Branch Prediction

Philipp Koehn

30 March 2018

Control Hazard

- Also called branch hazard
- Selection of next instruction depends on outcome of previous
- Example

```
beq $s0, $s1, ff40
sub $t0, $s0, $t3
```

  - sub instruction only executed if branch condition fails

→ cannot start until branch condition result known

Methods

- Assume branch not taken
  - start execution of following instructions
  - if wrong, flush them

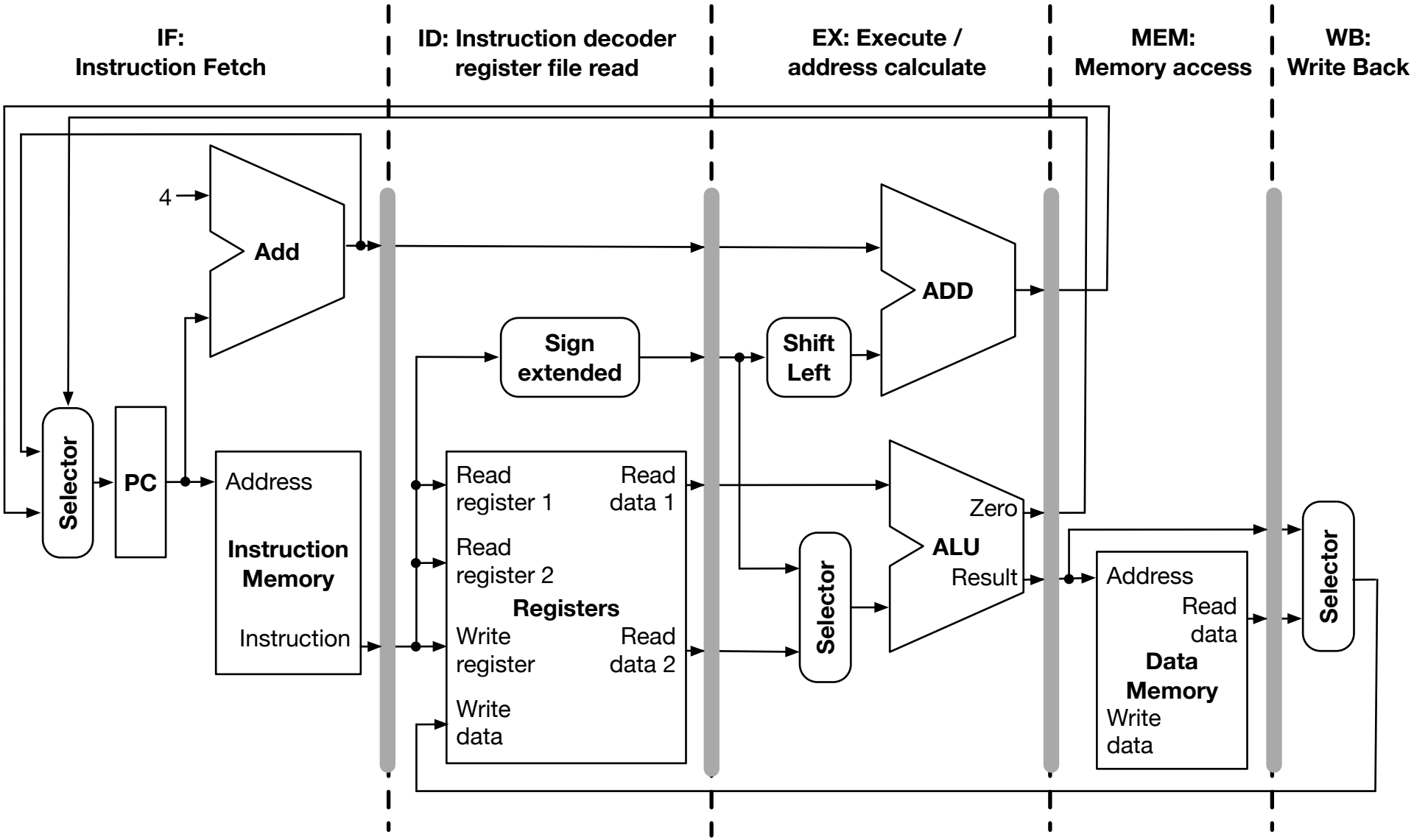
- Reduce delay of branches
  - compute branch address and condition in fewer cycles

