Lecture 17: Pipeline Hazards

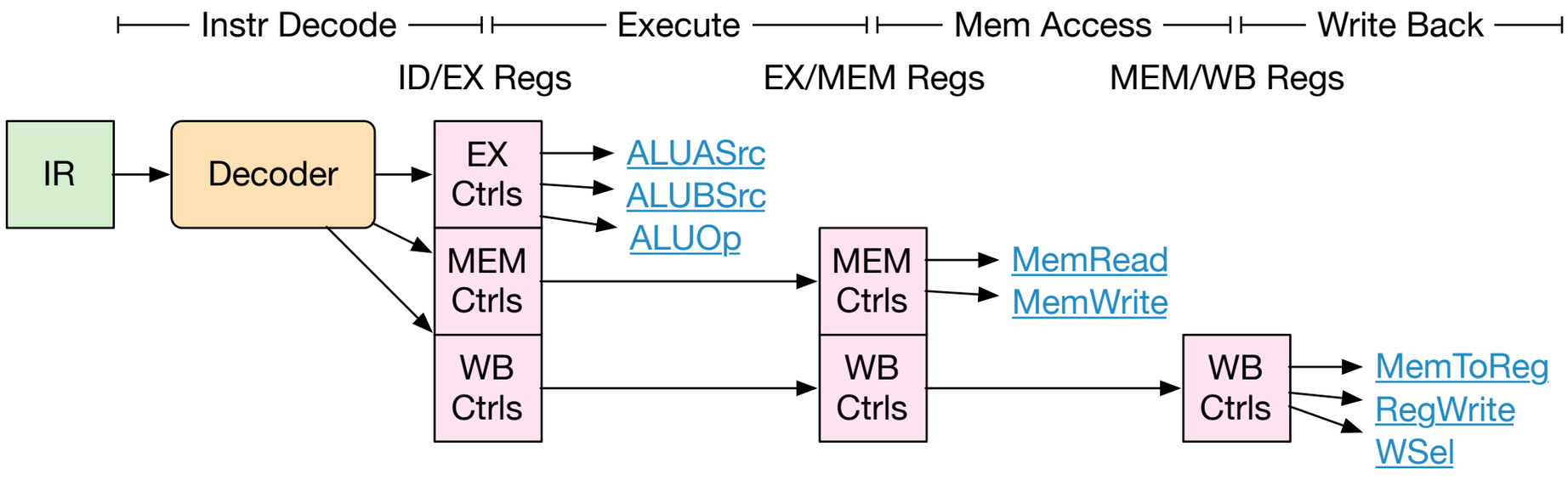
Spring 2024
Jason Tang

Topics

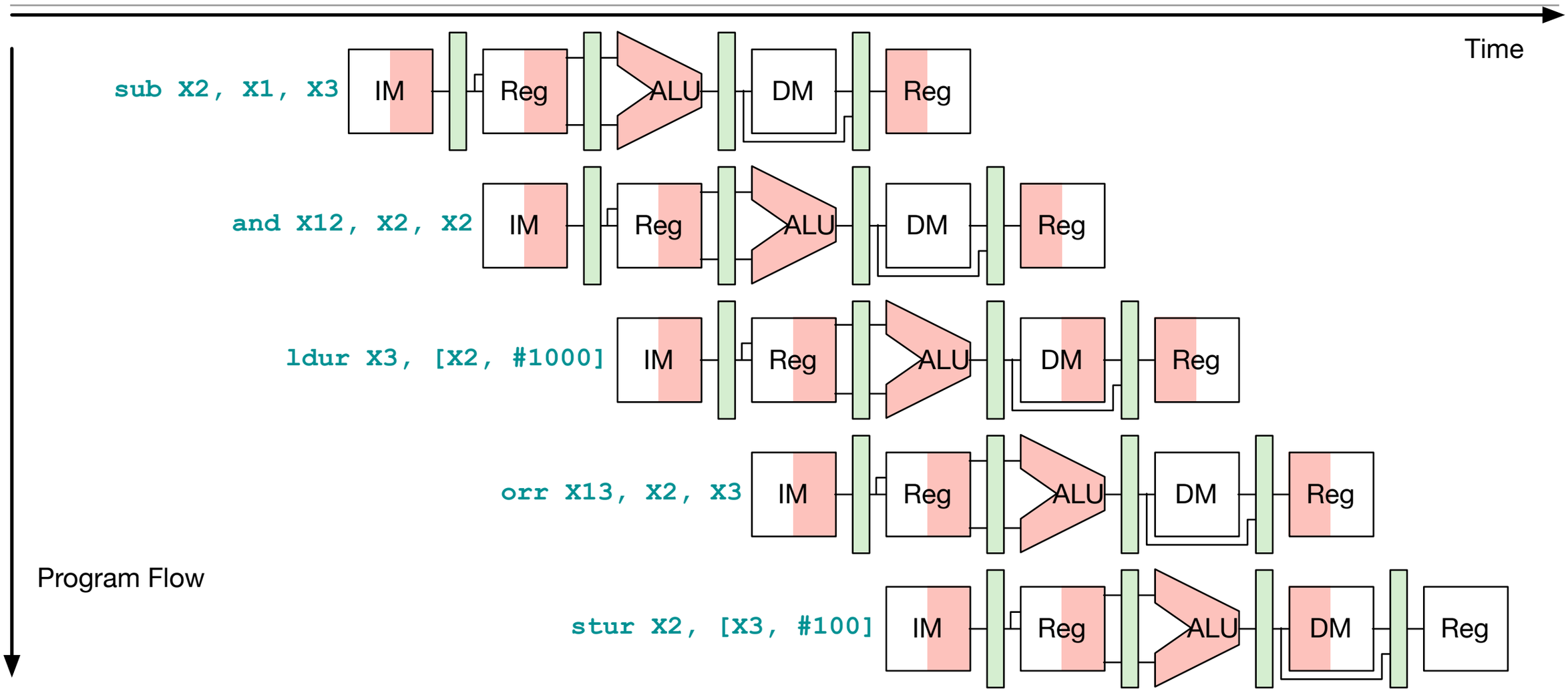
- Data forwarding
- Pipeline stalls
- Branch predictors
- Exception handling

