Pipelined CPU

Jinyang Li

Slides are based on Patterson and Hennessy

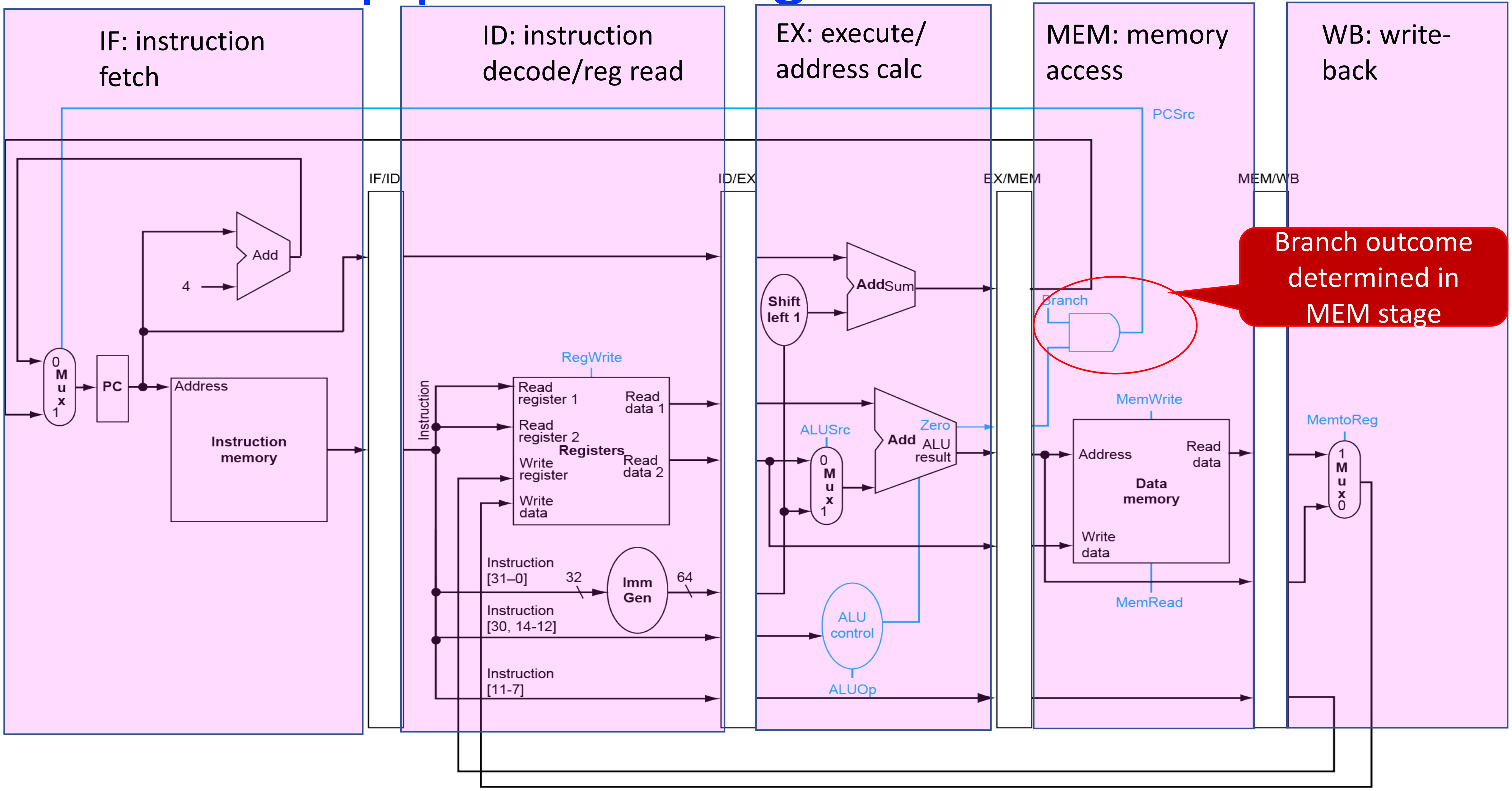
What we’ve learnt so far

- Single cycle RISC-V CPU design
- 5 stage pipelined RISC-V CPU
- Challenges: hazards:
  - Structure (To mitigate, add resources)
  - Data (To mitigate, do forwarding/bypassing)
    - Load-use must stall
  - Control (??)