  $\rightarrow$ less flushing

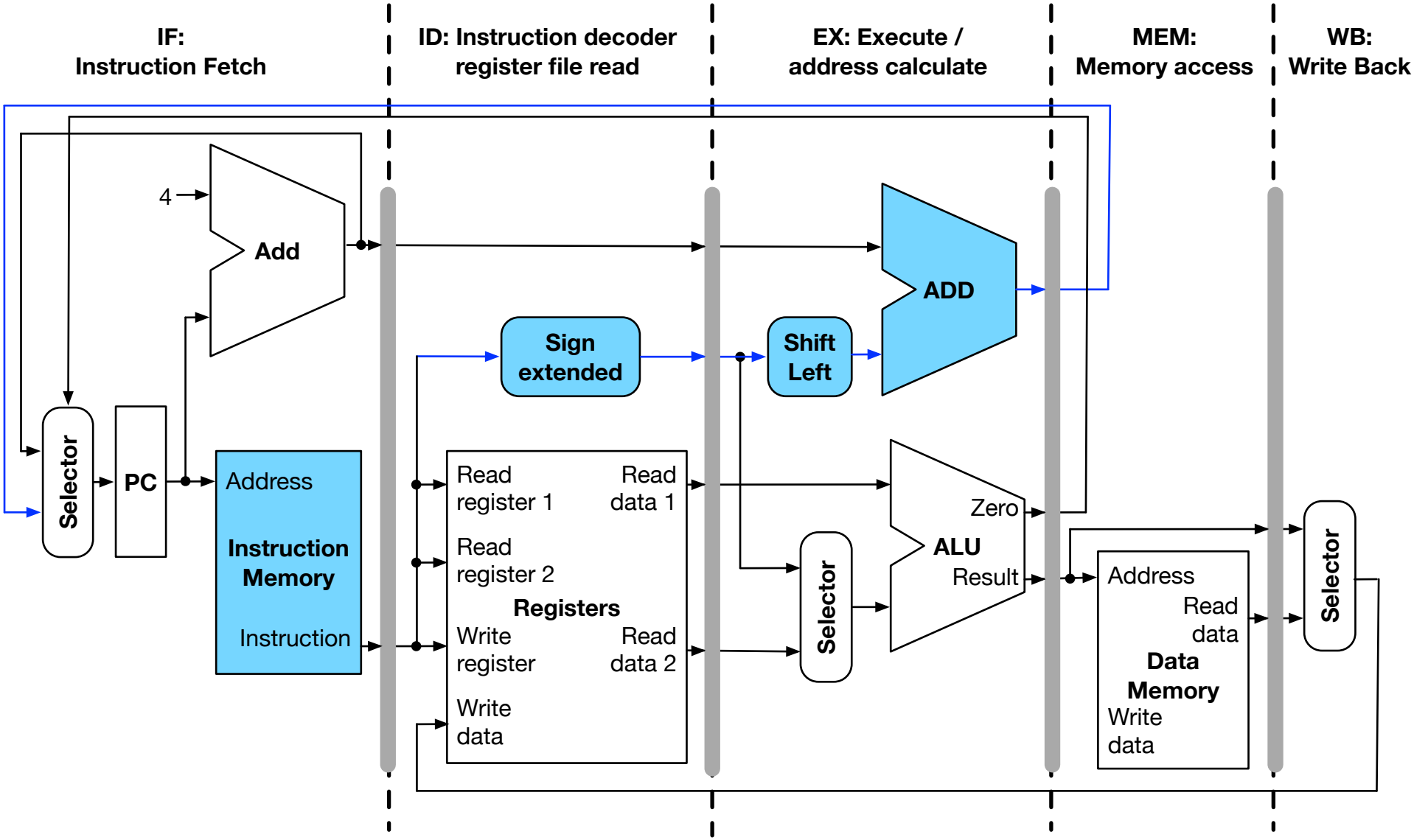
- Dynamic branch prediction

assume branch not taken

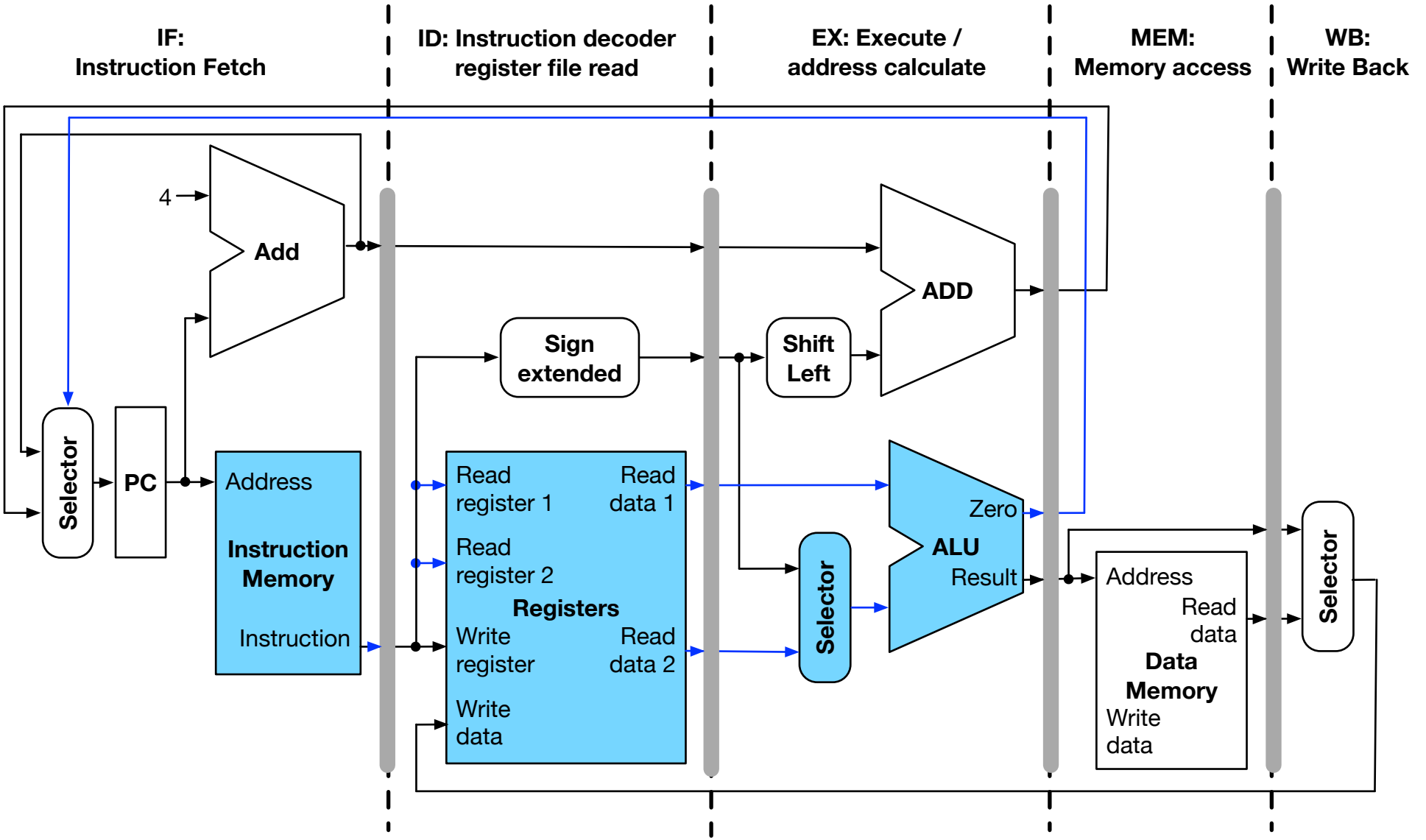
Pipelined Datapath



Branch: Address Calculation