Data Stationary Control

- Main instruction decoder generates control signals during Instruction Decode / Register File Read (ID) cycle
  - Control signals for Execute / Address Calculation (EX) are used 1 cycle later
  - Control signals for Memory Access (MEM) are used 2 cycles later
  - Control signals for Write Back (WB) are used 3 cycles later



Data Hazard Example



- Caused by backward dependency at ID stage

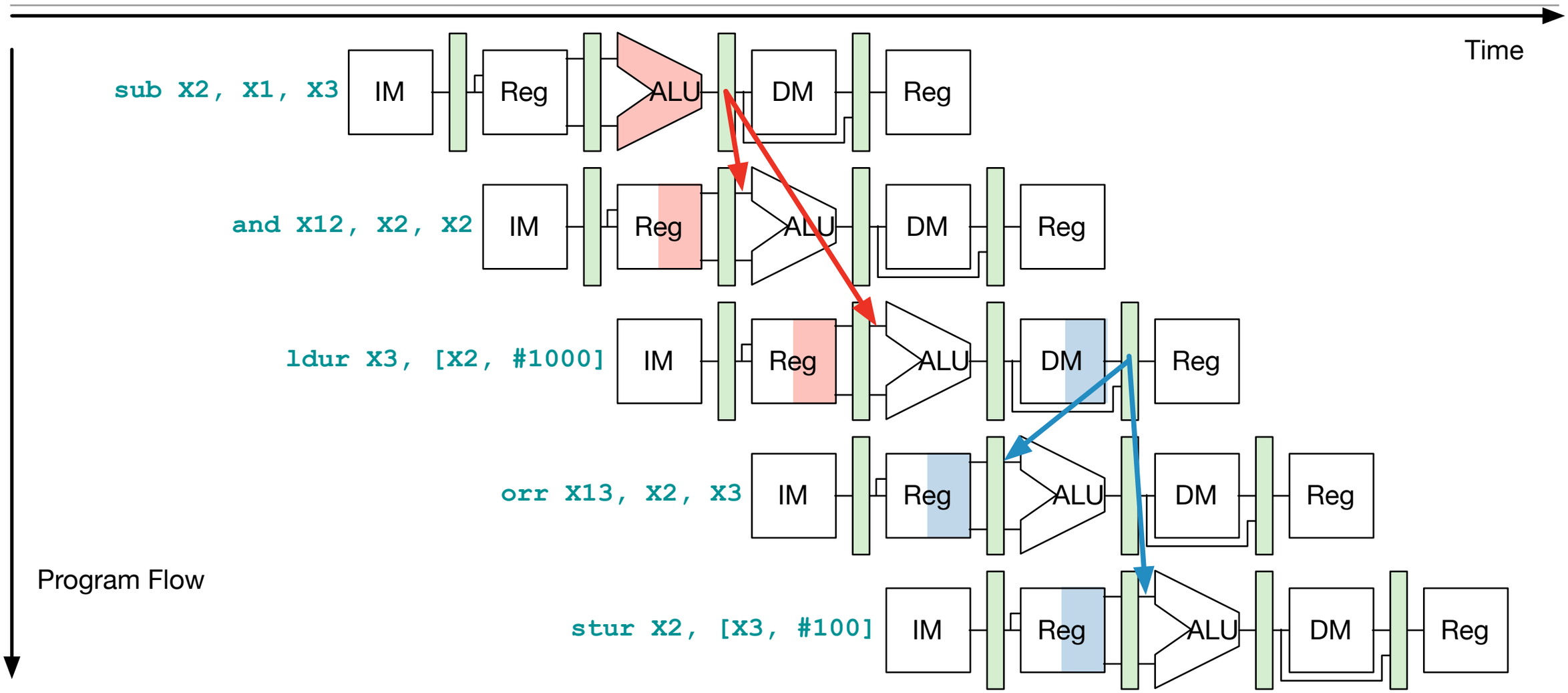
- Assume writes occur on first half of clock cycle, then reads on second half

Data Availability

Stage Numbers, as "Data Available Normally / Earliest"			Write to Register File	
			R-Type	Load
			6/4	6/5
Read from Register File	R-Type	2/3	4/1	4/2
	Load	2/3	4/1	4/2
	Store (addr)	2/3	4/1	4/2
	Store (val)	2/4	4/1	4/1

