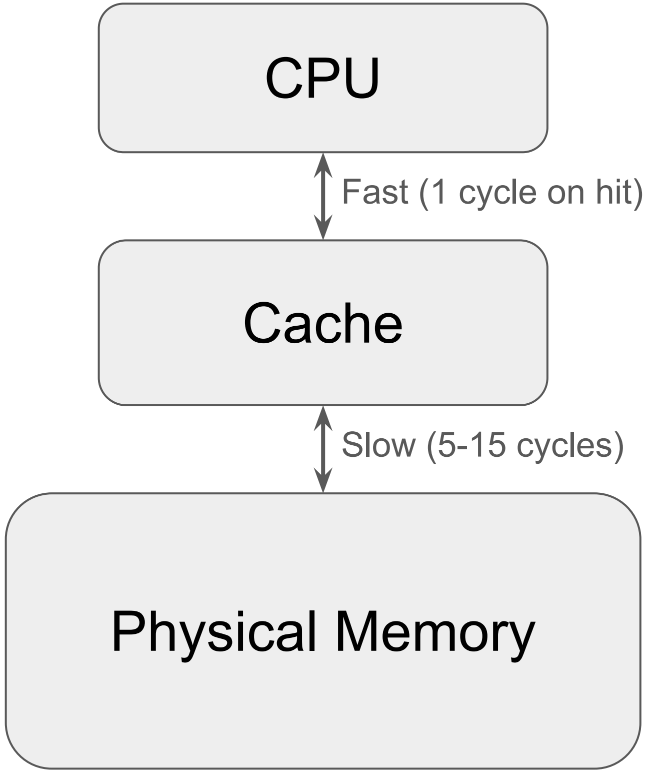
Checkpoint 2 Recap Cont.

Stalling: Stopping part of the pipeline from moving forward to resolve hazards

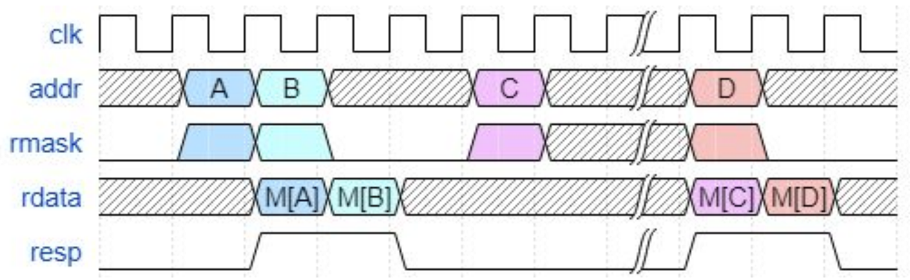
Have you seen stalling yet?

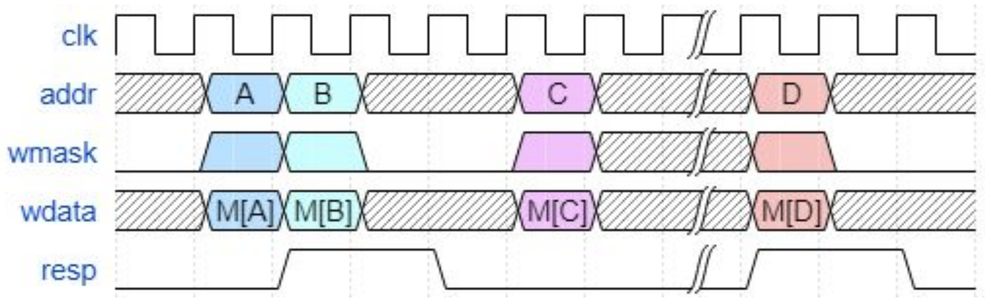
Checkpoint 3 Memory model

- More realistic model
- Memory operation:
  - CPU requests read/write
  - If memory is in cache, hit
  - If not, need to interact with physical memory
- Some memory requests will take longer