Branch: Condition Checking

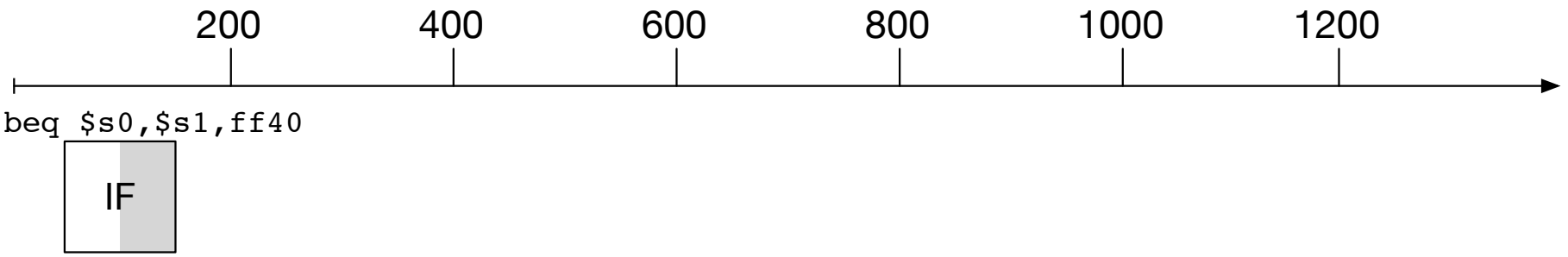


Idea

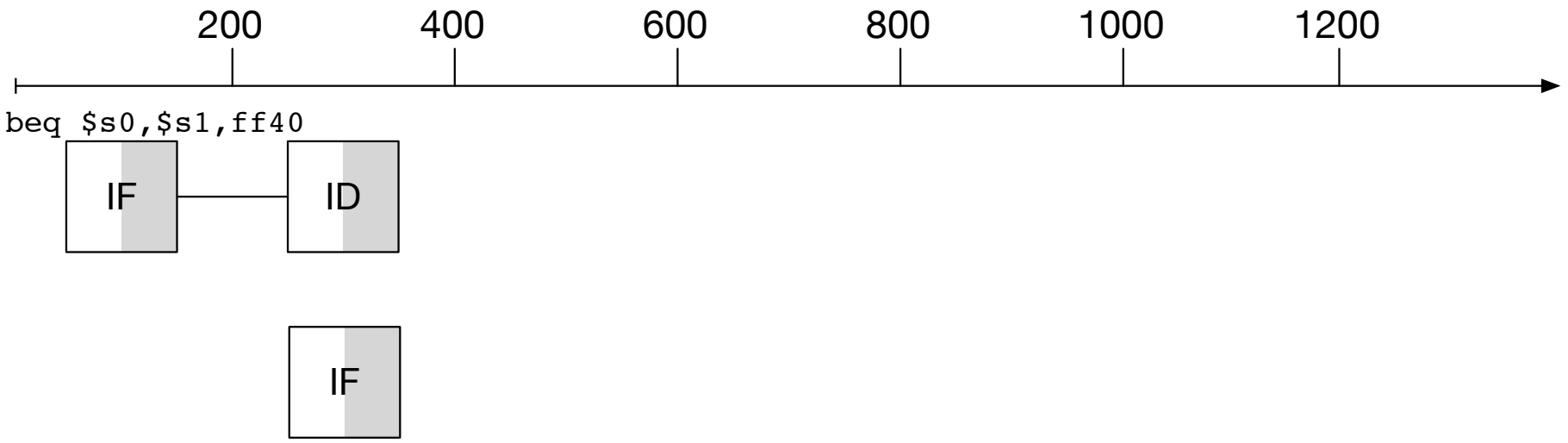
- Assume branch not taken
- Execute the subsequent instructions
- If branch should have been taken