- Let stages be numbered, where "stage 6" is after the instruction exits pipeline
- This table describes a register dependency as a stage number pair
- Difference of cycle numbers = minimum spacing between instructions (as measured in cycles)

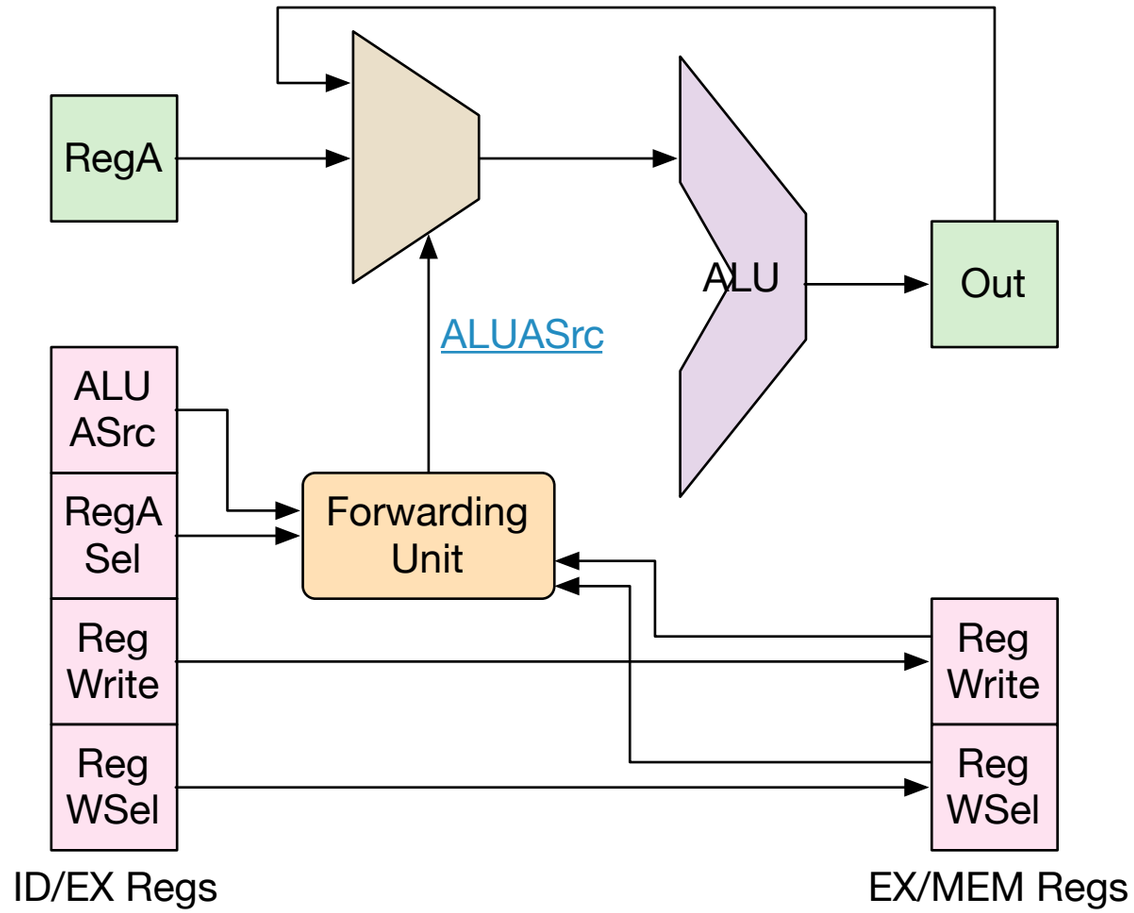
Data Hazard Types



- "Type 1": EX/MEM.RegWSel → ID/EX.RegASel [or ID/Ex.RegBSel]
- "Type 2": MEM/WB.RegWSel → ID/EX.RegASel [or ID/Ex.RegBSel]
- No hazard after WB to ID/EX.RegASel (or BSel), because writes occur before reads