Today’s lesson

- Handling control hazard
  - Branch hazard
  - exceptions
- Briefly, more advanced topics:
  - Multiple-issue
  - Subword parallelism (SIMD)

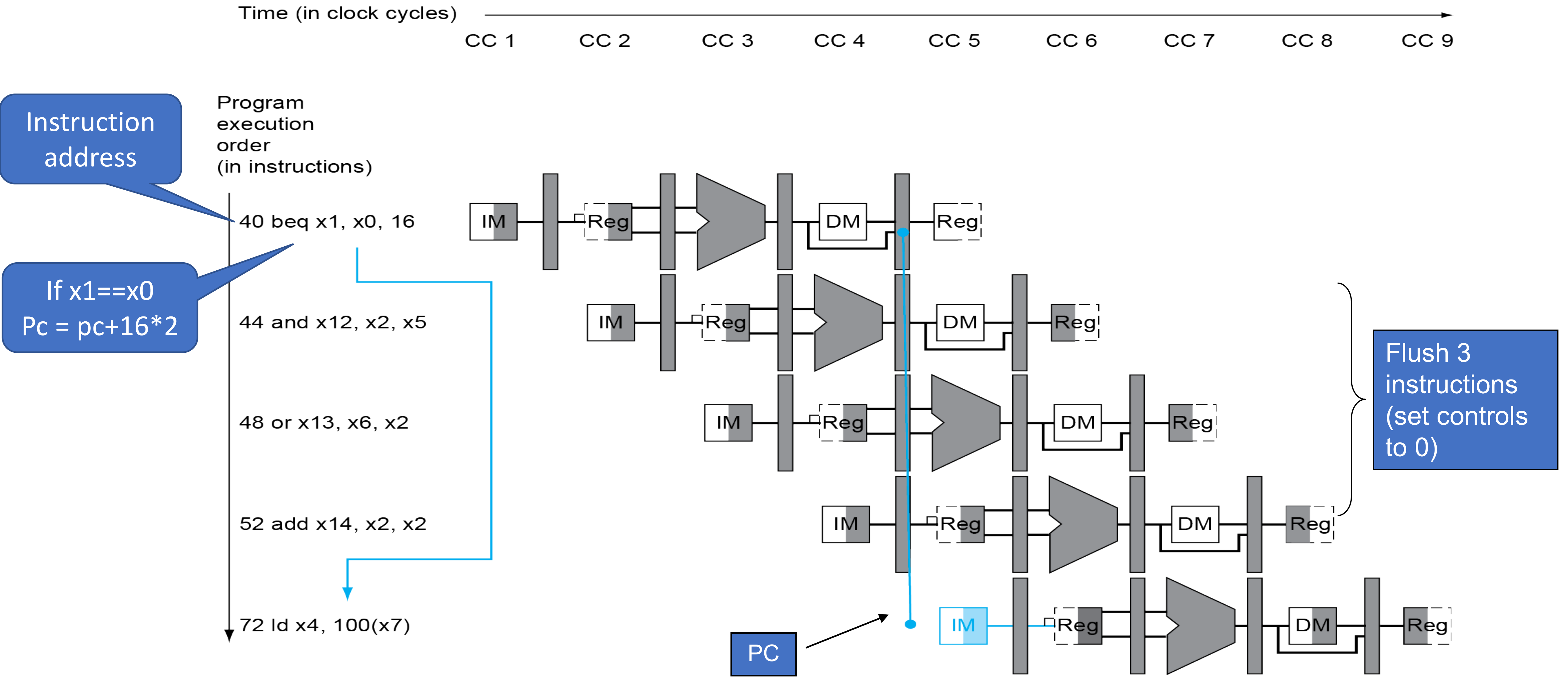
Recall our pipelined design so far



Branch outcome determined in MEM stage

Branch Hazard

- If branch outcome is determined in MEM

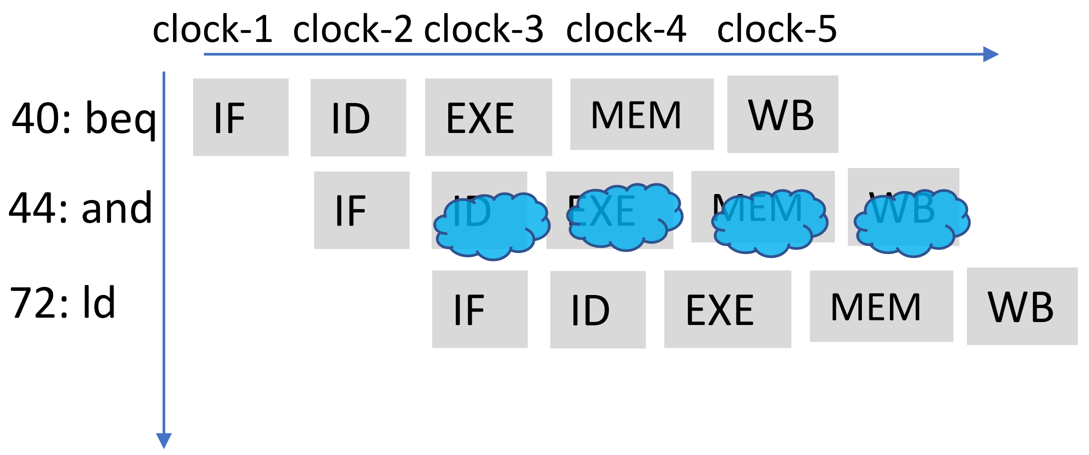


Reducing Branch Delay