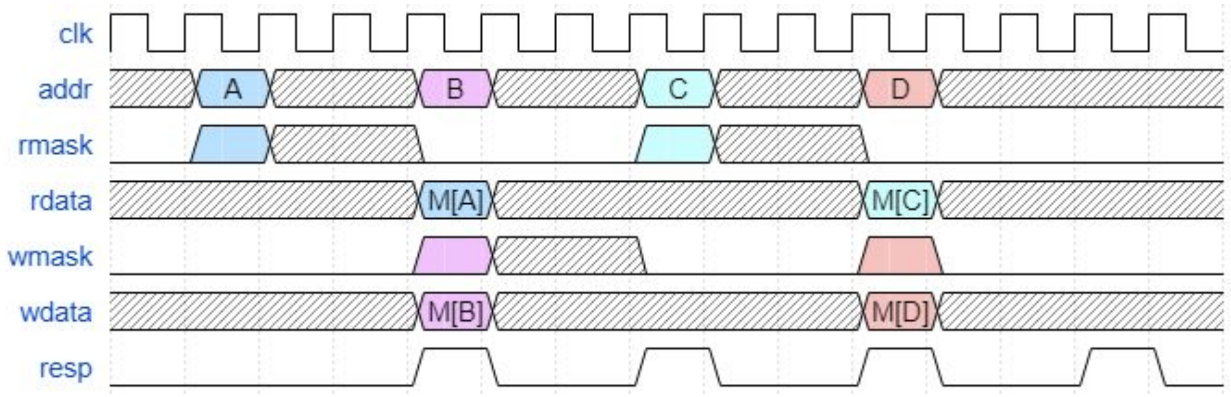
Memory model timing



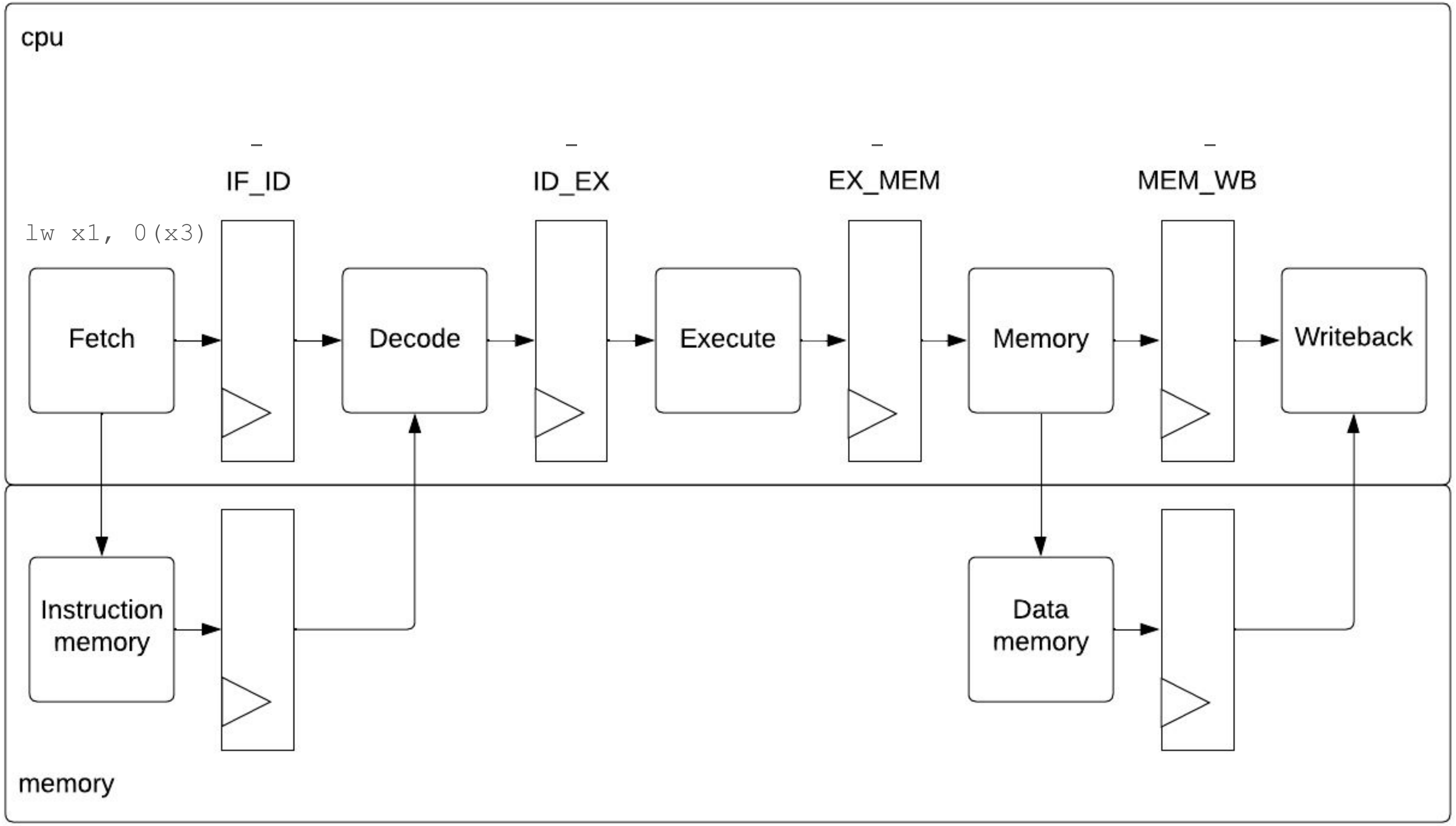


Memory model timing with asm

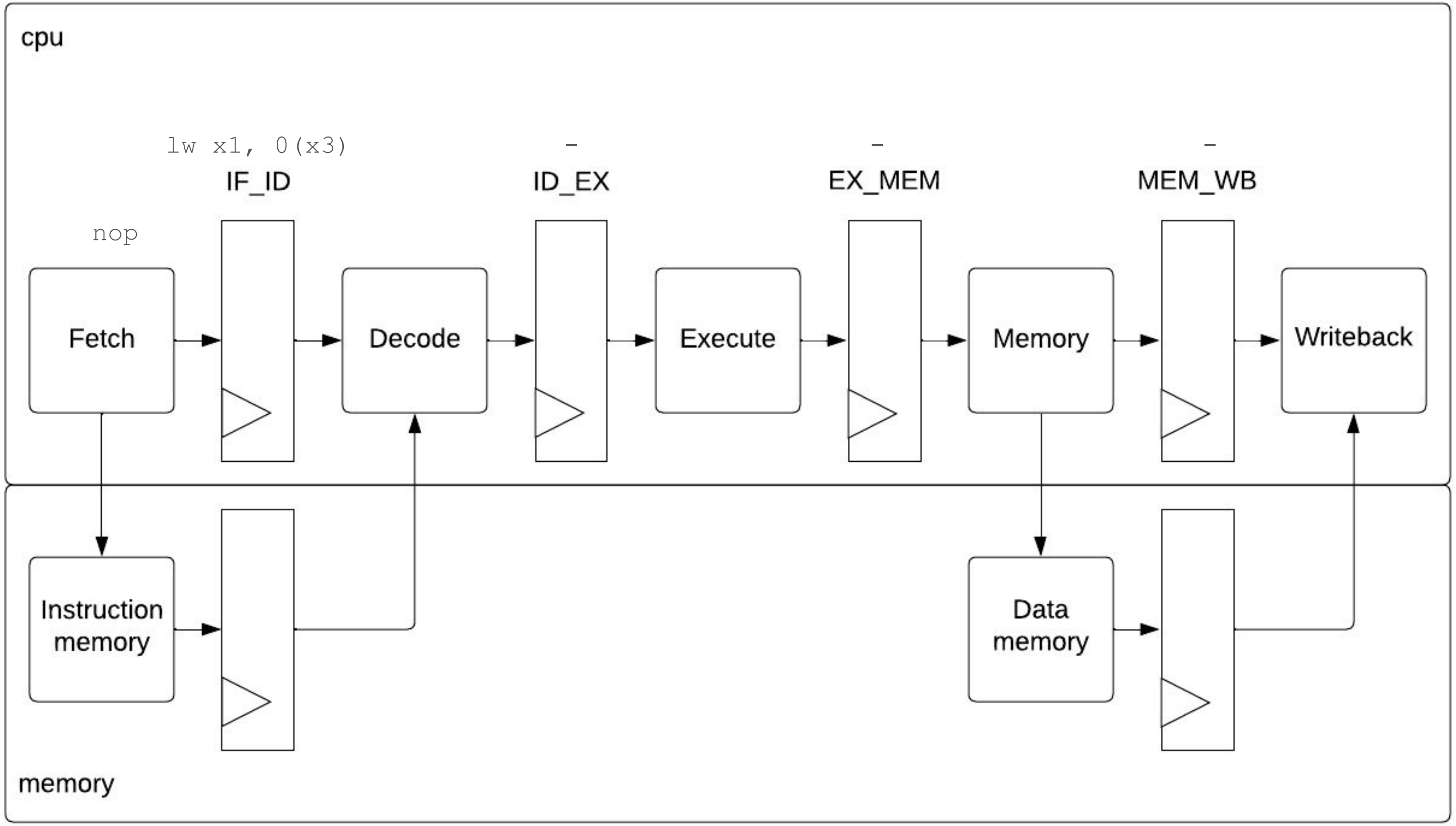
```
lw x1, 0(x5)
???
sw x2, 0(x6)
???
lw x3, 0(x7)
???
sw x4, 0(x8)
```