  $\rightarrow$ flush out subsequent instruction processing

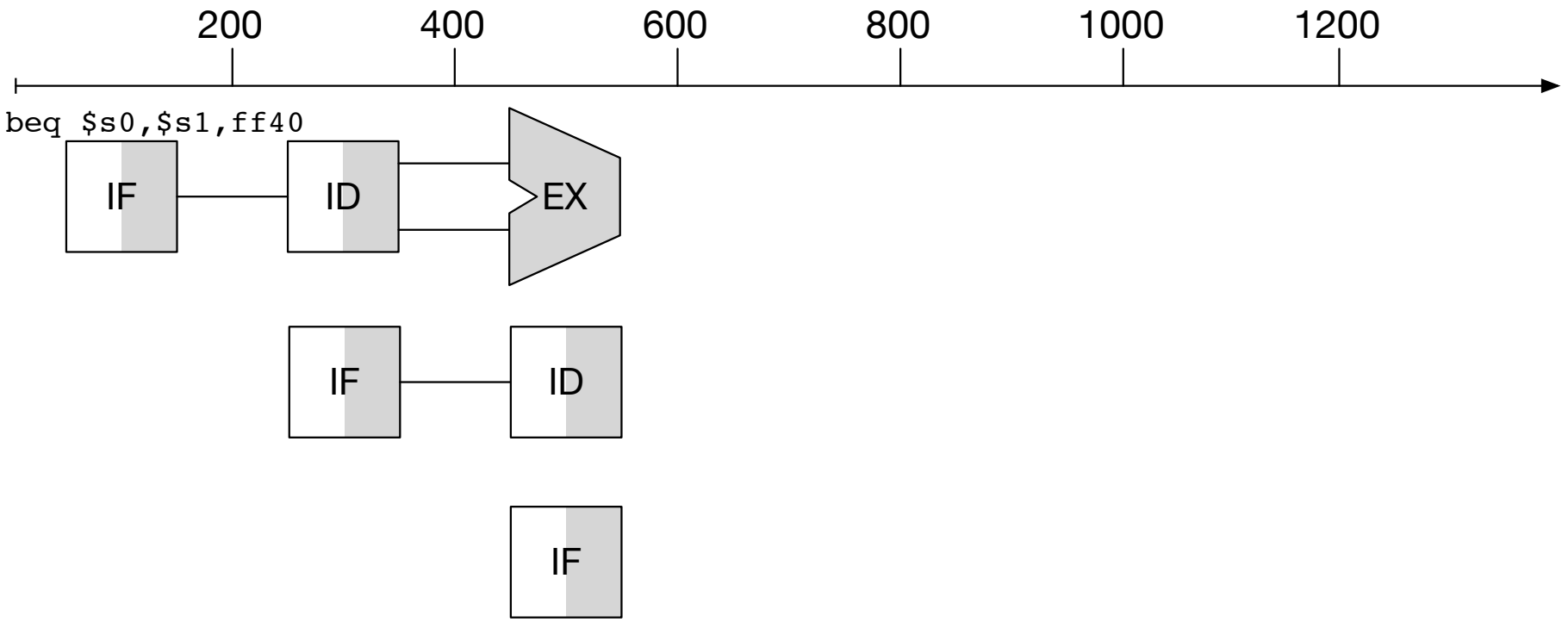
Branch Execution



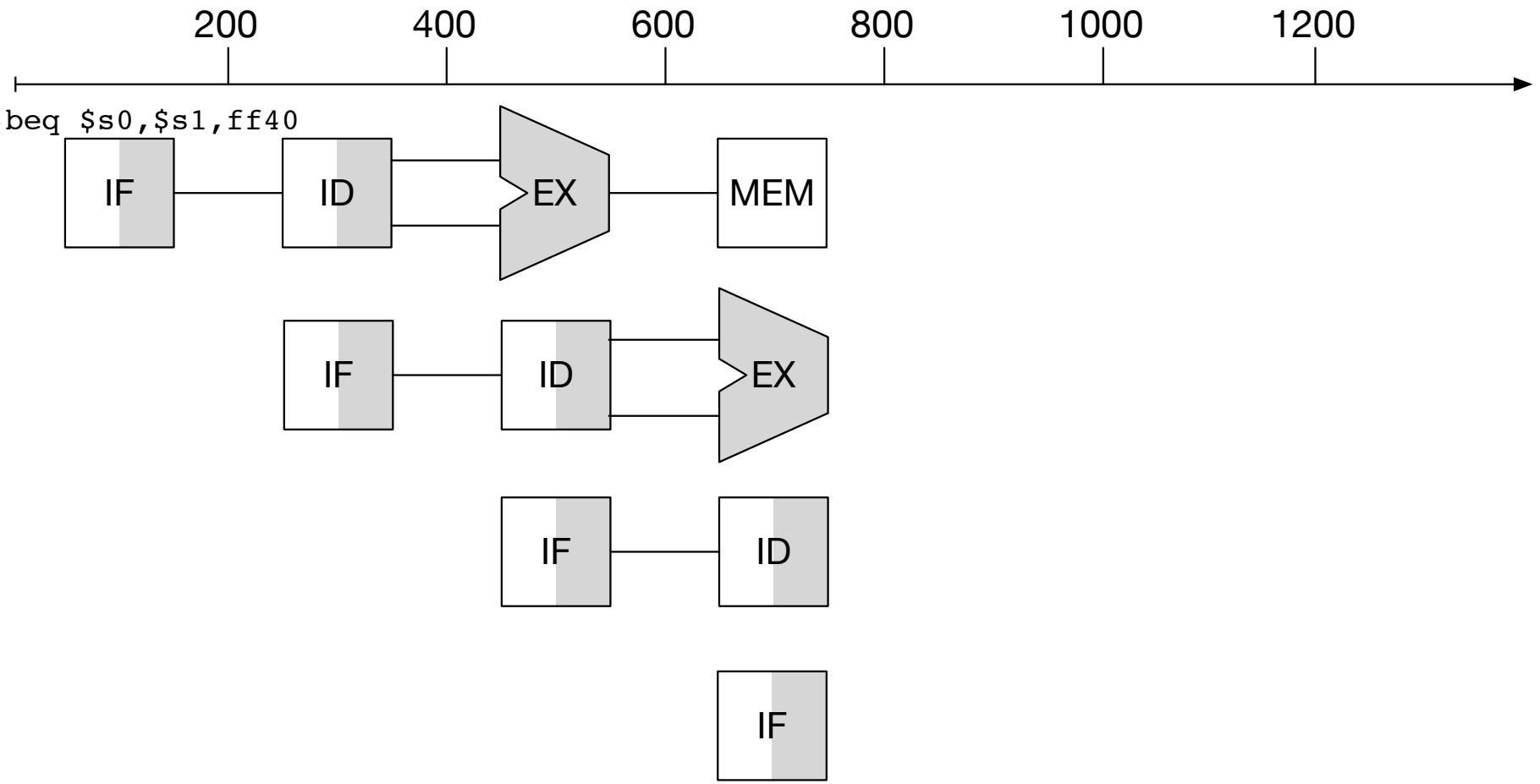
Branch Execution



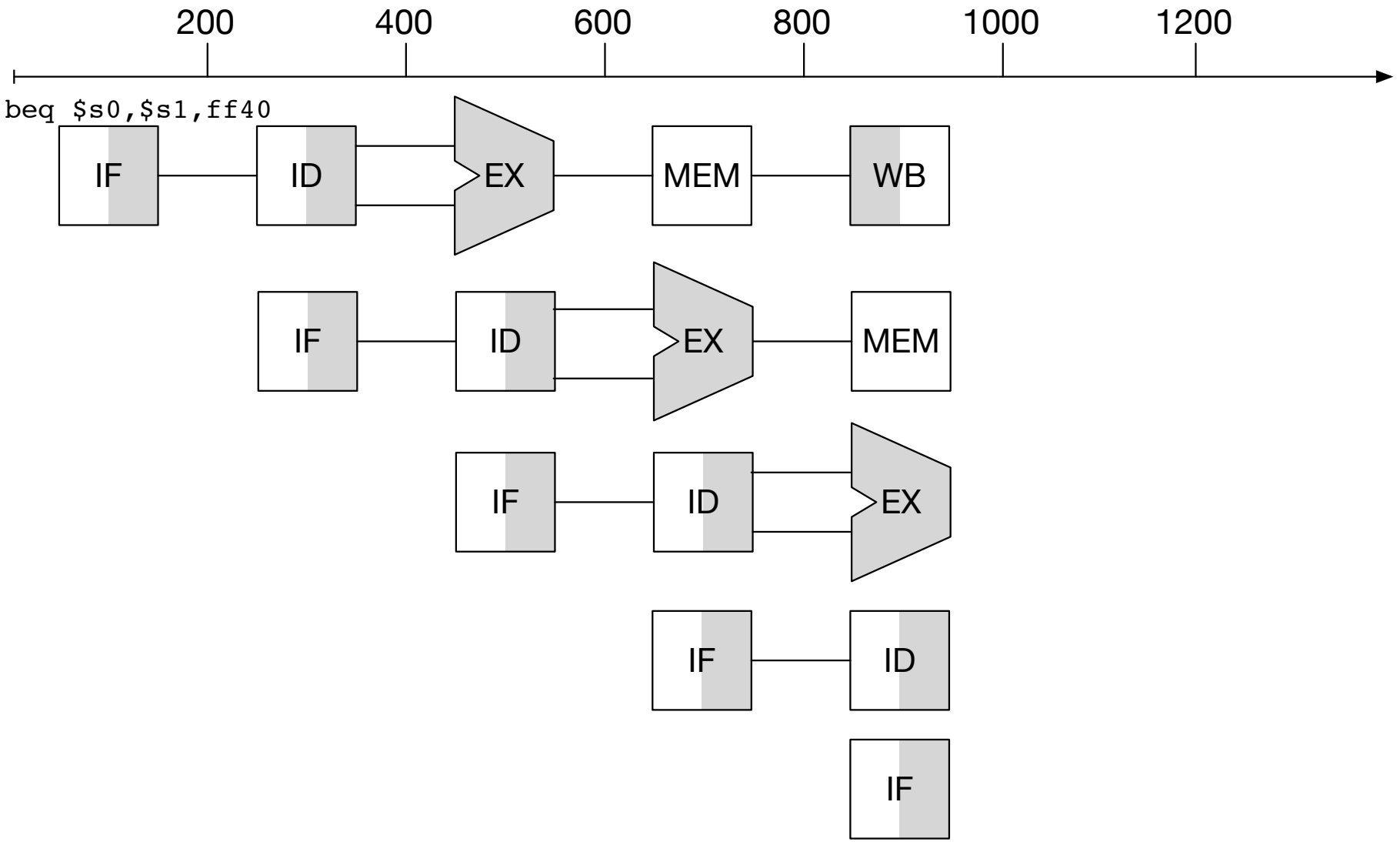
Branch Execution



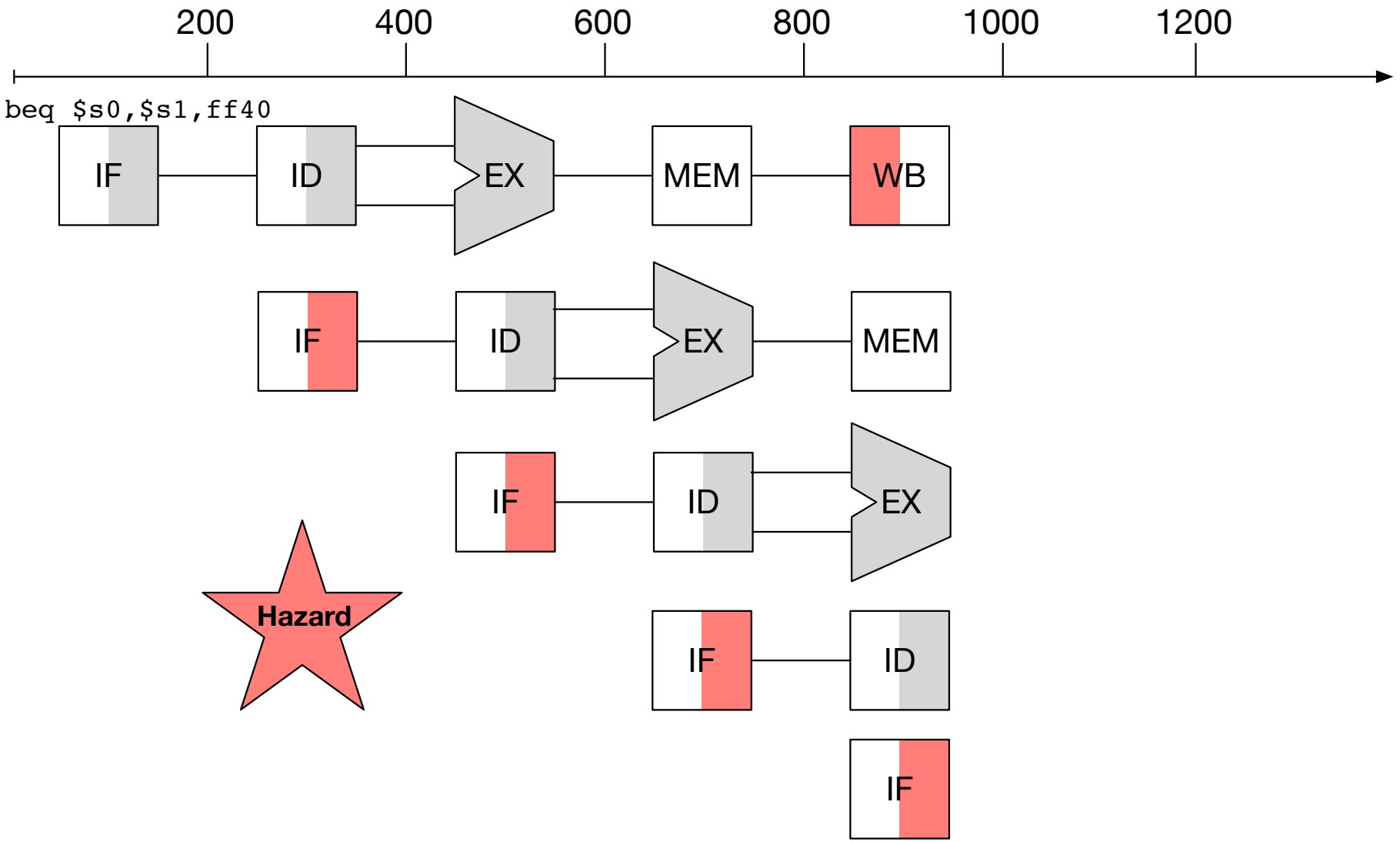