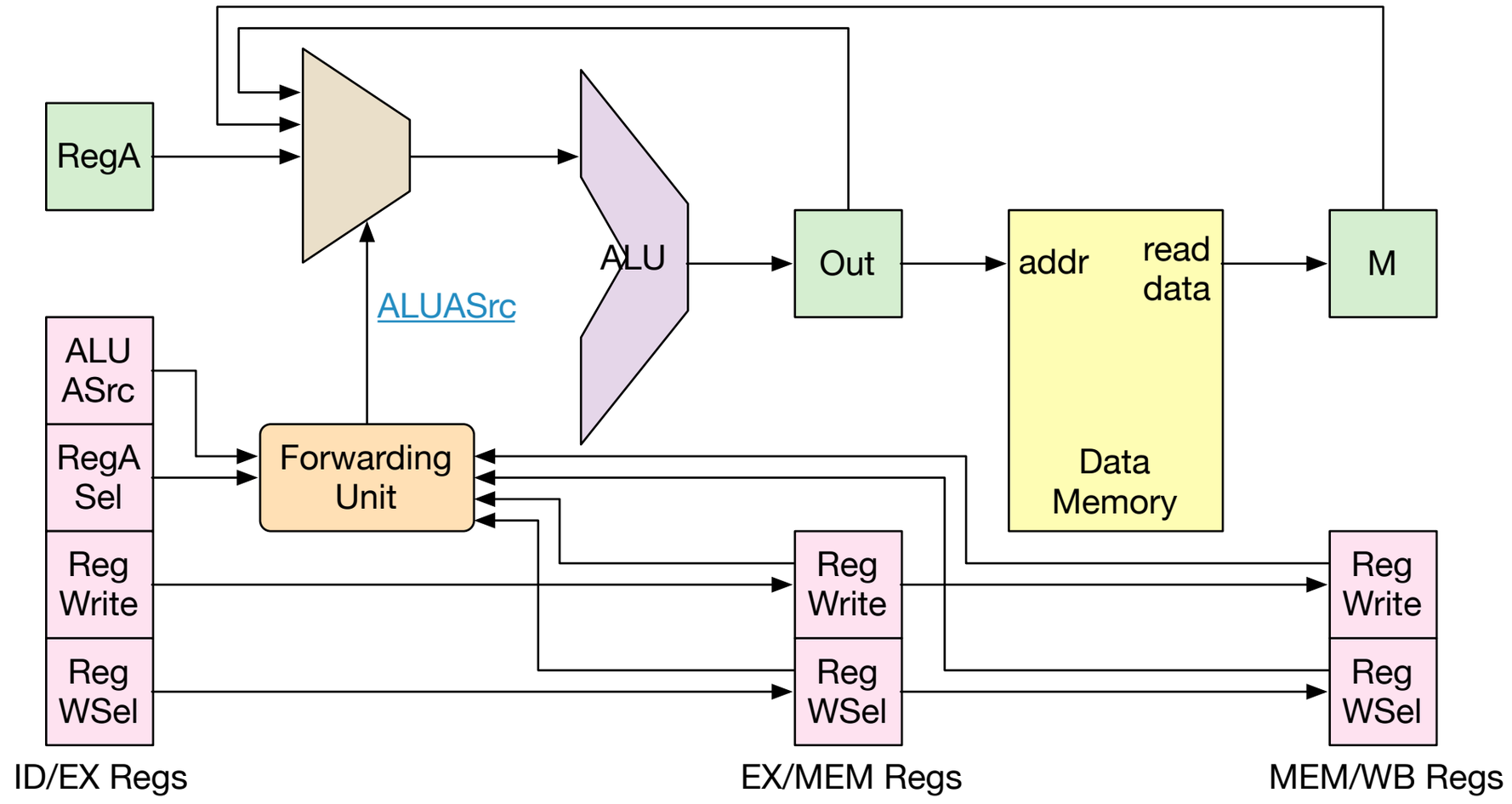
Data Forwarding

- Only matters when an earlier stage is modifying a register value
- Add a Forwarding Unit that uses data stationary registers to select ALU input
- Two ways for Forwarding Unit to detect, depending upon data hazard type



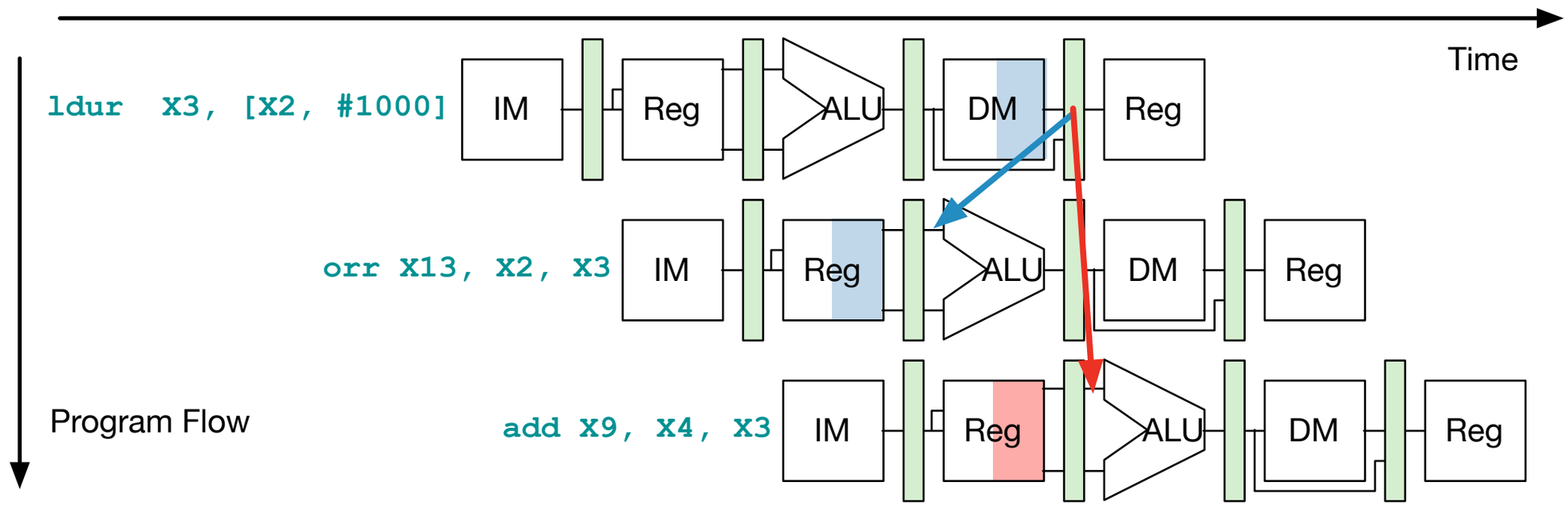
  - Type 1: EX/MEM.RegWrite *and* (EX/MEM.RegWSel = ID/EX.RegASel [or BSel])
  - Type 2: MEM/WB.RegWrite *and* (MEM/WB.RegWSel = ID/EX.RegASel [or BSel])