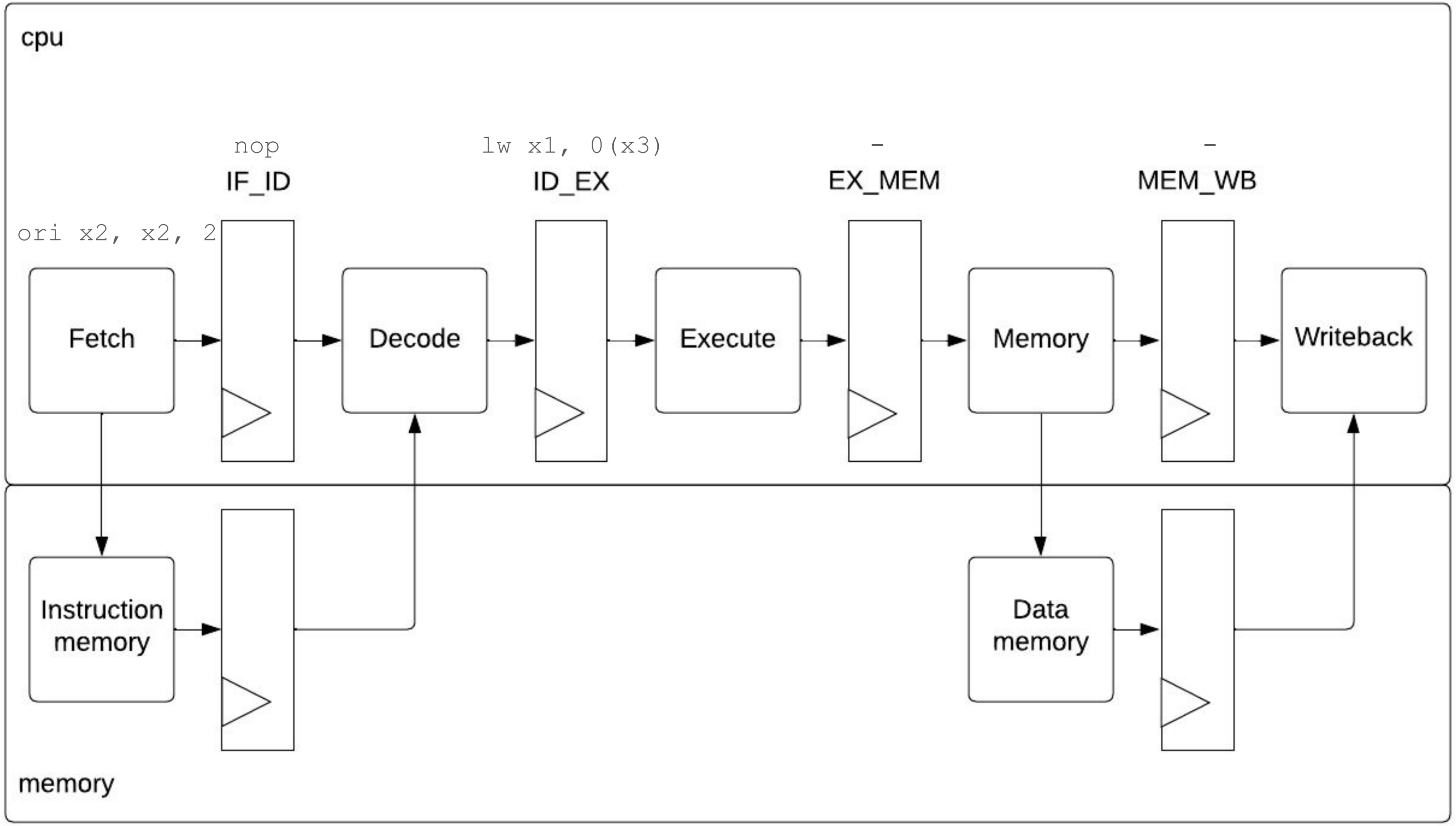
Data Memory stall illustration: Cycle 1



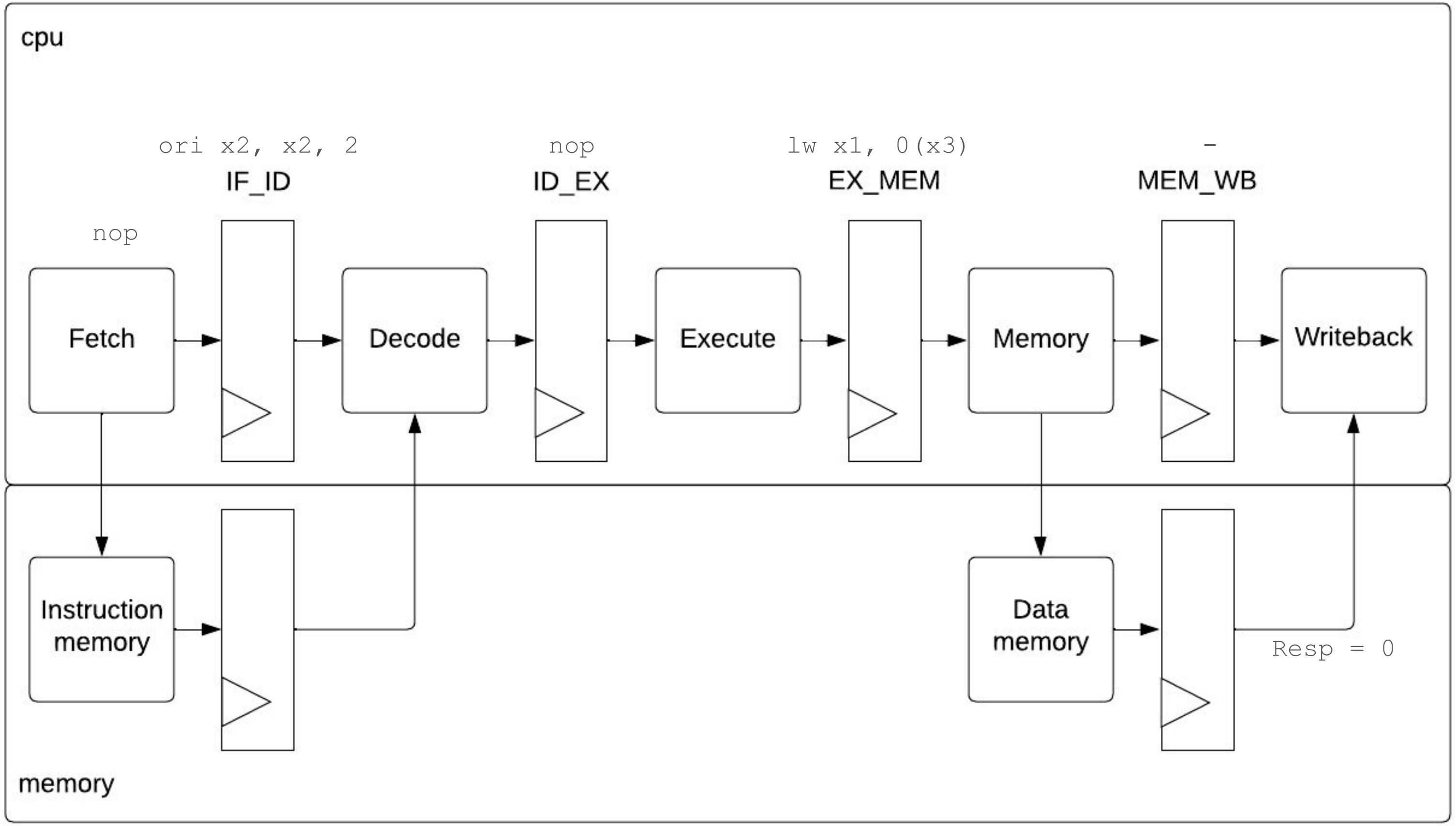
Data Memory stall illustration: Cycle 2



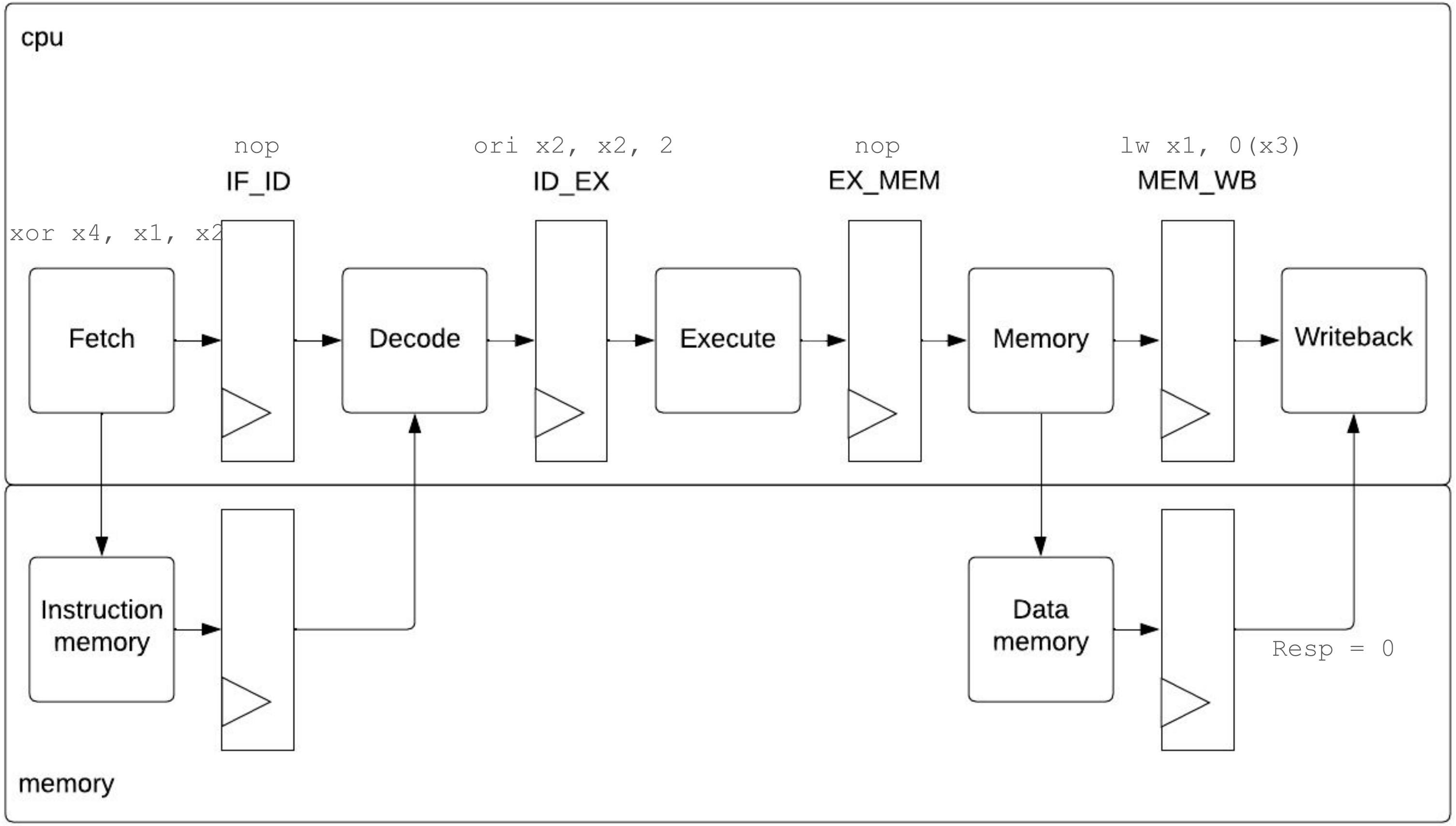