Branch Execution



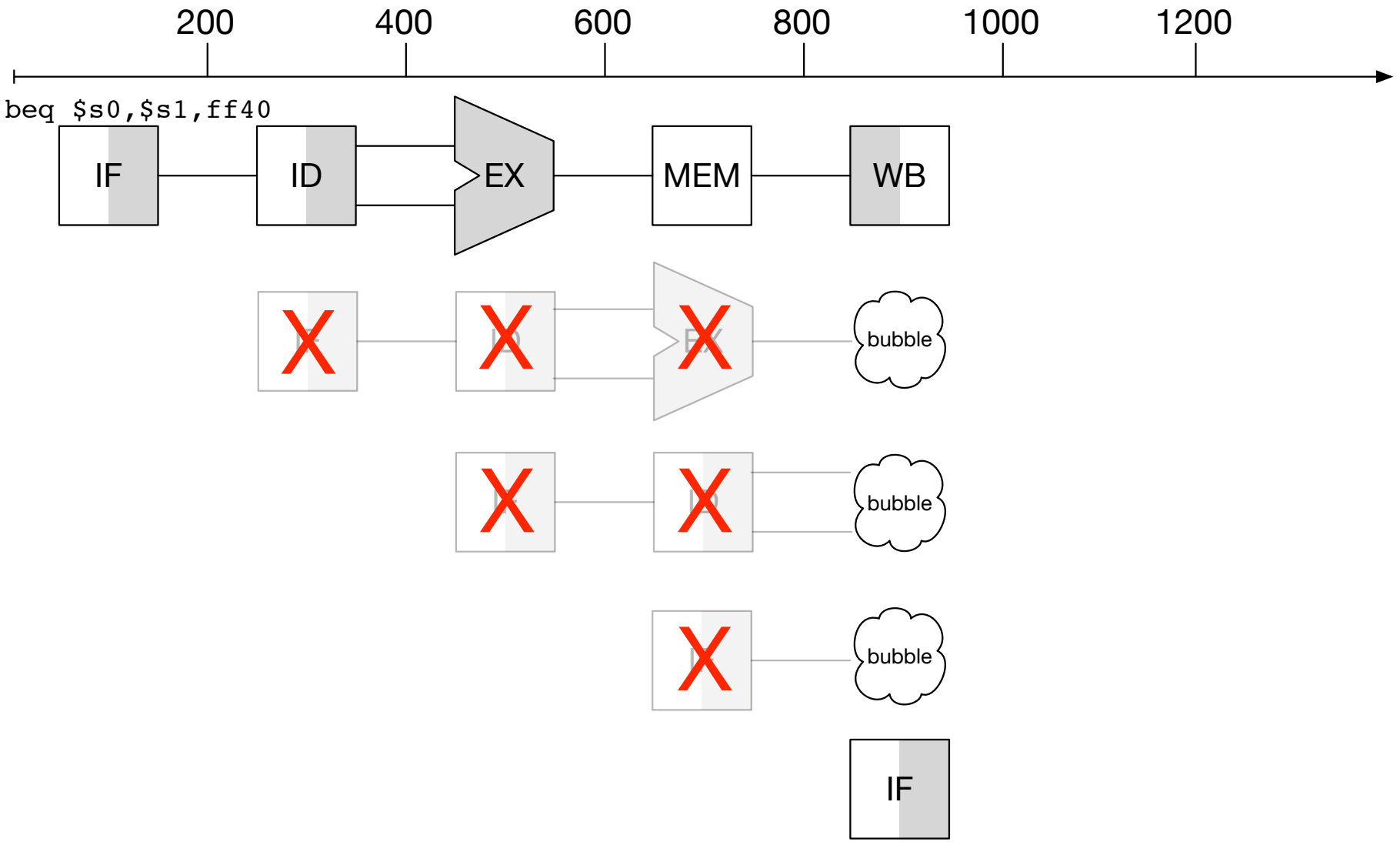
Branch Execution



Take Branch: Invalidates Instructions



Flush Instructions



Flush Instructions

- Change program counter
- Instructions in stages IF, ID, EX, MEM

$\Rightarrow$ Re-fetch instruction in IF

$\Rightarrow$ Zero out control lines for ID, EX, MEM

fast branch execution

Idea

- Branch instruction

```
beq $s0, $s1, ff40
```

- Computations required
  - target address (PC + specified offset)
  - condition check (simple equality)

