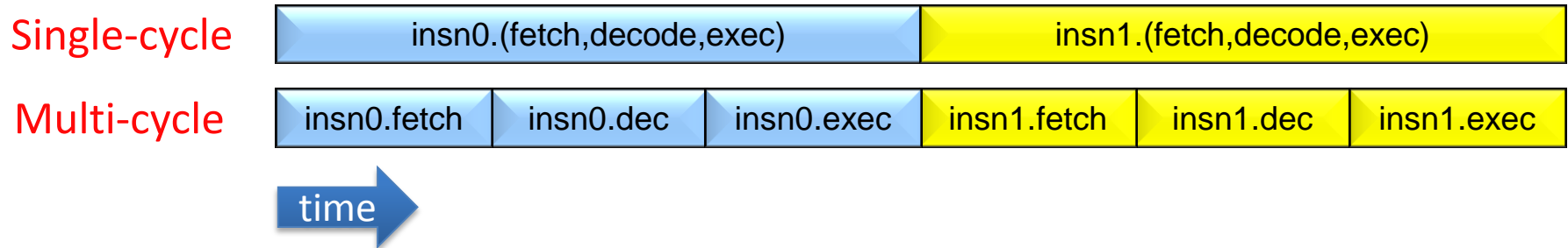


CSE 502:

Computer Architecture

Core Pipelining

Before there was pipelining...



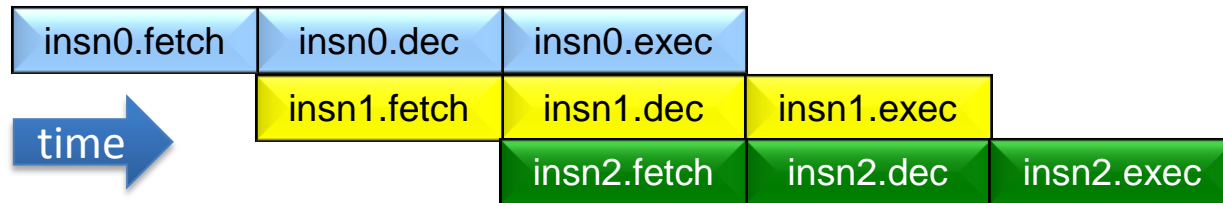
- Single-cycle control: hardwired
 - Low CPI (1)
 - Long clock period (to accommodate slowest instruction)
- Multi-cycle control: micro-programmed
 - Short clock period
 - High CPI
- Can we have both low CPI and short clock period?

Pipelining

Multi-cycle



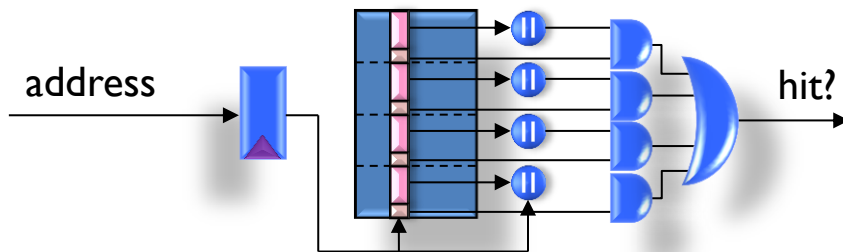
Pipelined



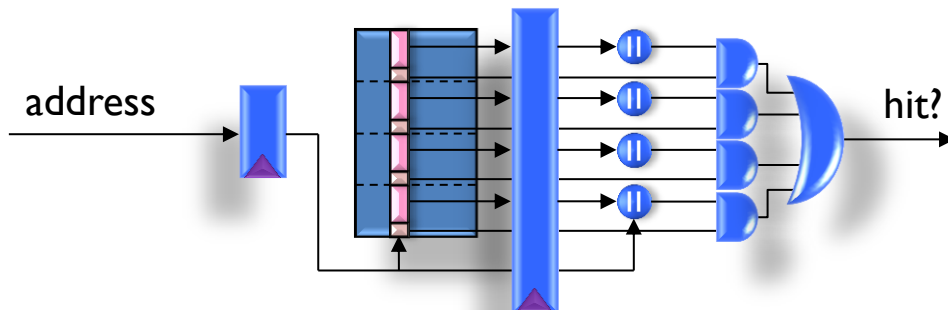
- Start with multi-cycle design
- When insn0 goes from stage 1 to stage 2
... insn1 starts stage 1
- Each instruction passes through all stages
... but instructions enter and leave at faster *rate*