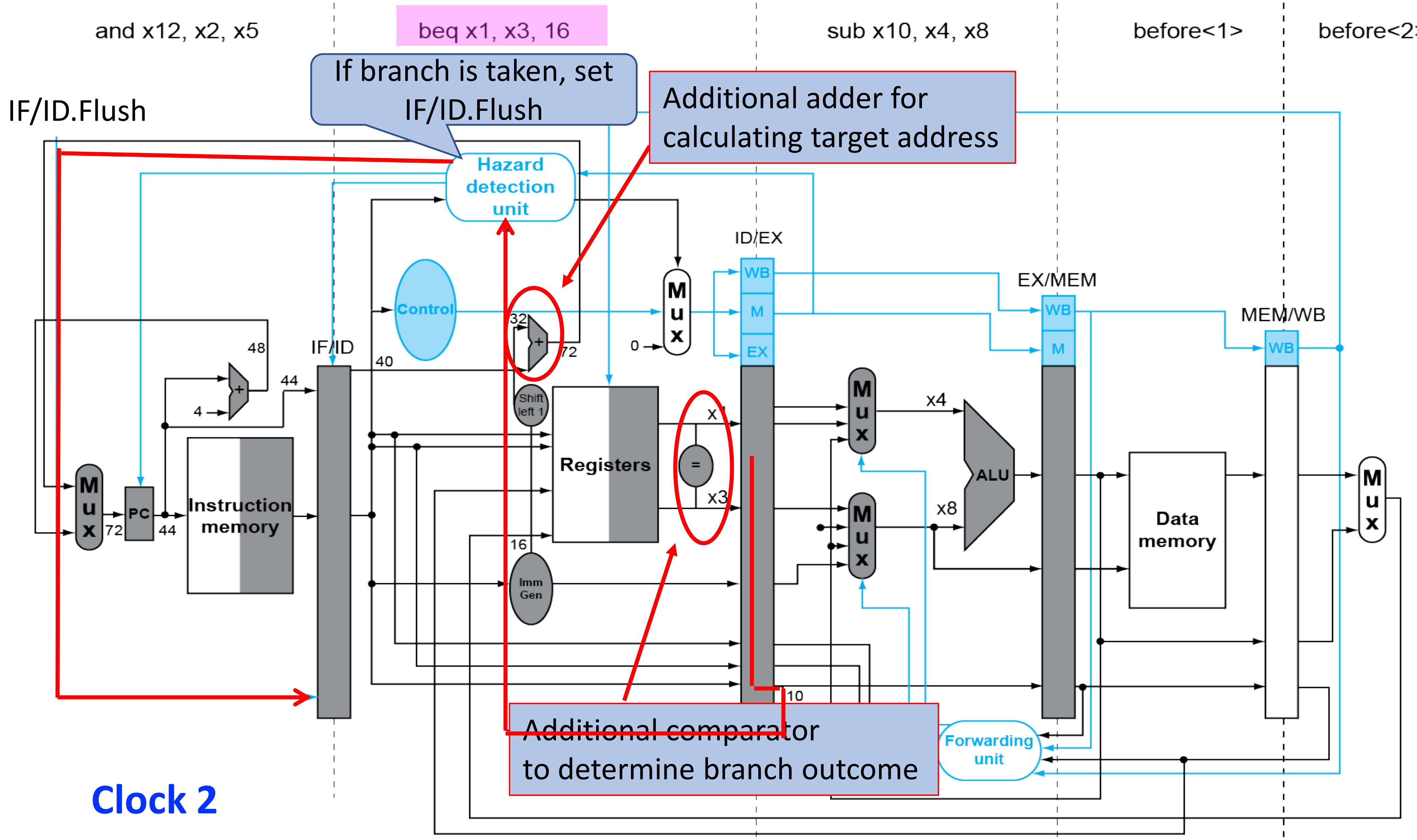
- Add hardware to determine branch outcome earlier (e.g. ID instead of MEM) → fewer instructions to flush

```
36:   sub   x10, x4, x8
40:   beq   x1,  x3, 16   // PC-relative branch
                          // to 40+16*2=72
44:   and   x12, x2, x5
48:   orr   x13, x2, x6
52:   add   x14, x4, x2
56:   sub   x15, x6, x7
      ...
72:   ld    x4, 50(x7)
```