- Idea: carry out these computations quickly

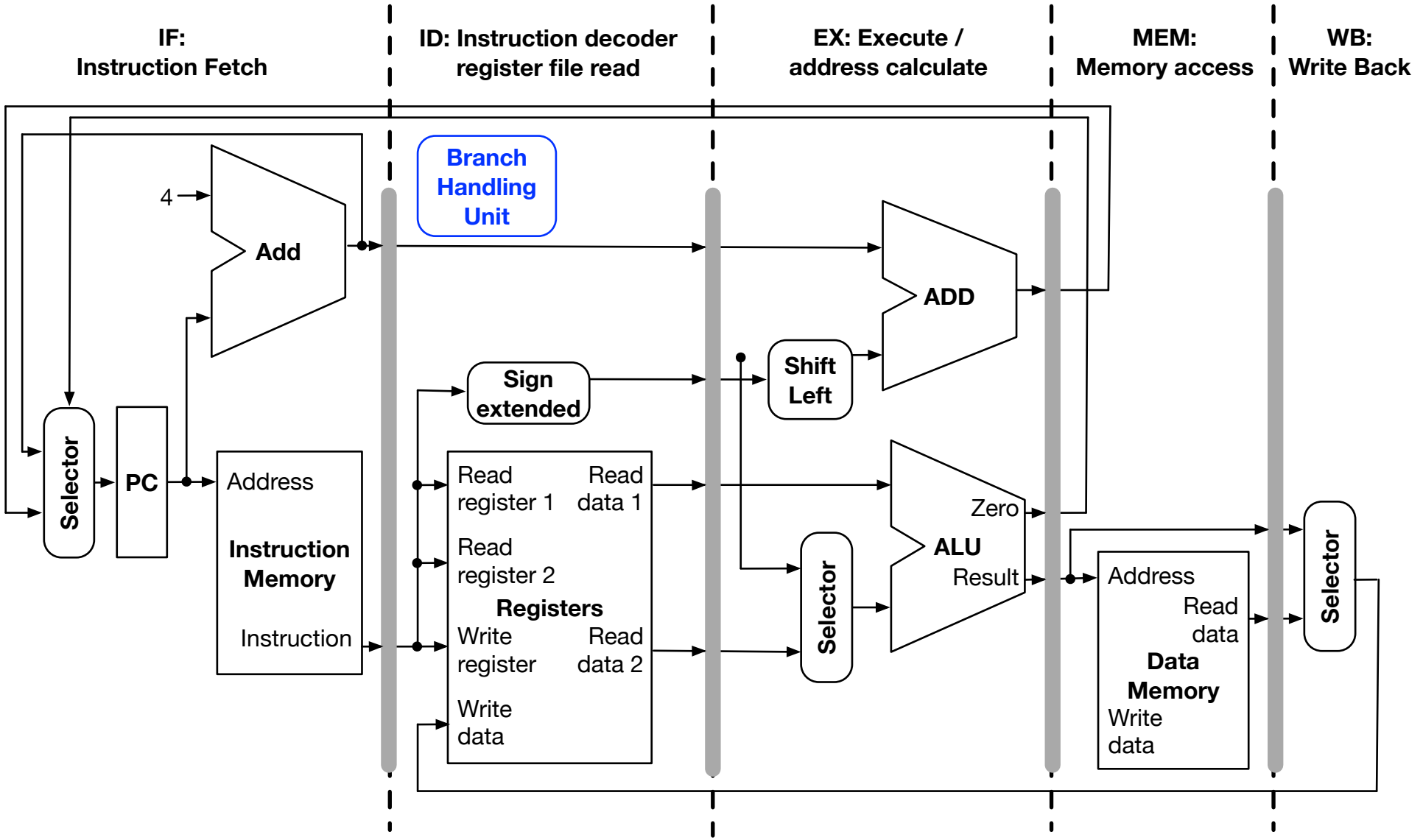
Branch Computation in ID Stage

- Role of ID stage
  - decode what the instruction is
  - look up register values

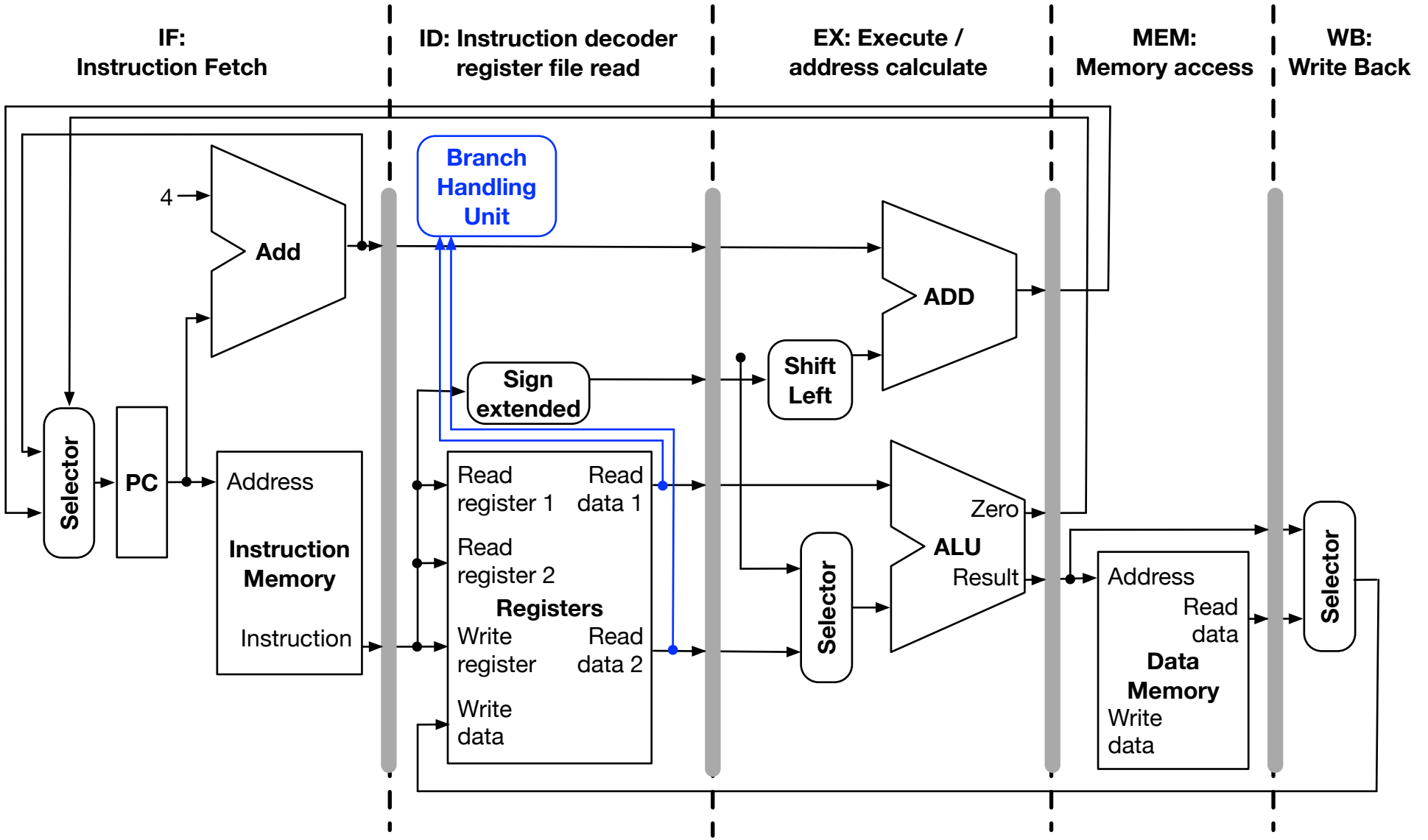
- Now
  - just assume that it is a branch
  - add special wiring for branch calculation
  - if not a branch, ignore results