Can have as many insns in flight as there are stages

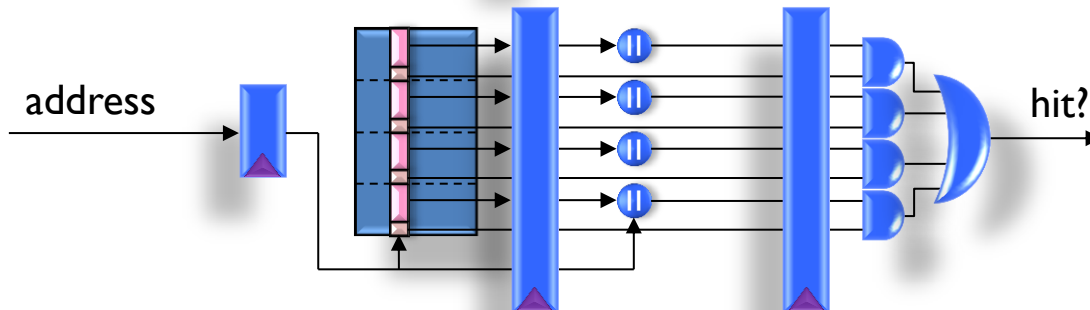
Pipeline Examples



Stage delay = n
Bandwidth = $\sim (1/n)$



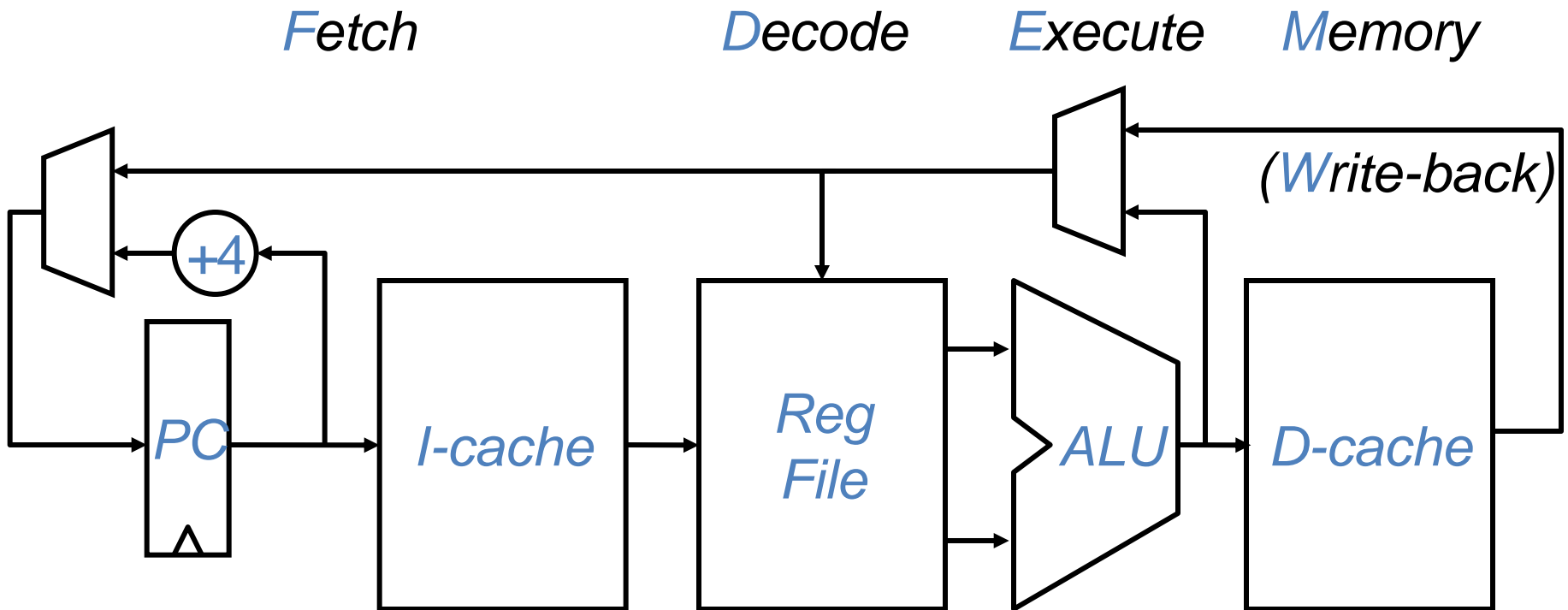
Stage delay = $n/2$
Bandwidth = $\sim (2/n)$