Data Memory stall illustration: Cycle 3



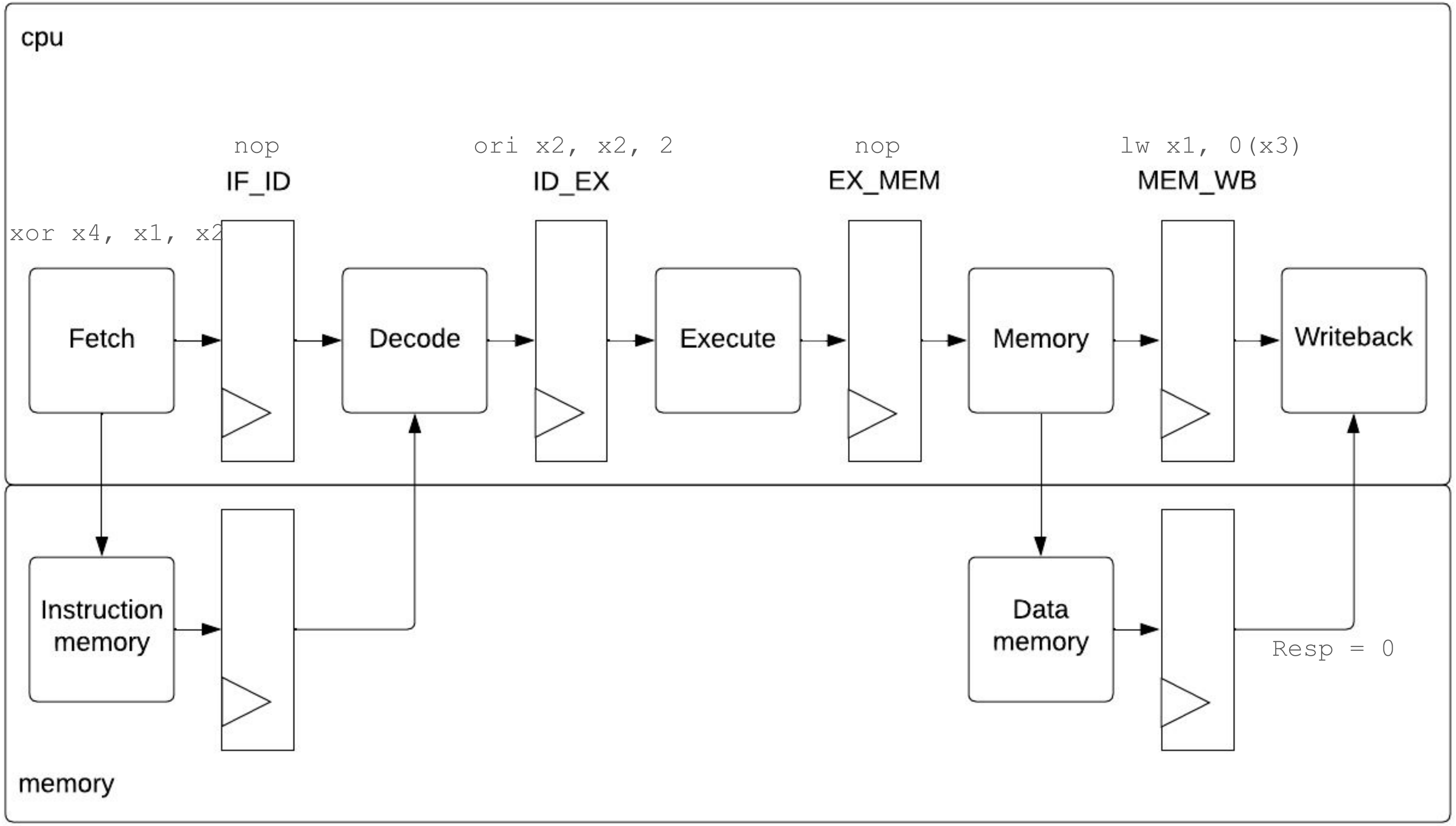
Data Memory stall illustration: Cycle 4



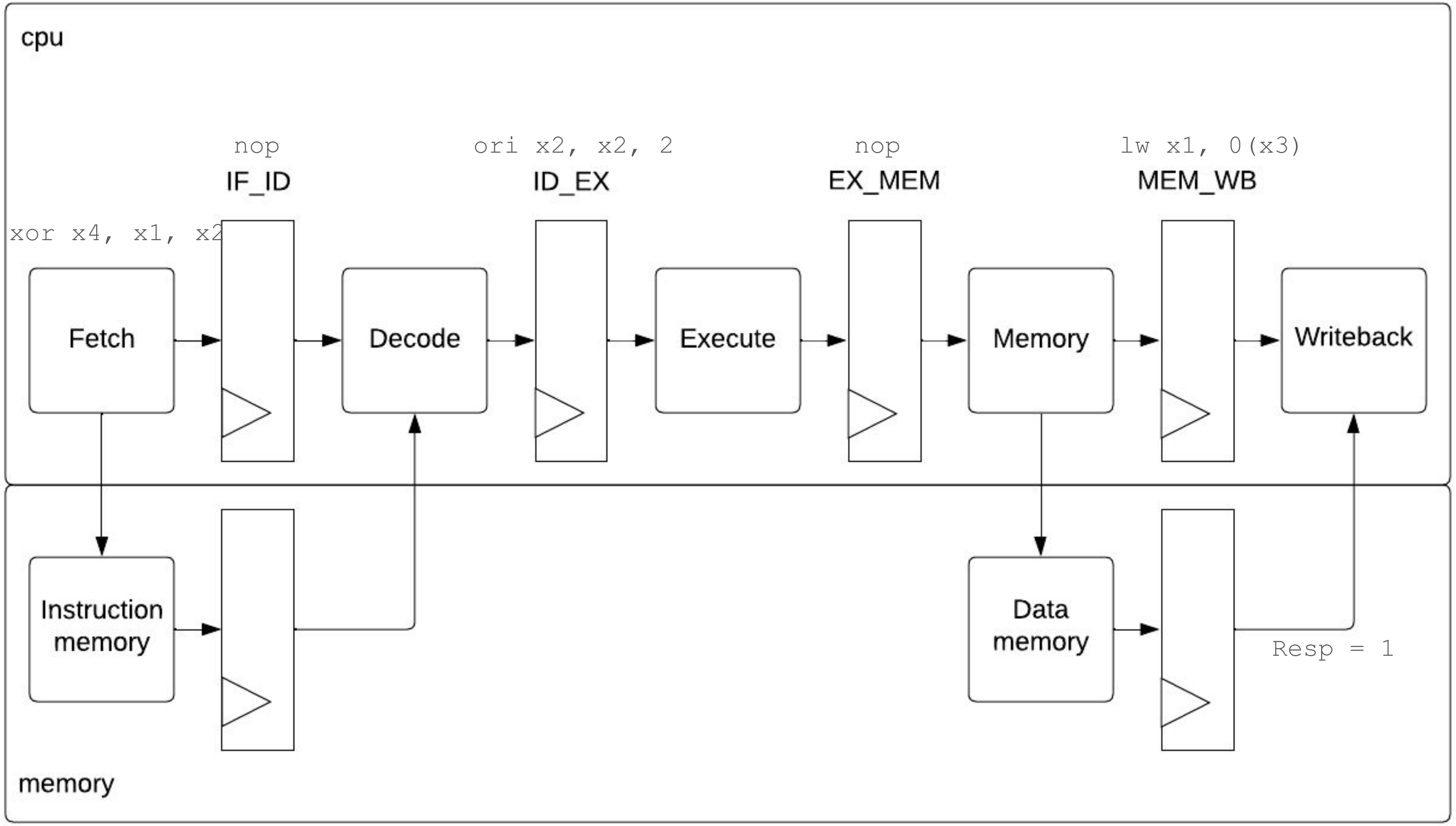
Data Memory stall illustration: Cycle 5



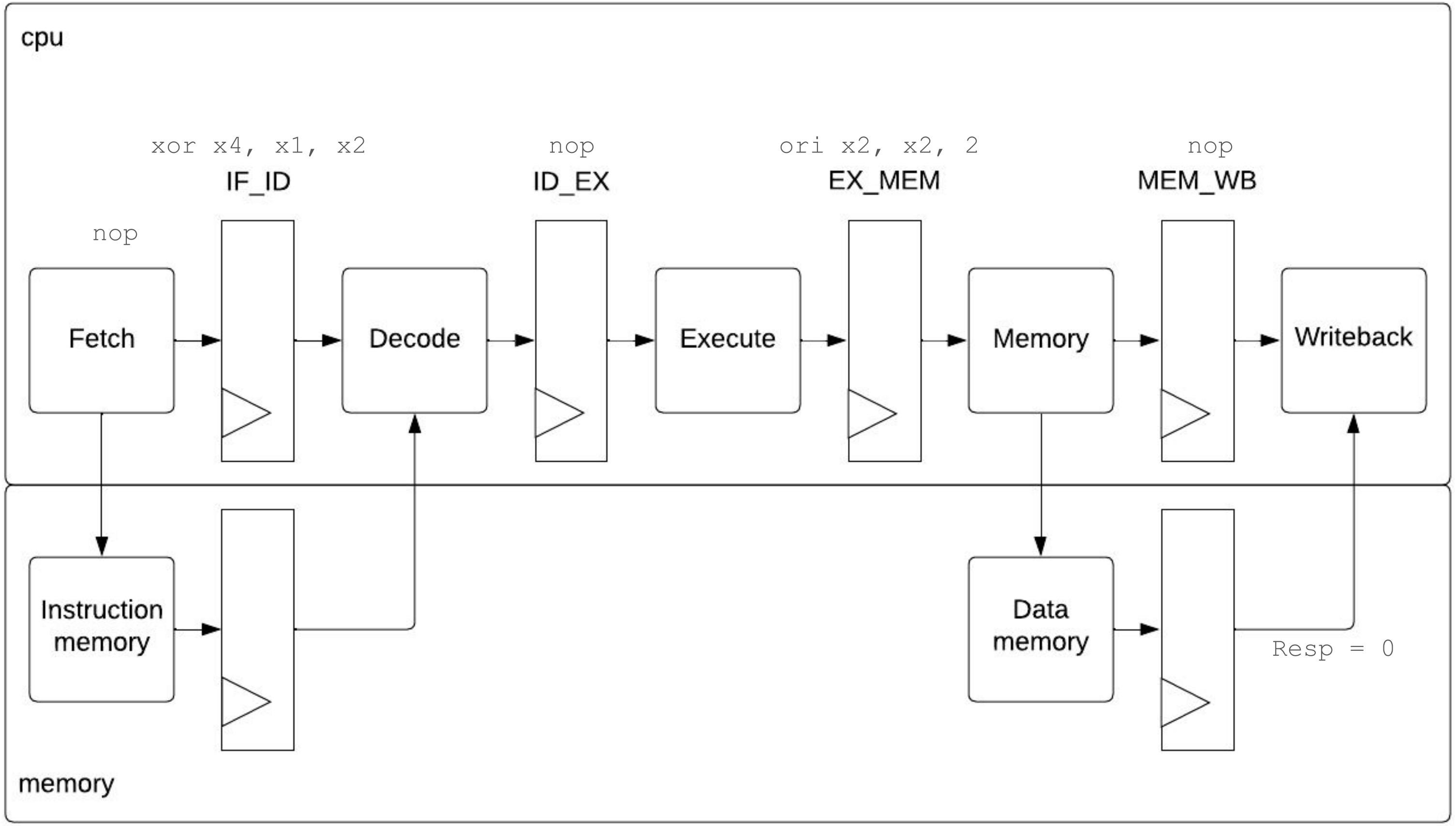