- Benefit: reduce branch flushing from 3 to 1 instruction

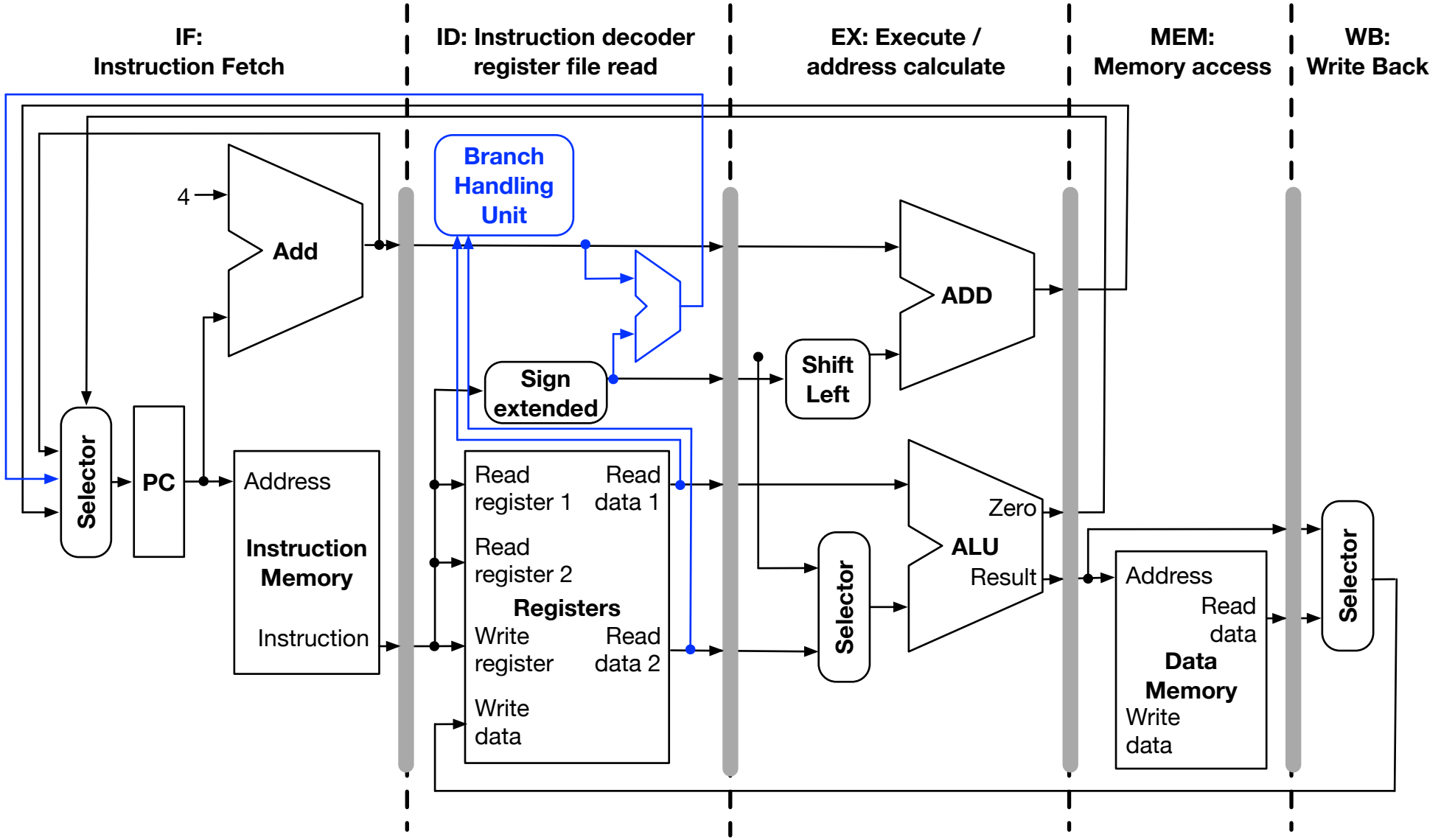
Pipeline with Hazard Detection Unit



Register Values for Condition Check



Address Calculation



Data Hazard

- Condition check requires register values
- These may be changed by preceding instructions in pipeline

$\Rightarrow$ Forwarding and stalling needed

- Adds complexity

branch prediction

Prediction

- So far: predict branch not taken
- Compiler may order instructions $\rightarrow$ more frequent case in sequence
- Now: dynamic branch prediction based on branch history table

Branch History Table

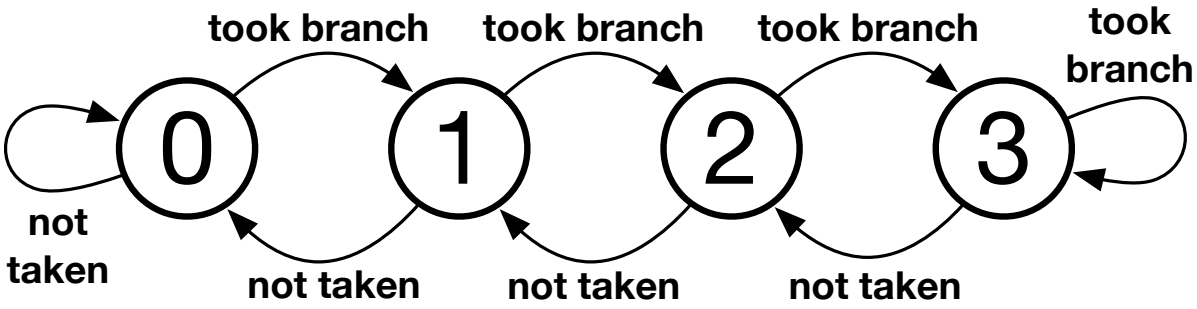
- Idea: keep record of branch history
- Many branches, many executions
- Keep it simple:
  - index by lower order bits of branch address (ignore collisions)
  - just store last decison (1 bit)
- Special memory in ID stage

Example: Loop

- Example: Loop 9 times, then exit loop
- False predictions
  - last iteration (not taken, after 9 times taken)
  - first iteration (taken, previously exited loop)
- Prediction accuracy: 8/10 = 80%
- Can we do better?

2 Bits

- Idea: record frequency



- Previous example (loop 9 times, then exit loop)

Iteration	Value	Prediction
1	2	take branch (correct)
2	3	take branch (correct)
...	...	...
9	3	take branch (correct)
10	3	take branch (wrong)