Data Forwarding



- Extend Forwarding Unit to take into account MEM/WB registers
  - EX/MEM registers take precedence over MEM/WB
- Use similar idea to generate ALUBSrc control

Data Hazard Caused by Load



- Data forwarding only works when minimum spacing between instructions is 1 (arrow points towards right)

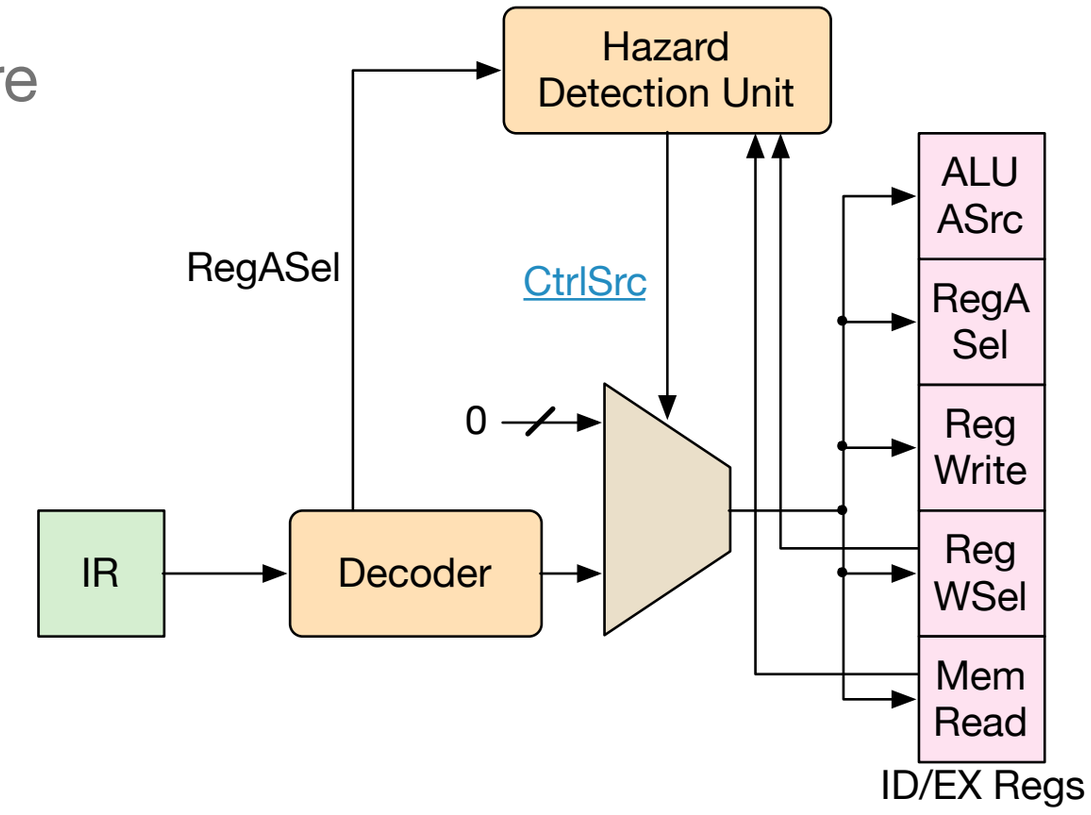
- When spacing is 2 (or greater, arrow points left), must resolve by stalling pipeline

    - Be sure to stall only instruction(s) earlier in pipeline; ensure that load instruction continues executing

Hazard Detection Unit

- If a data stationary register's controls are all zero, then no work will be performed on that part of the CPU for that cycle (a no-op)