Data Memory stall illustration: Cycle 6



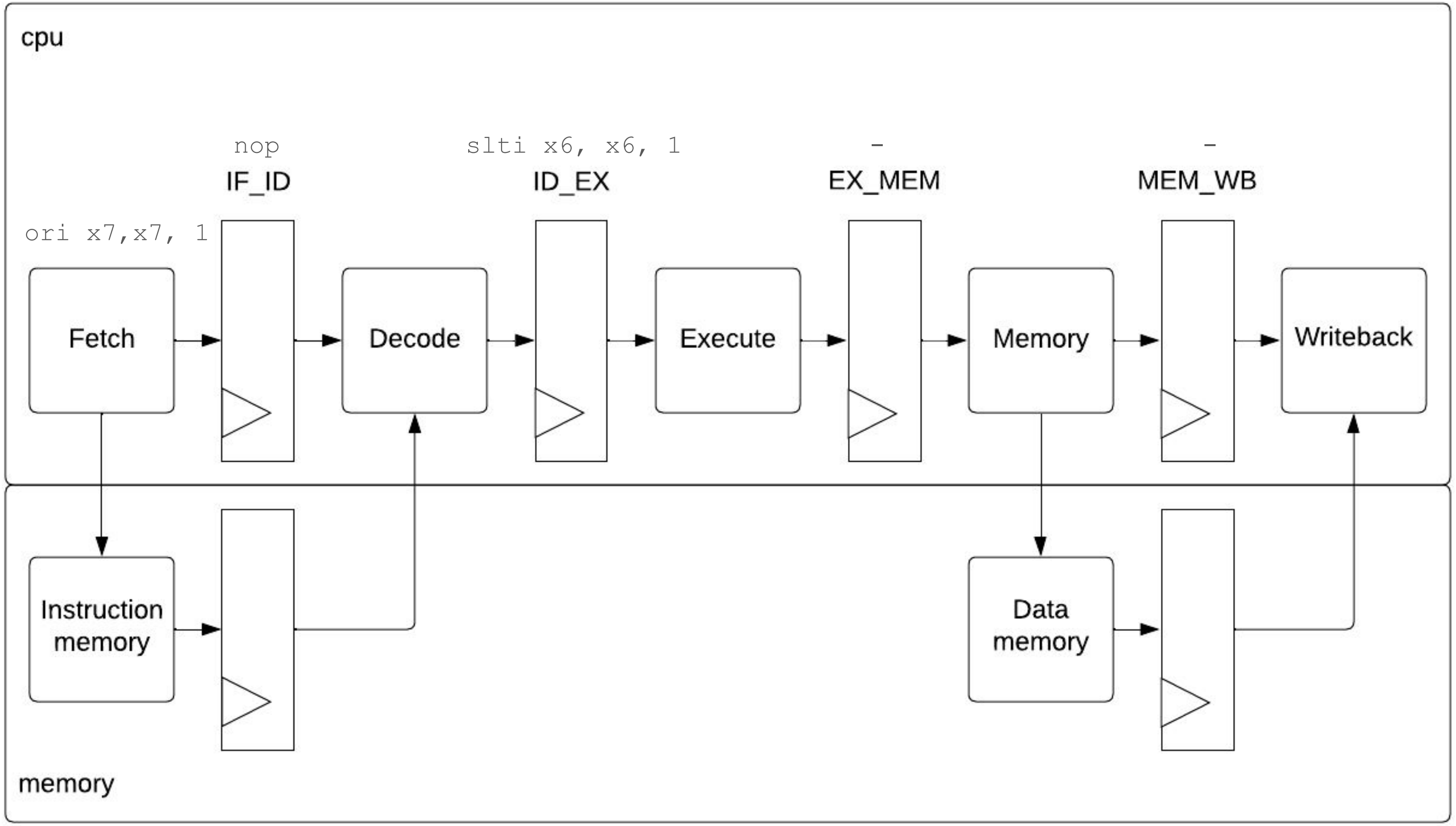
Data Memory stall illustration: Cycle 12



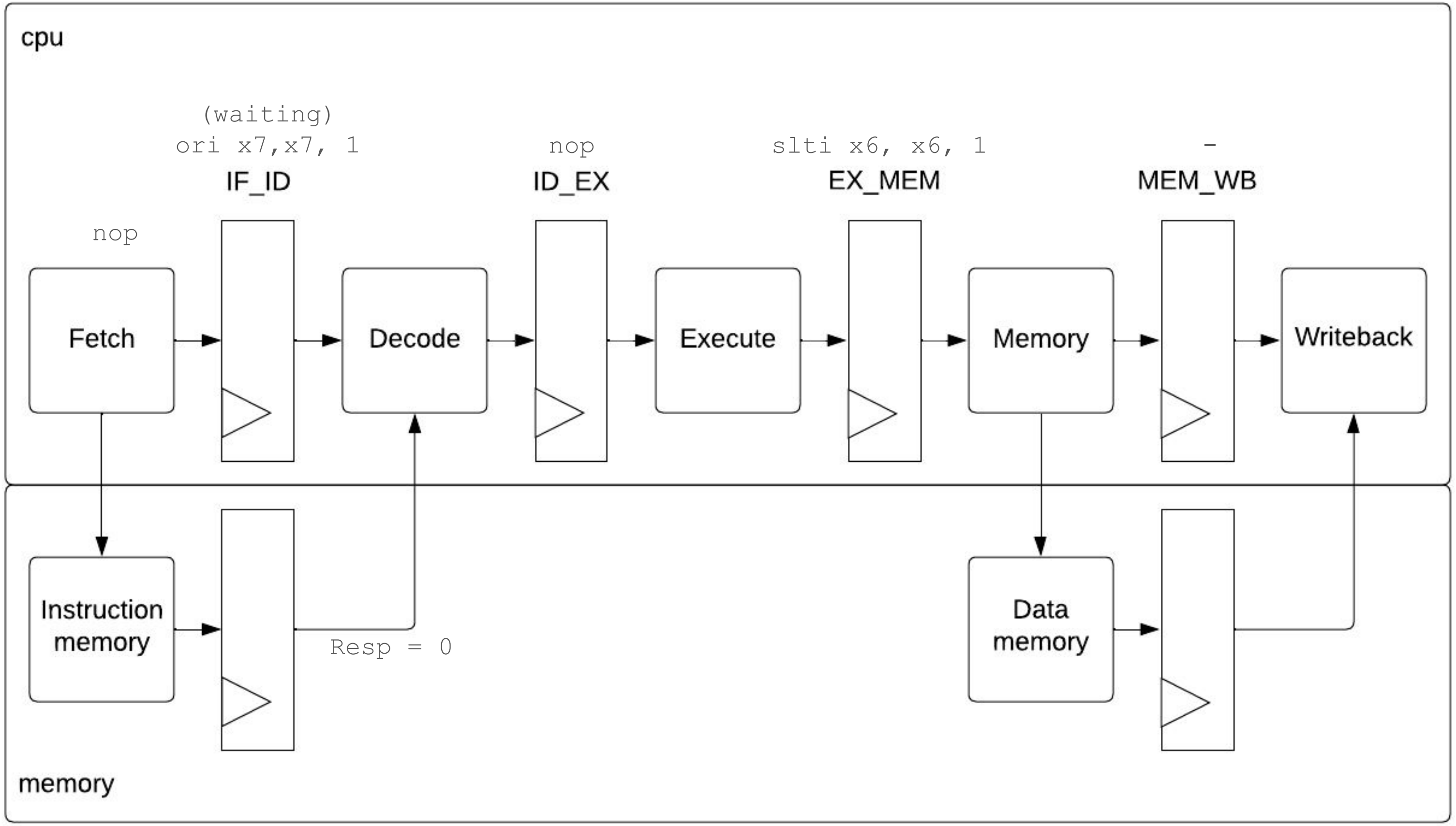
Data Memory stall illustration: Cycle 13