How many instructions to flush
if branch outcome is known in ID?

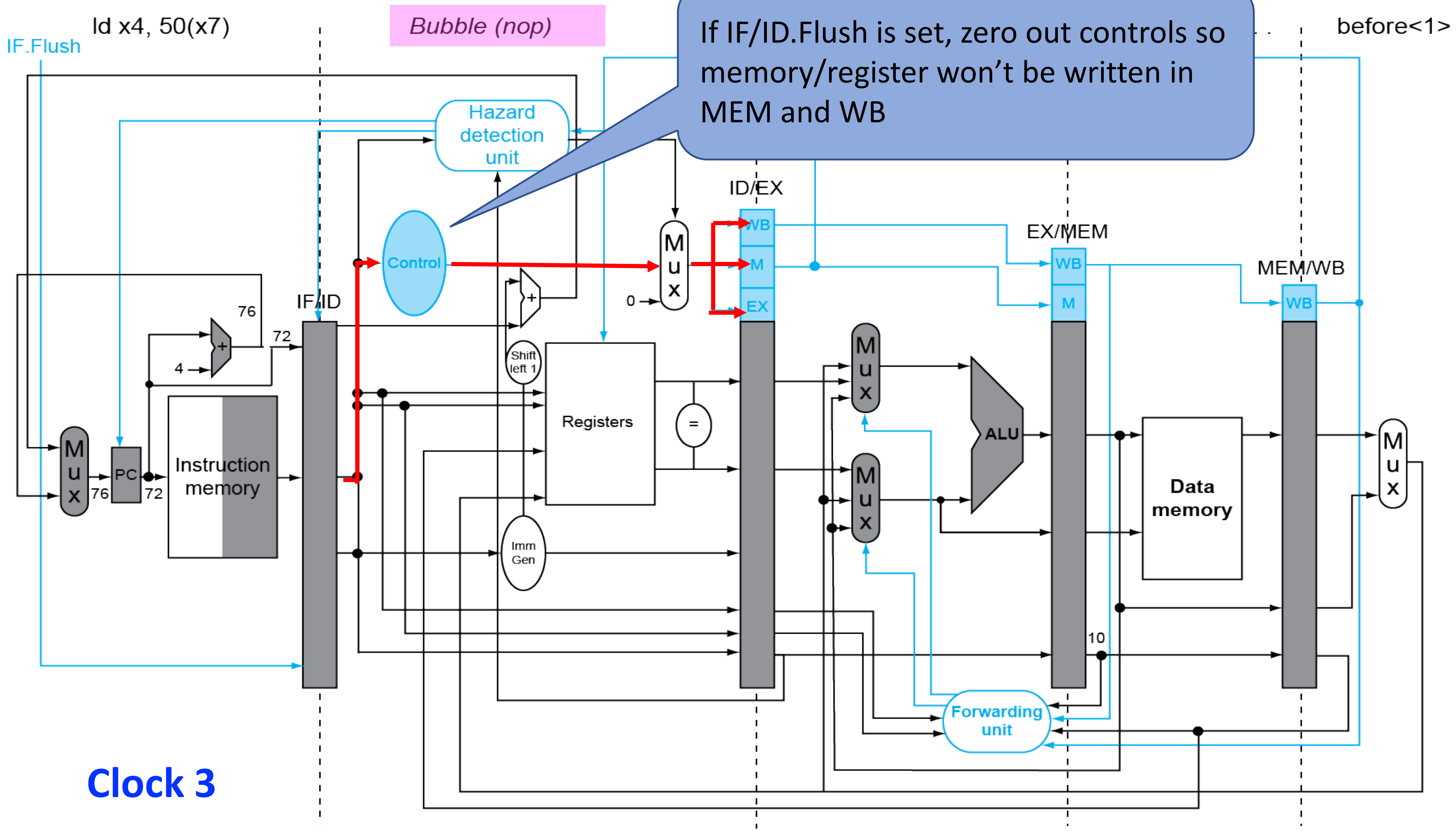


Branch determined in ID and is taken



Clock 2

Branch determined in ID and is taken



If IF/ID.Flush is set, zero out controls so memory/register won't be written in MEM and WB

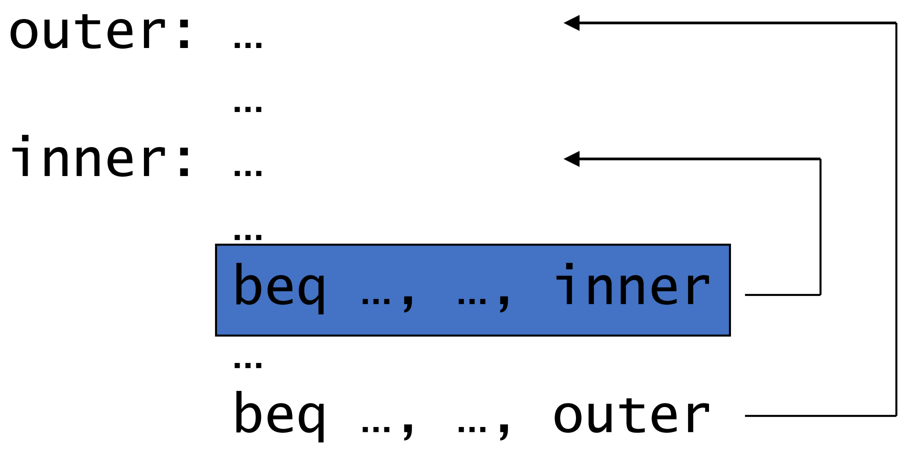
Clock 3

Dynamic Branch Prediction

- Our simple 5-stage pipeline’s branch penalty is 1 bubble, but
  - In deeper pipelines, branch penalty is more significant
- Solution: dynamic prediction
  - Branch prediction buffer (aka branch history table)
    - Indexed by recent branch instruction addresses
    - Stores outcome (taken/not taken) } 1-bit predictor
  - To execute a branch
    - Check table, fetch fall-through or target based on predicted outcome.
    - If wrong, flush pipeline and flip prediction

1-bit Predictor's shortcoming

- Inner loop branches mispredicted twice!