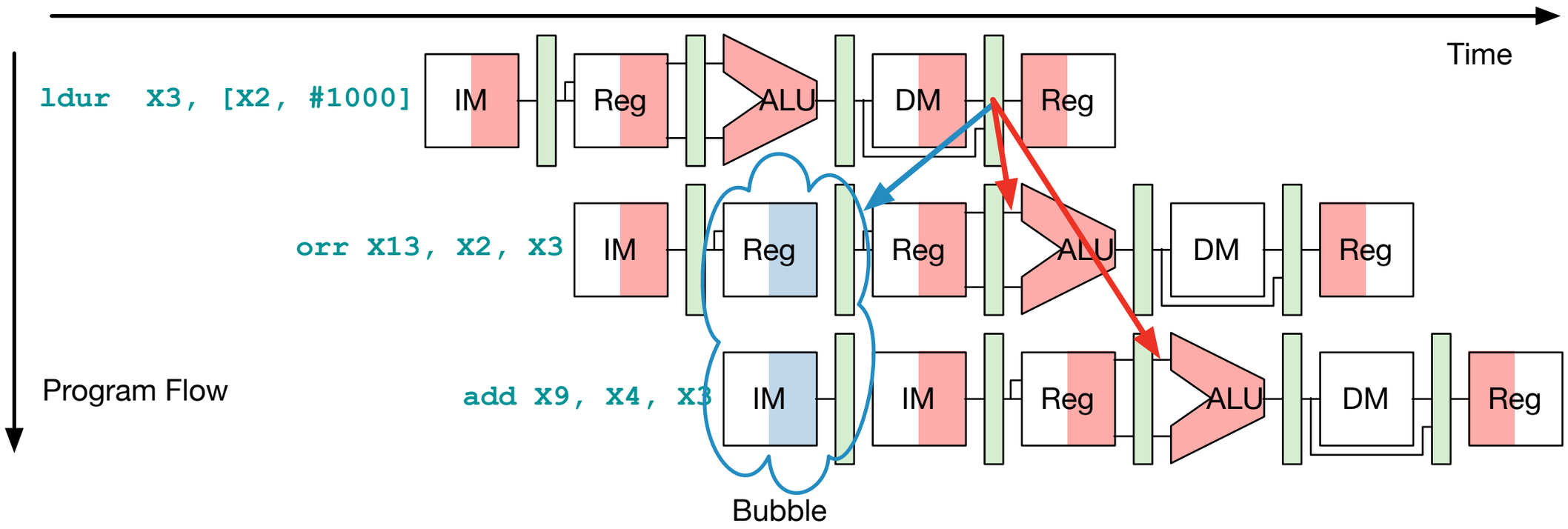
- Add a Hazard Detection Unit that uses data stationary registers to insert bubbles into pipeline



- Hazard Detection Unit stalls pipeline when it detects a load-use data hazard:

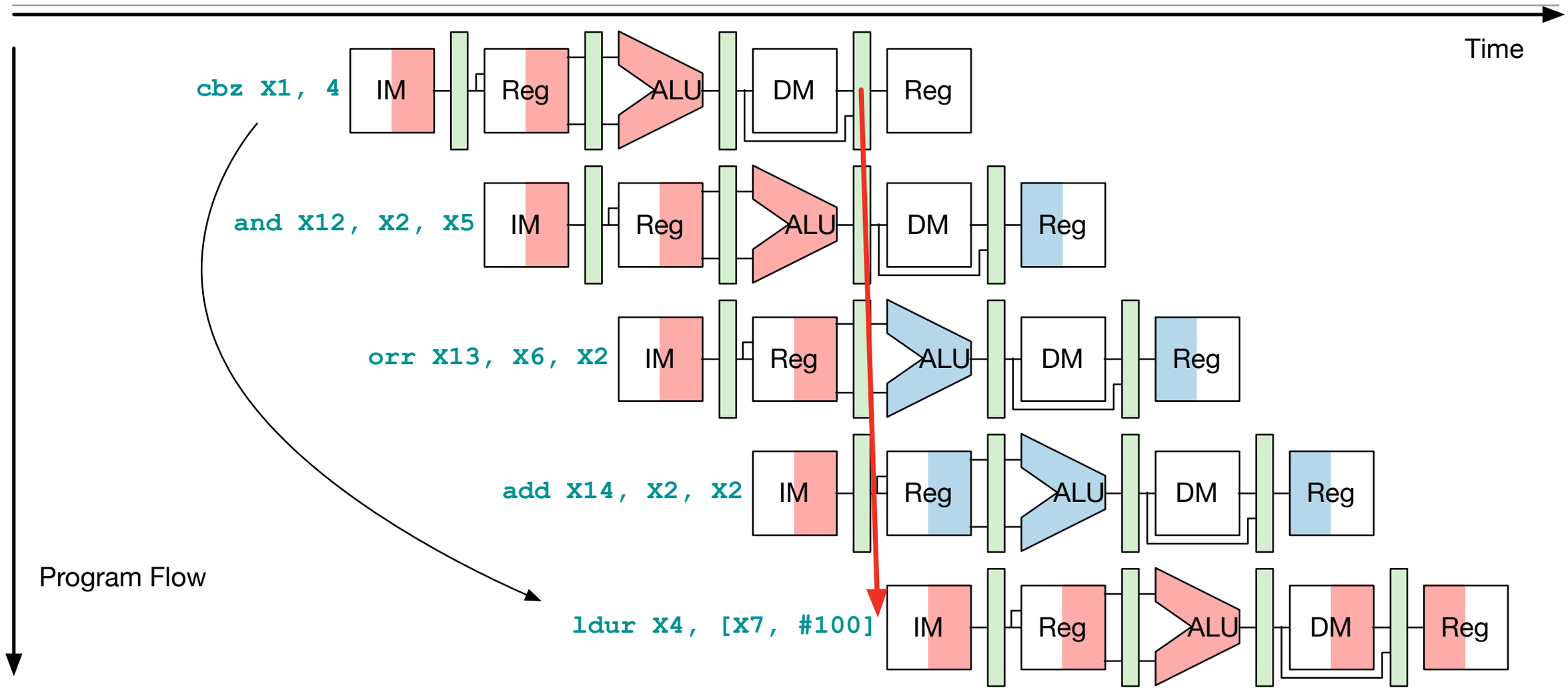
  - ID/EX.MemRead *and* (ID/EX.RegWSel = IF/ID.RegASel [or BSel])