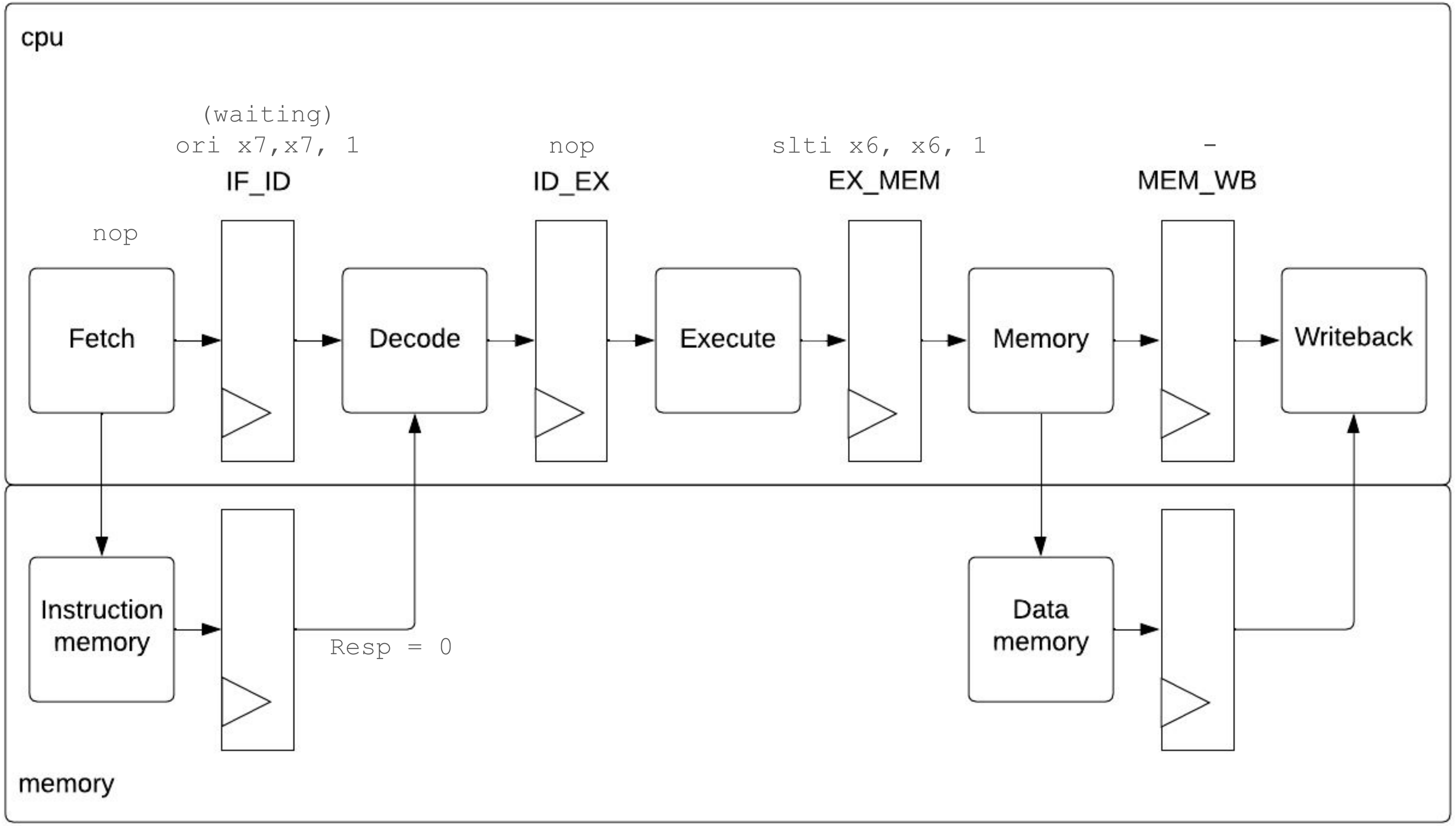
Instruction stall illustration: Cycle 1



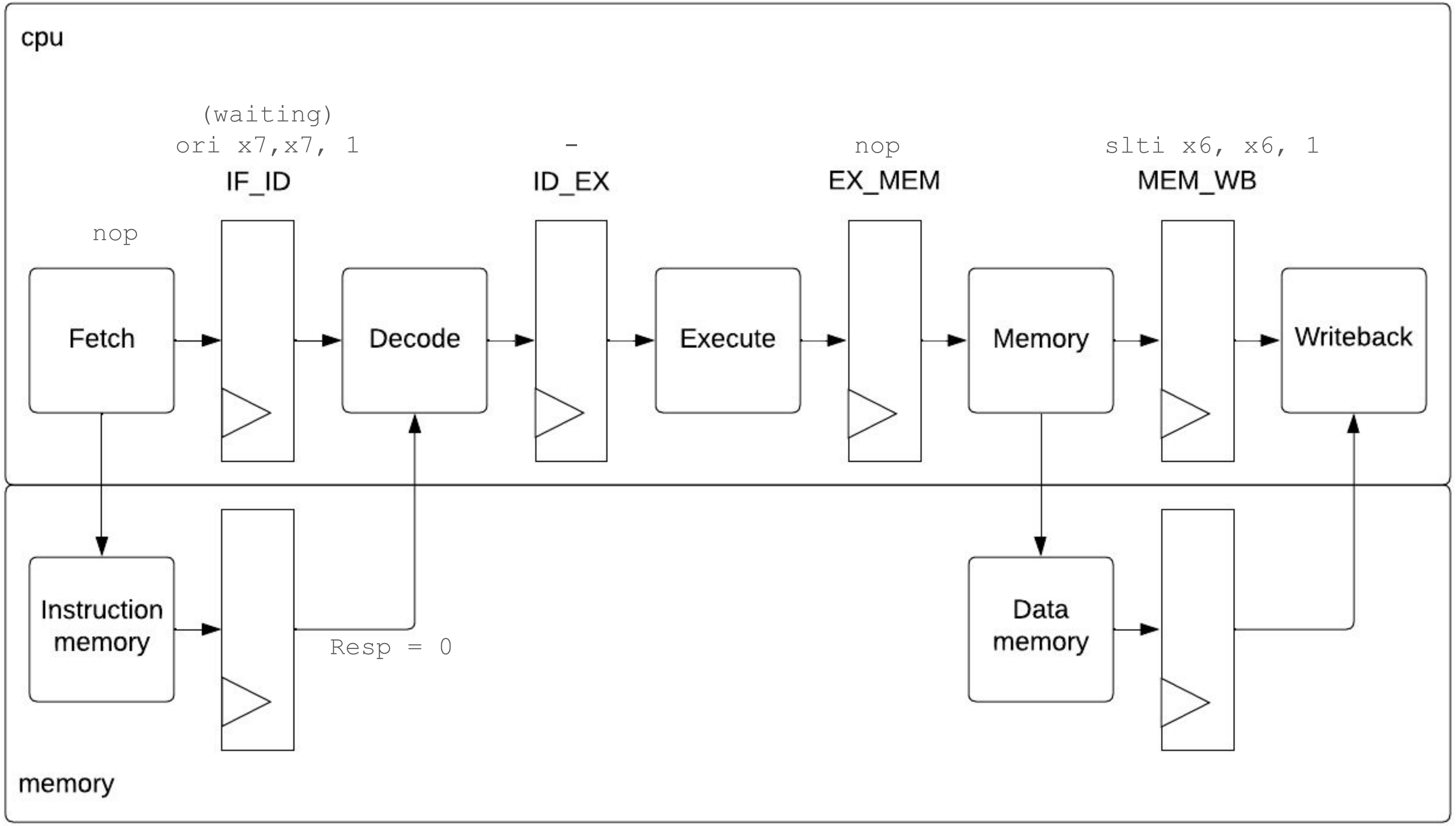
Instruction stall illustration: Cycle 2



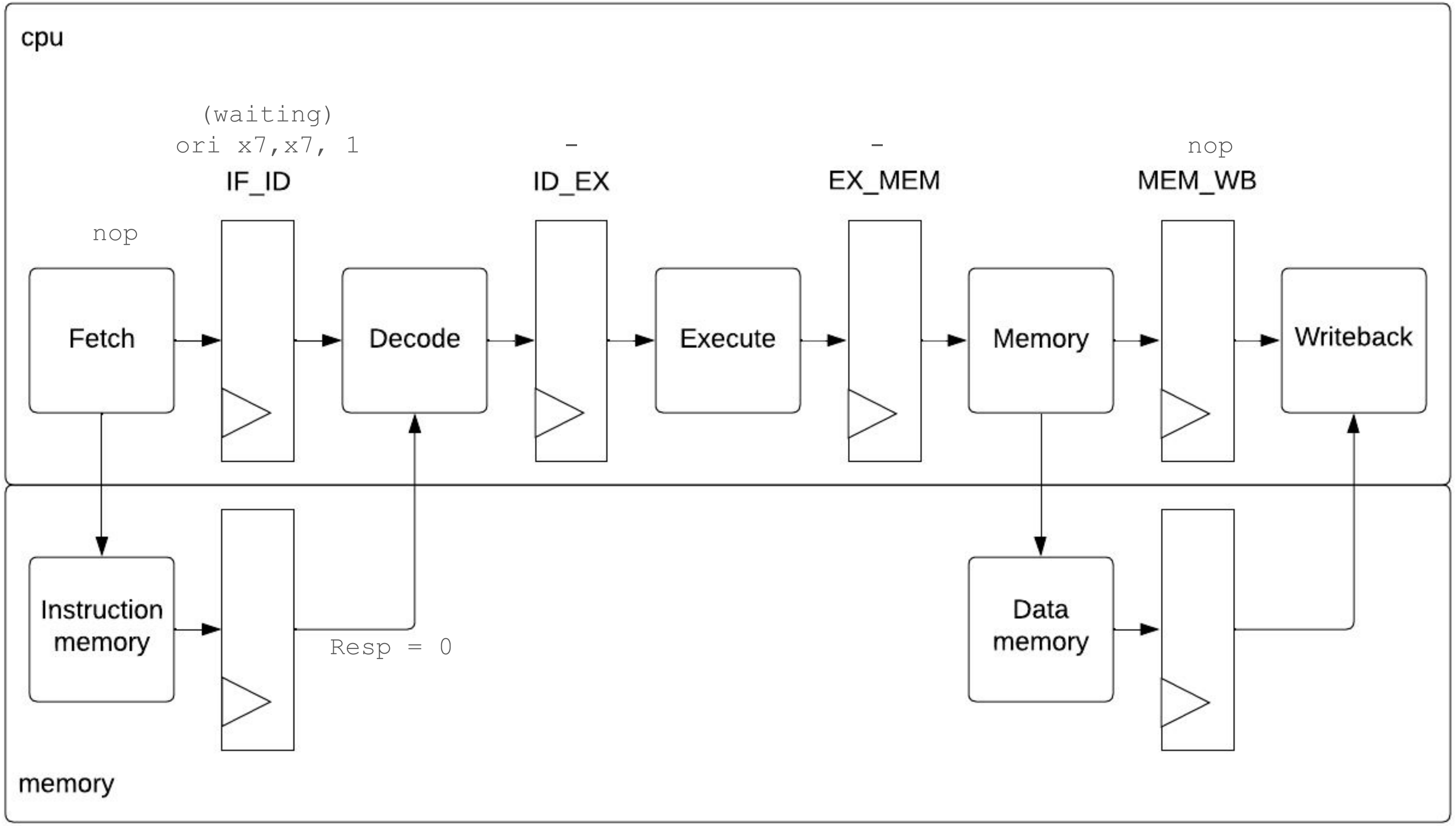