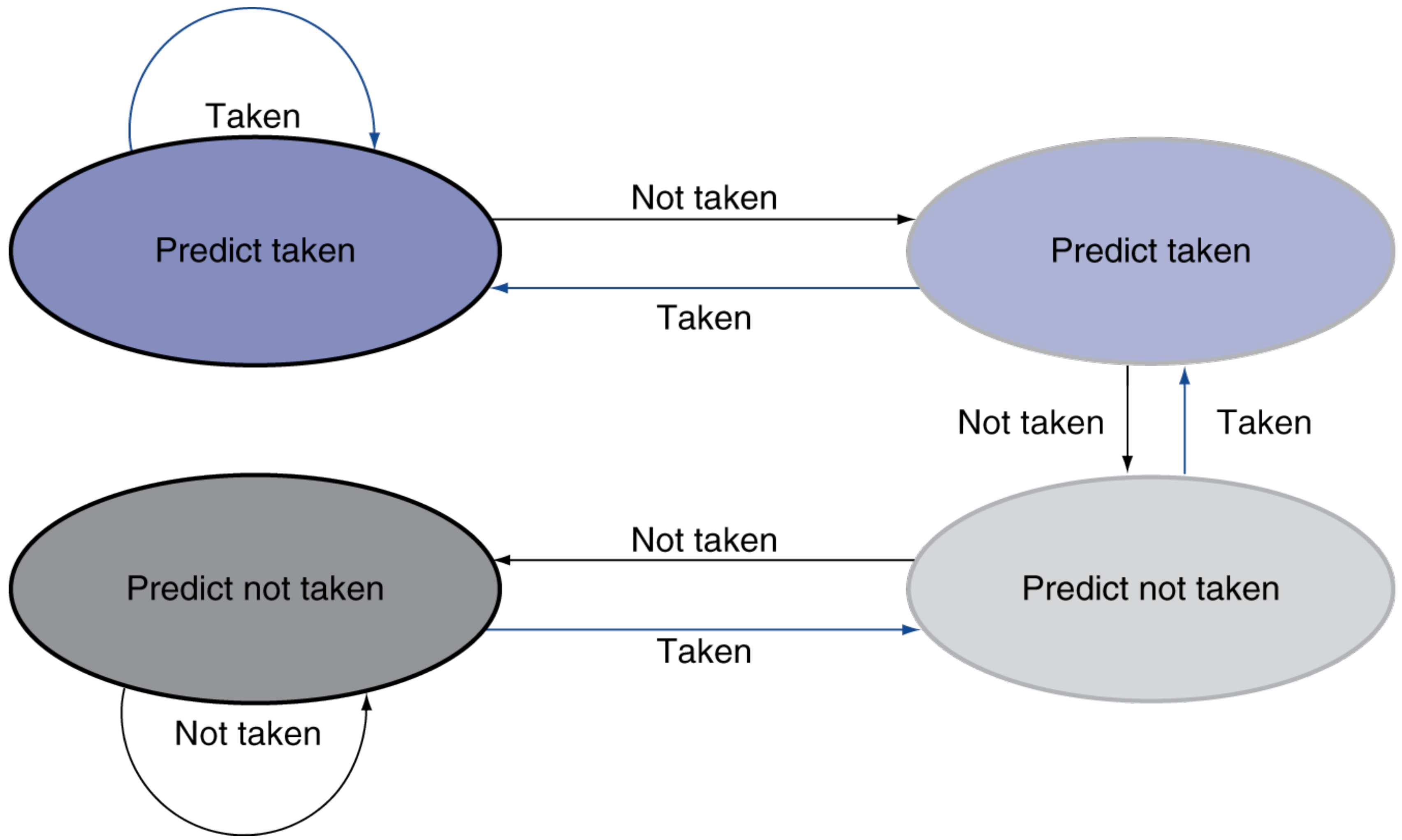


```
outer: …
       …
inner: …
       …
       beq …, …, inner
       …
       beq …, …, outer
```