Stalled Pipeline



- When Hazard Detection Unit sets a pipeline stage's data stationary registers to zero, that introduces a bubble into pipeline
  - Bubble delays execution by one cycle; intermediate results are ignored
  - Thus that stage's operations are repeated

Control Hazard Example



- Pipeline continues fetching instructions, while branch instruction resolves

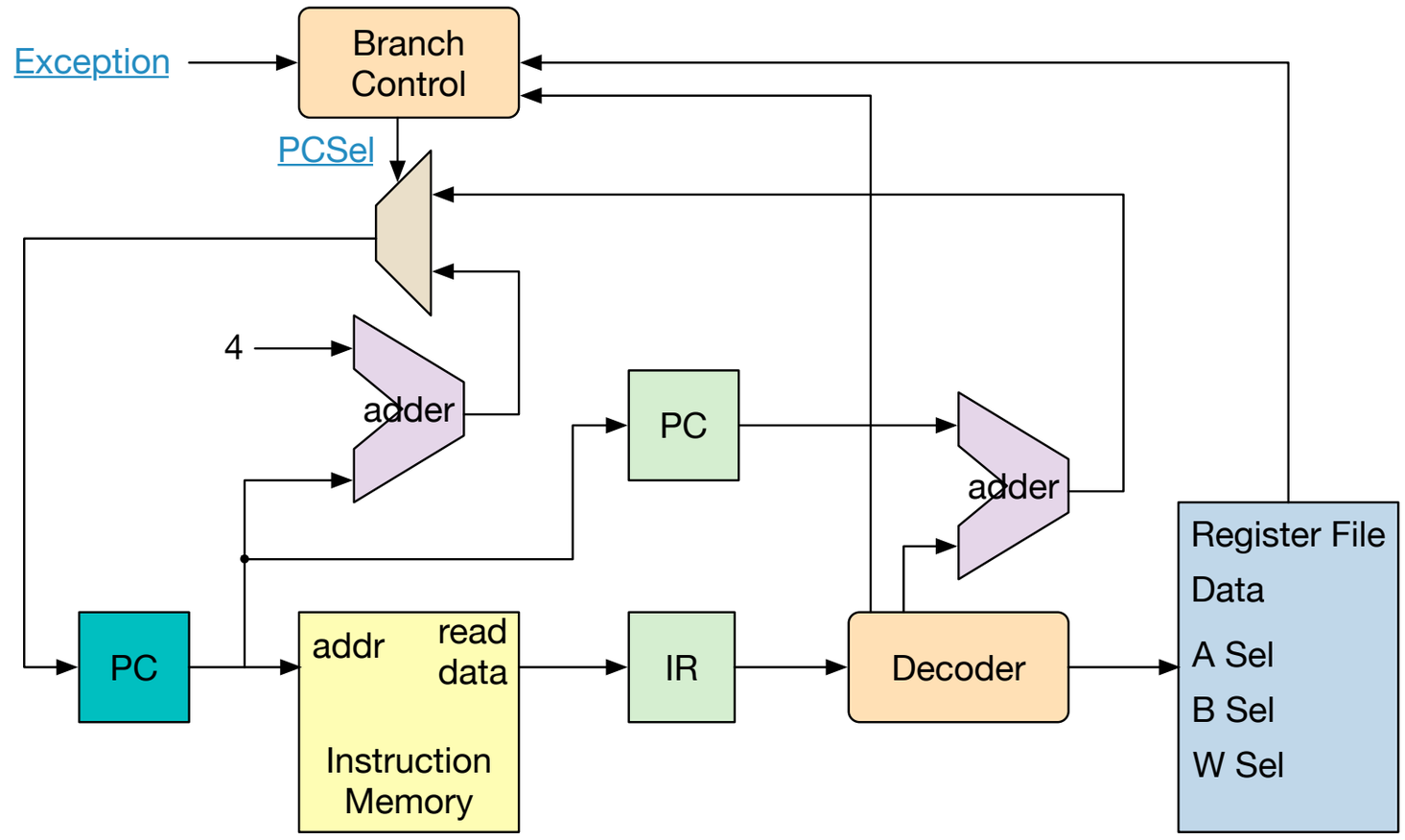
- Can be solved via delayed branching and branch prediction

Branch Flushing

- In simplest design, conditional branching is resolved during MEM stage
- Stalling entire pipeline while resolving a branch is too slow
- Instead, one approach is to assume that branch will not be taken
  - If branch is taken, then flush entire pipeline and restart processing
- For very long pipelines, pipeline flush is very expensive
  - ARM Cortex-A53 has a pipeline length of 8; Intel Coffee Lake pipeline has at least 14 stages