Instruction stall illustration: Cycle 3



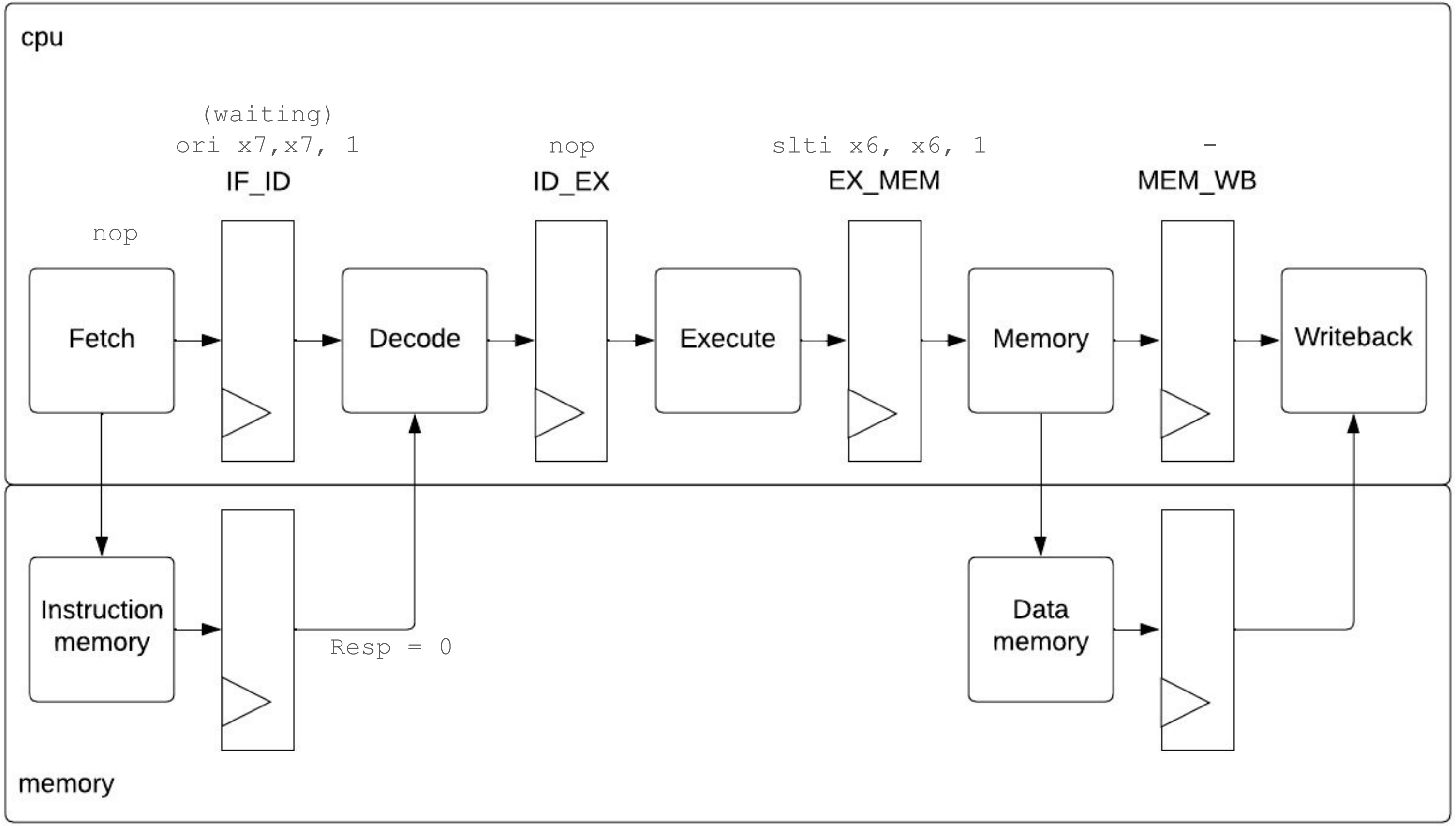
Instruction stall illustration: Cycle 4 (with backpressure)



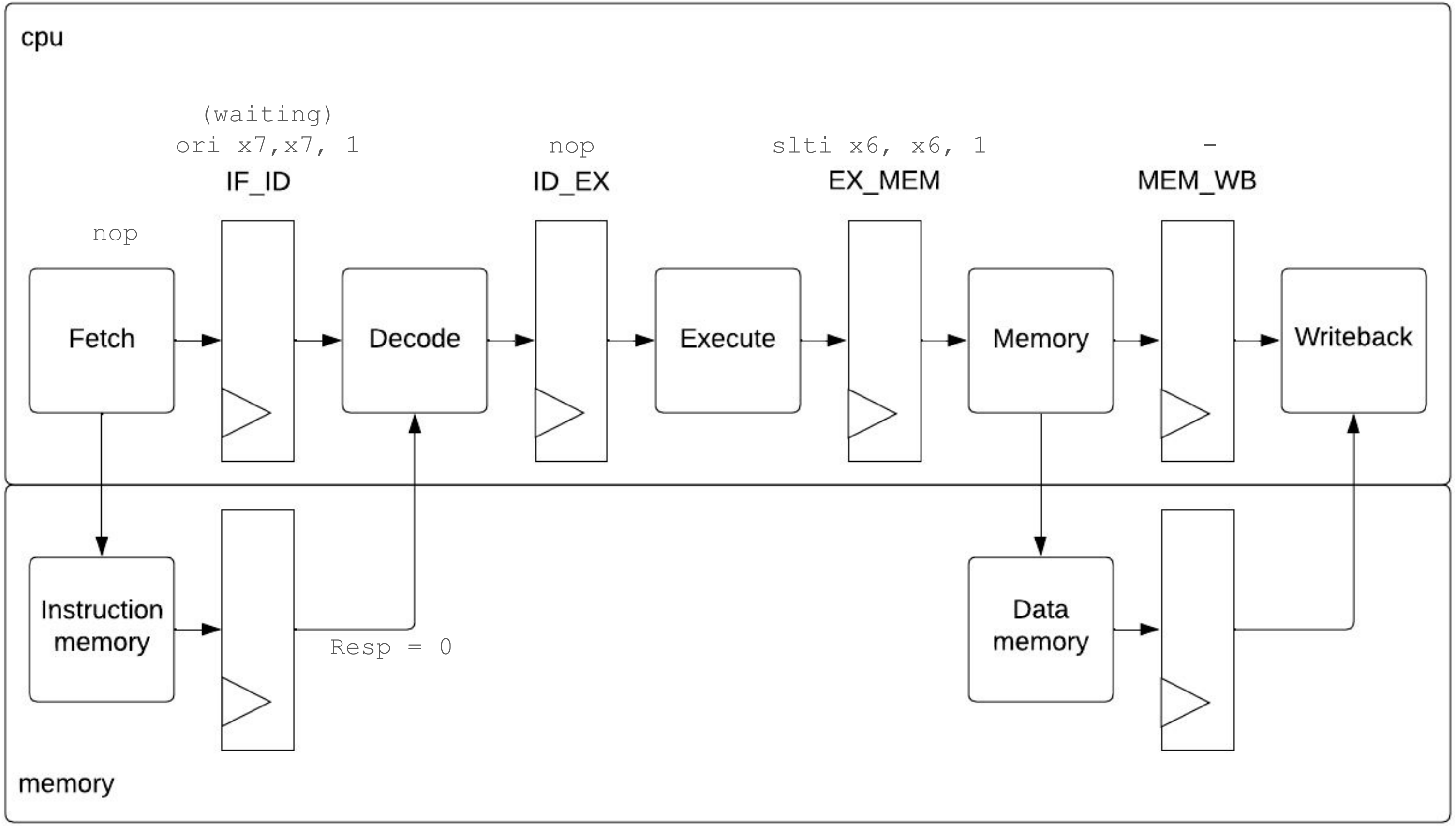
Instruction stall illustration: Cycle 5 (with backpressure)