- Mispredict as taken on last iteration of inner loop
- Then mispredict as not taken on first iteration of inner loop next time around

2-Bit Predictor

- Only change prediction on two successive mispredictions



Calculating Branch Target (needed if branch is predicted taken)

- Even with predictor, still need to calculate the target address
  - 1-cycle penalty for a taken branch
- Branch target buffer
  - Cache of target addresses
  - Indexed by PC when instruction fetched
    - If hit and instruction is branch predicted taken, fetch target immediately

Beyond basic pipelining

- ILP: execute multiple instructions in parallel
- To increase ILP
  - Deeper pipeline
    - Less work per stage $\Rightarrow$ shorter clock cycle
  - Multiple issue
    - Replicate pipeline stages $\Rightarrow$ multiple pipelines
    - Start multiple instructions per clock cycle
      - Finish multiple Instructions Per Cycle (IPC>1)
    - E.g., 4GHz 4-way multiple-issue
      - 16 billion instructions/sec, peak IPC = 4 (CPI = 1/IPC = 0.25)
    - Challenges: dependencies among multi-issued instructions
      - reduce peak IPC

Dynamic Multiple Issue

- "Superscalar" processors
- CPU decides whether to issue 0, 1, 2, … each cycle
  - Avoiding structural and data hazards
- Dynamic pipeline scheduling:
  - Execute instructions out of order to avoid stalls
  - But commit results to registers in order

```
ld   x31,20(x21)
add  x1,x31,x2
sub  x23,x23,x3
```

Can start sub while add is waiting for ld

Does Multiple Issue Work?

- Yes, but not as much as we'd like
- Programs have real dependencies that limit ILP
- Some dependencies are hard to eliminate
    - e.g., pointer aliasing
- Memory delays and limited bandwidth
    - Hard to keep pipelines full
- Speculation can help if done well

Other ways to improve performance: subword parallelism

- ML/graphics applications perform same operations on vectors
  - Partition a 128-bit adder into:
    - Sixteen 8-bit adds
    - Eight 16-bit adds
    - Four 32-bit adds
- Also called data-level parallelism, vector parallelism, or Single Instruction, Multiple Data (SIMD)
- Intel’s AVX introduces 256-bit floating point registers
    - 8 single-precision ops
    - 4 double-precision ops

Other ways to improve performance: loop unrolling (software)

- Replicate loop body to expose more parallelism
  - Reduces loop-control overhead
  - Avoid loop-carried dependency
    - Store followed by a load of the same register