Reducing Branch Delay

- Could move decision earlier, to ID stage
  - Calculate target address with a second adder
  - Reduce penalty of pipeline flush to 1 cycle



- Introduces a data hazard problem, if the branch condition or branch target address depends upon earlier instruction
  - Resolved via data forwarding, or a stall if waiting upon a load result

Delay Slot

- If a branch penalty is one cycle, then a common hardware trick is to rely upon a delay slot

  - While branch instruction is resolving, the compiler reorders instructions to place an instruction *immediately* after branch that *does not* depend upon the result of the branch

- If compiler cannot find a usable instruction, then it inserts a **`nop`**

```
add X1, X2, X3
cbz X2, 10
```

```
cbz X2, 10
add X1, X2, X3
```

delay slot

Static Branch Prediction

- Different ways to improve branch prediction
- On PowerPC, make prediction based upon direction of branch and hint bit:
  - If direction is negative, then assume branch will be taken; otherwise assume branch is not taken
- Hint bit within instruction reverses logic

```
const char *home_dir;

home_dir = getenv("HOME");
if (likely(home_dir))
    printf("home directory: %s\n", home_dir);
else
    perror("getenv");
```

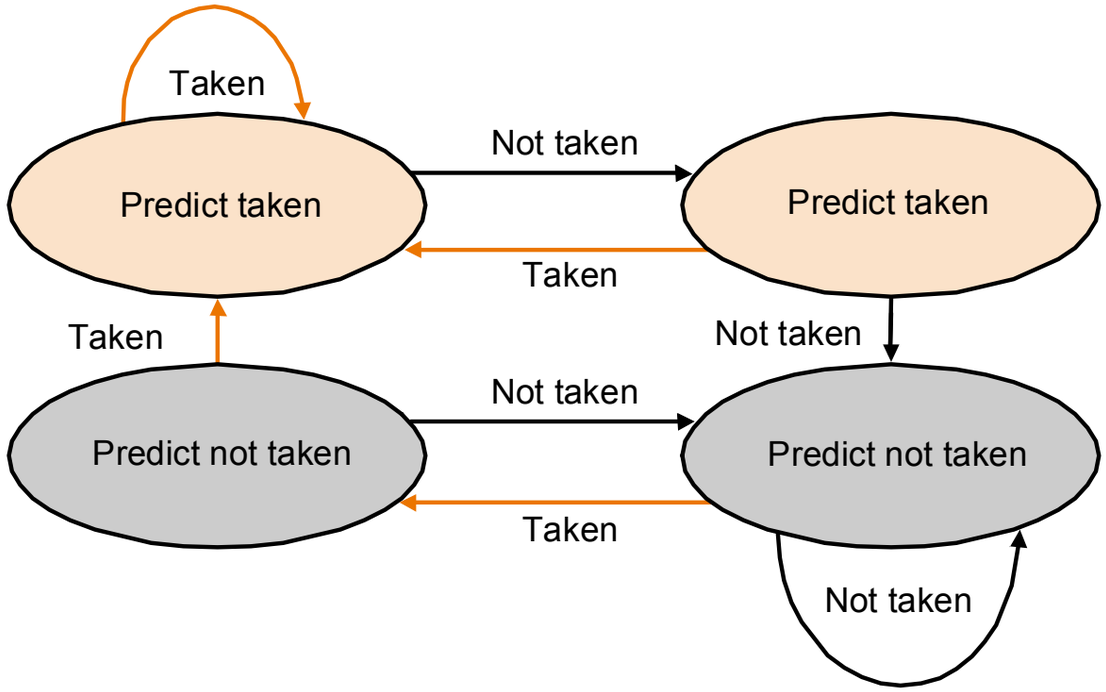
http://www.mactech.com/articles/develop/issue_20/20balance.html;
https://www.geeksforgeeks.org/branch-prediction-macros-in-gcc/

Dynamic Branch Prediction

- Record history of branch resolutions, indexed by instruction address
- Simplest approach uses one-bit per branch instruction to record last result
  - Branch result different from prediction will flip history bit
- If the code is looping, this means the prediction fails at least once, at the end of the loop
  - If that same code loop is executed again, the first loop iteration will also fail

Dynamic Branch Prediction

- Improved branch predictor uses two-bit buffer to better capture history of the branch instruction



- Intel's Branch Prediction Unit (BPU) records at least the previous 16 branch results in a Branch Target Buffer (BTB)
  - Records patterns like T / T / T / N / T / T / T / N

https://software.intel.com/en-us/articles/branch-and-loop-reorganization-to-prevent-mispredicts

Exceptions in a Pipeline

- Exceptions are another type of control hazard, and resolved like a branch

- Many ways to detect an exception

Exception Type	Stage(s) when exception can occur
Invalid PC address for instruction fetch	IF
Undefined instruction	ID
Arithmetic exception	EX
Invalid memory access for data read	MEM
External hardware interrupt	any

Exception Handling

- When exception occurs, current PC is saved (on ARMv8-A, to ELR)
    - May also save other registers
- Allow pipeline to drain, by replacing instructions earlier in pipeline with no-ops
- Load PC with exception handling address and continue pipeline
    - When multiple exceptions occur, prioritize them to jump to most important exception first

Exception Handling Hardware