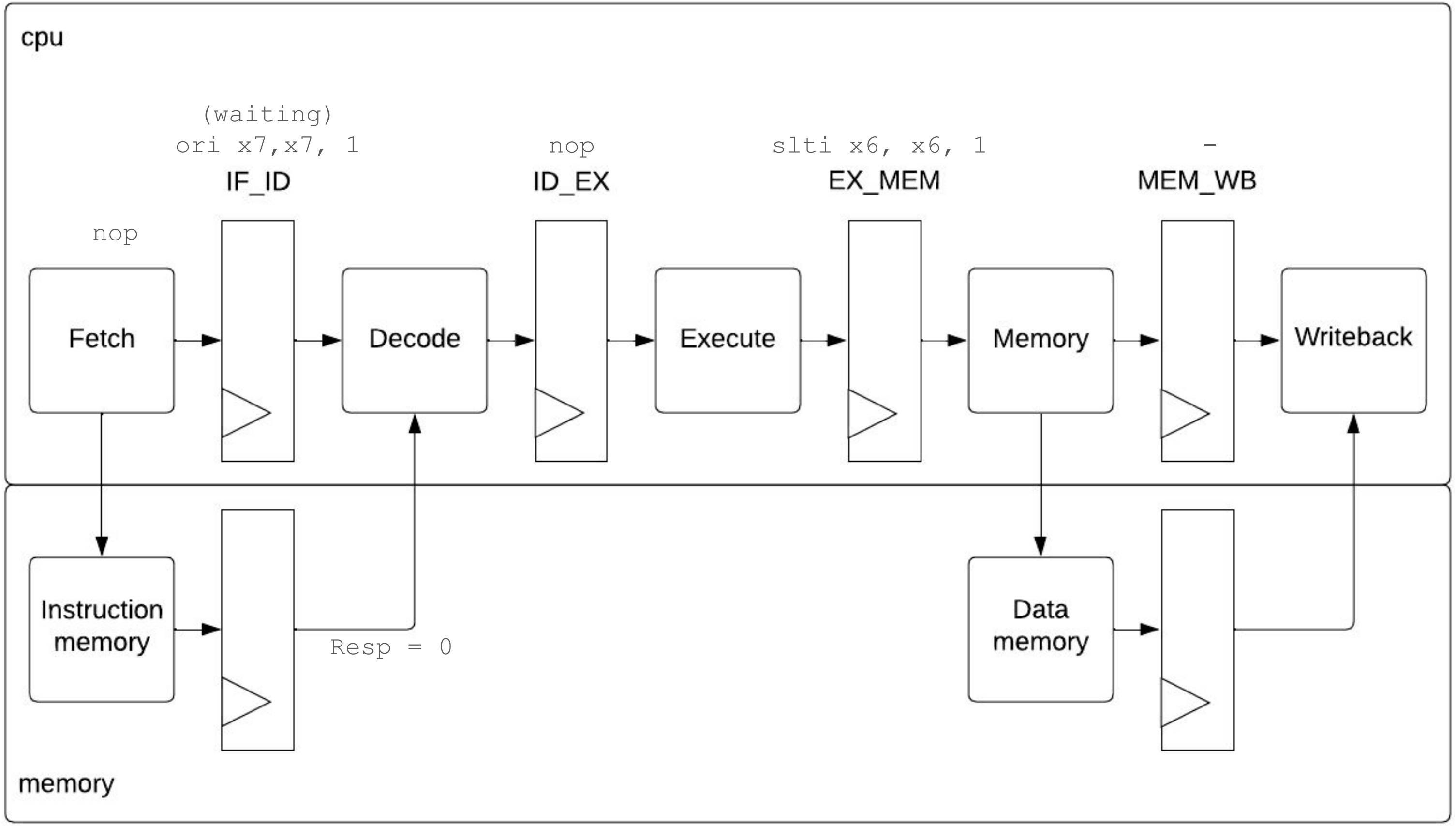
Instruction stall illustration: Cycle 3



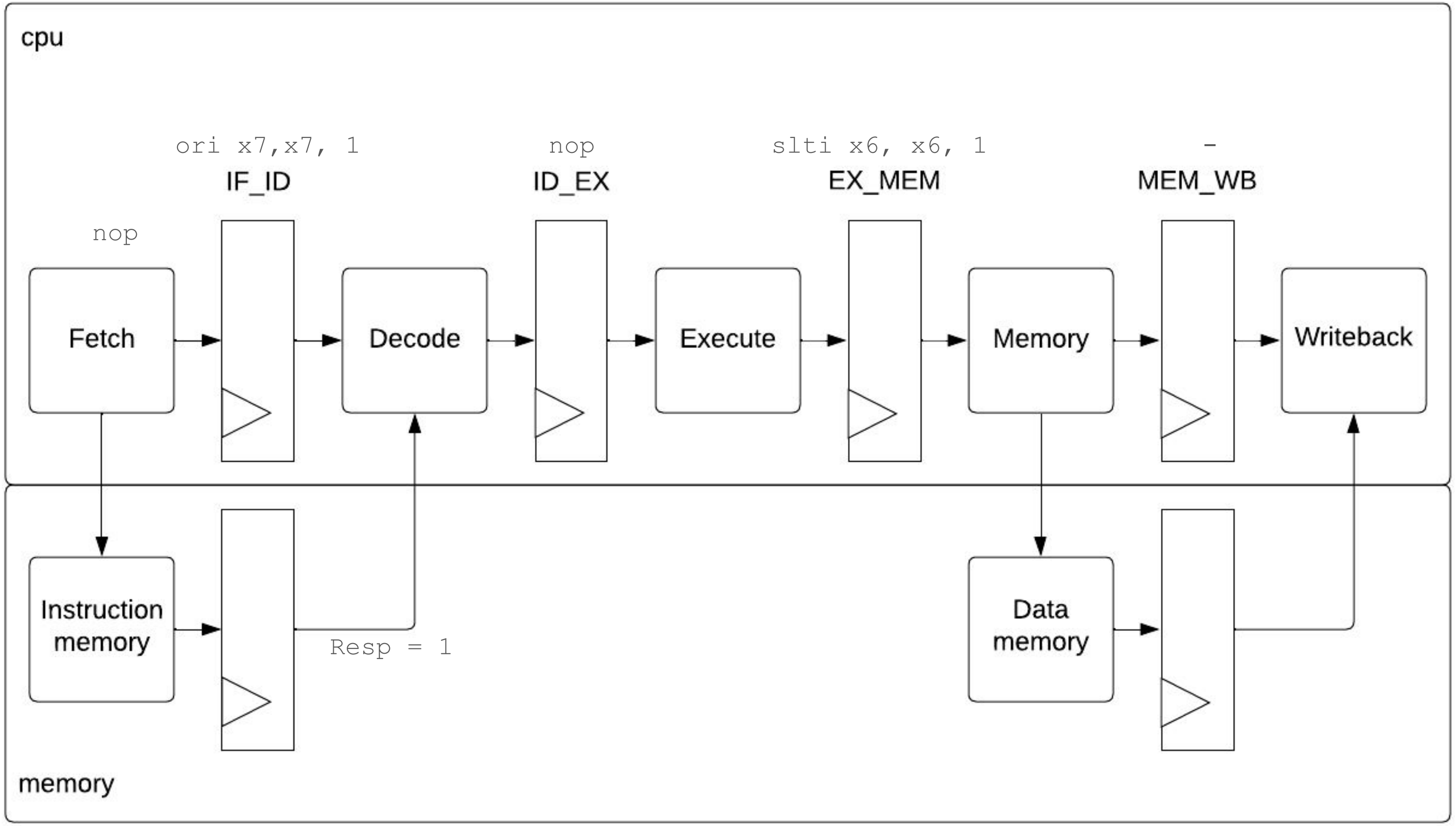
Instruction stall illustration: Cycle 4 (full stall)



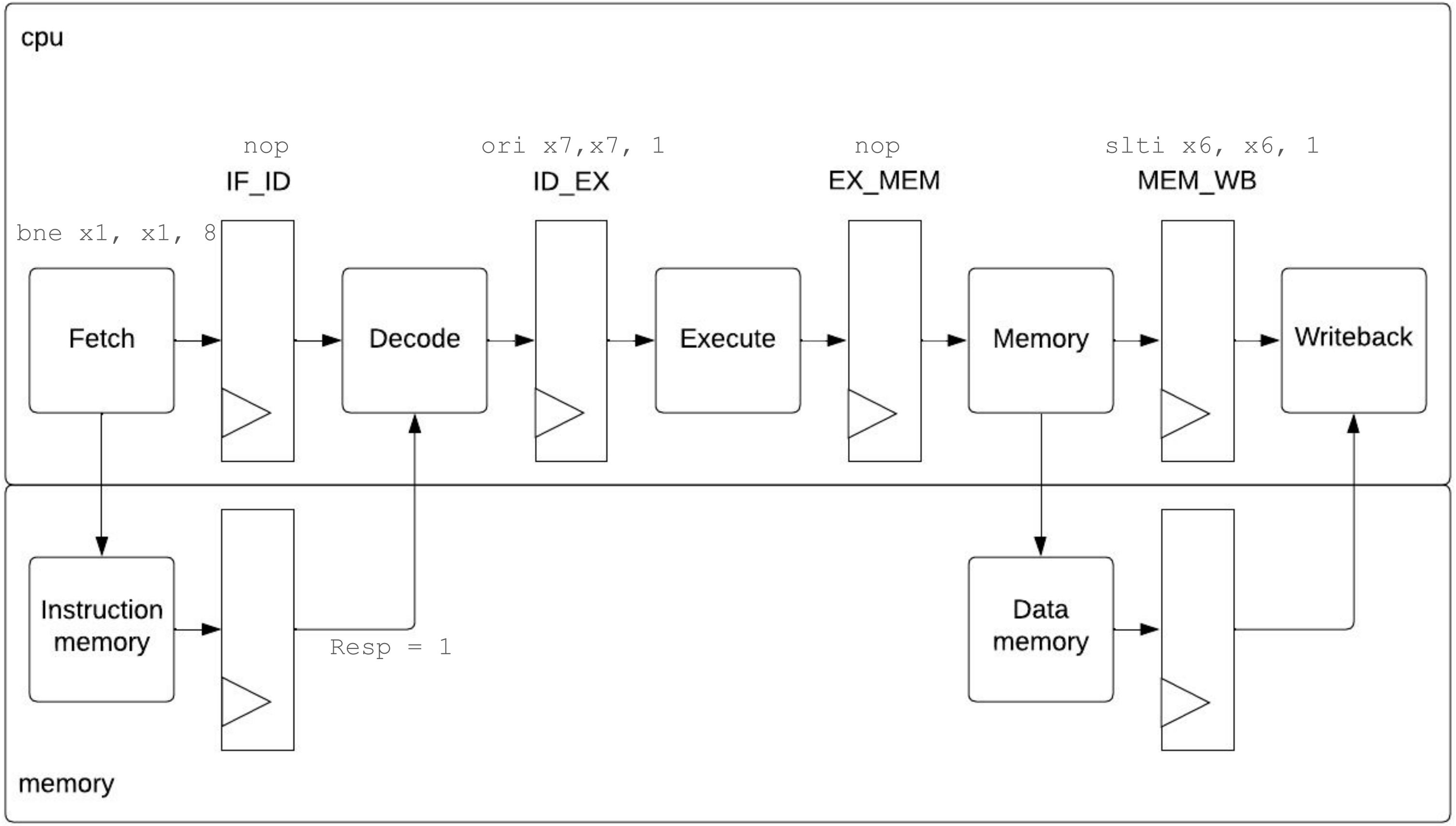
Instruction stall illustration: Cycle 10 (full stall)



Instruction stall illustration: Cycle 11 (full stall)



Instruction stall illustration: Cycle 12 (full stall)



Edge cases

- Both are stalling?
  - Just stall the whole thing like normal
- Both stalling but imem responds first?
  - Keep stalling, need to wait dmem anyways
- Both stalling but dmem responds first?
  - Keep stalling, need to wait for imem anyways
    - If using backpressure, can start letting everything but fetch/decode move forward
- Neither is stalling?
  - Done in CP2

Testing

- ordinary_dual_port.sv mimics cache!
- Write asm code with several instructions, to strain icache
    - Write small loops to see no-stalling behavior
    - Bigger loop can have mixed stalling/no stalling
- Write instructions with several loads and stores to strain dcache
    - Make sure loads/stores are to valid addresses
    - Test load/store to the same address consecutively - shouldn’t stall
- Targeted testing is your friend
- Use rand_tb to ensure all possible cases work

Questions?