Other ways to improve performance: loop unrolling

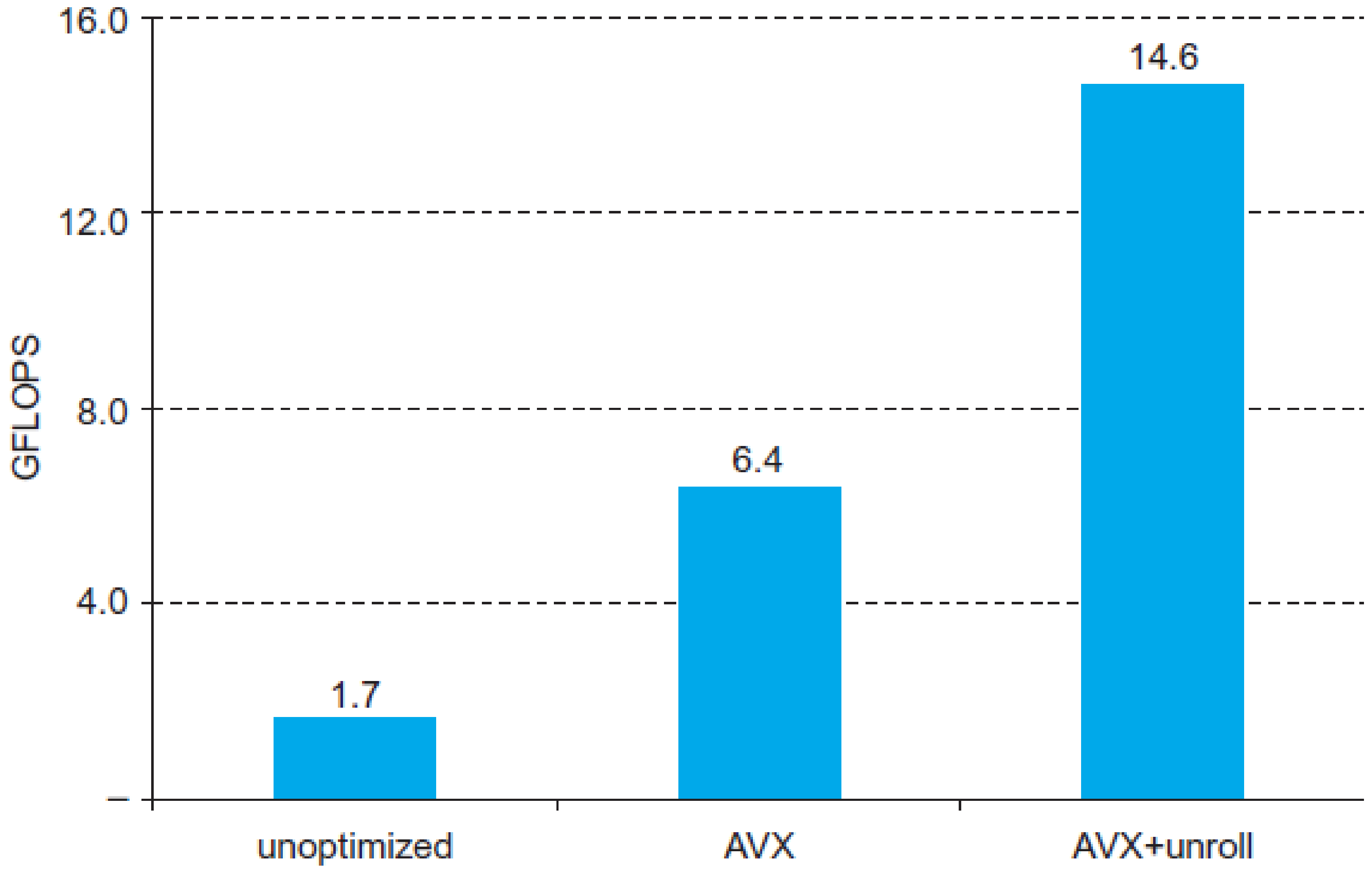
- Loop unrolling: replicate loop body
- Unrolling exposes more parallelism for multi-issue pipelined CPU
  - Reduces loop-control overhead
  - Avoid loop-carried dependency
    - Store followed by a load of the same register

```
long x[100];
…
for (int i = 0; i < 100; i++) {
    x[i] *= 2;
}
```

```
//suppose x1=&x[0]
//x2 = &x[100]
Loop:
    ld  x31, 0(x1)
    add x31, x31, x30
    sd  x30, 0(x1)
    addi x1, x1, 8
    blt x1, x2, Loop
```

```
//suppose x20=&x[0]
//x22 = &x[100]
Loop:
    ld  x31, 0(x1)
    add x31, x31, x30
    sd  x30, 0(x1)
    ld  x29, 8(x1)
    add x29, x29, x28
    sd  x28, 8(x1)
    addi x20, x20, 16
    blt x20, x22, Loop
```

Performance Impact



Summary on CPU design

- ISA influences design of datapath and control
- Pipelining improves instruction throughput using parallelism
  - More instructions completed per second
  - Latency for each instruction not reduced
- Hazards: structural, data, control
- Multiple issue and dynamic scheduling (ILP)
  - Dependencies limit achievable parallelism
- SIMD achieves data-level parallelism