Stage delay = $n/3$
Bandwidth = $\sim (3/n)$

Increases throughput at the expense of latency

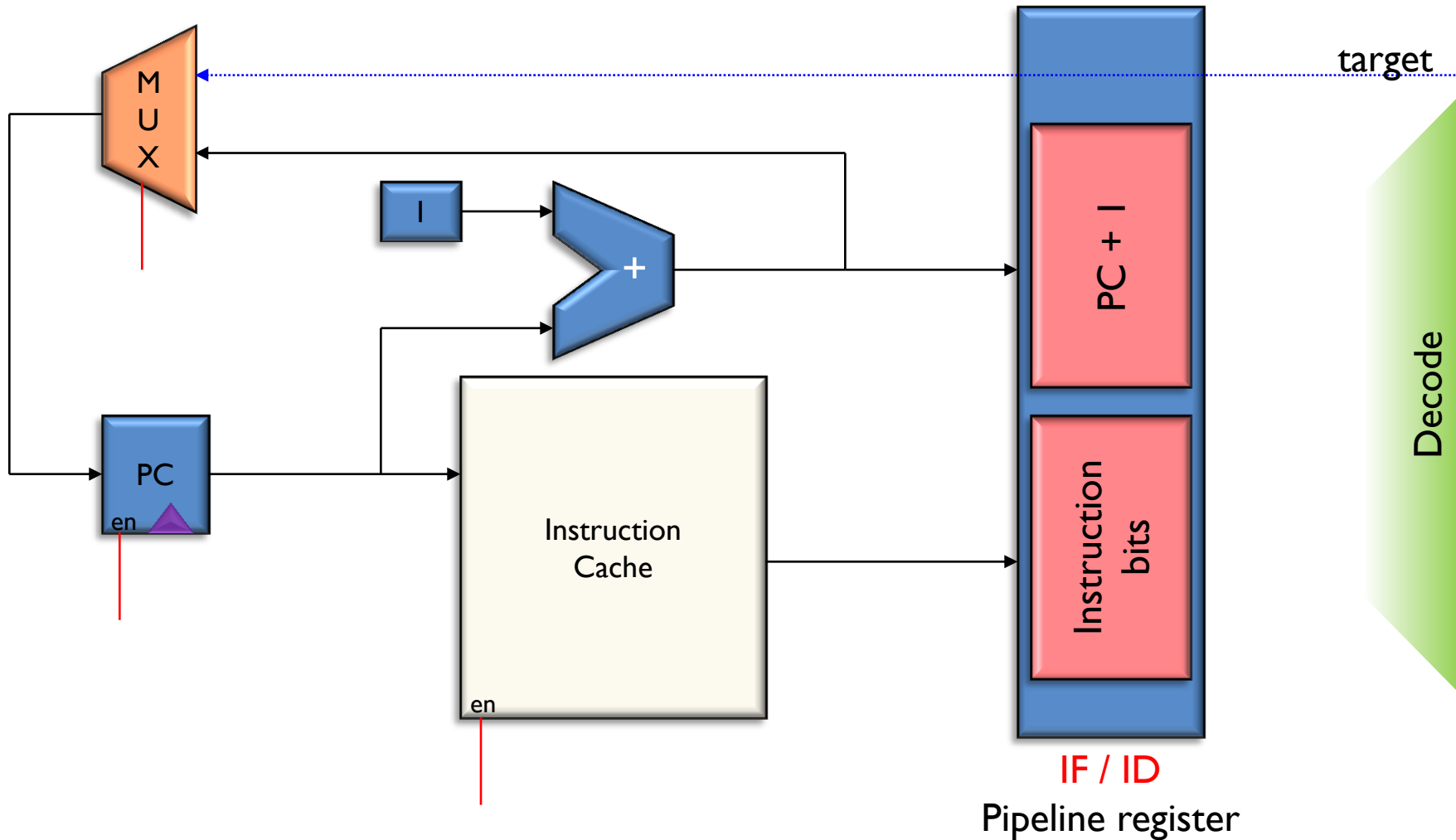
Processor Pipeline Review



Stage 1: Fetch

- Fetch an instruction from memory every cycle
 - Use PC to index memory
 - Increment PC (assume no branches for now)
- Write state to the pipeline register (IF/ID)
 - The next stage will read this pipeline register

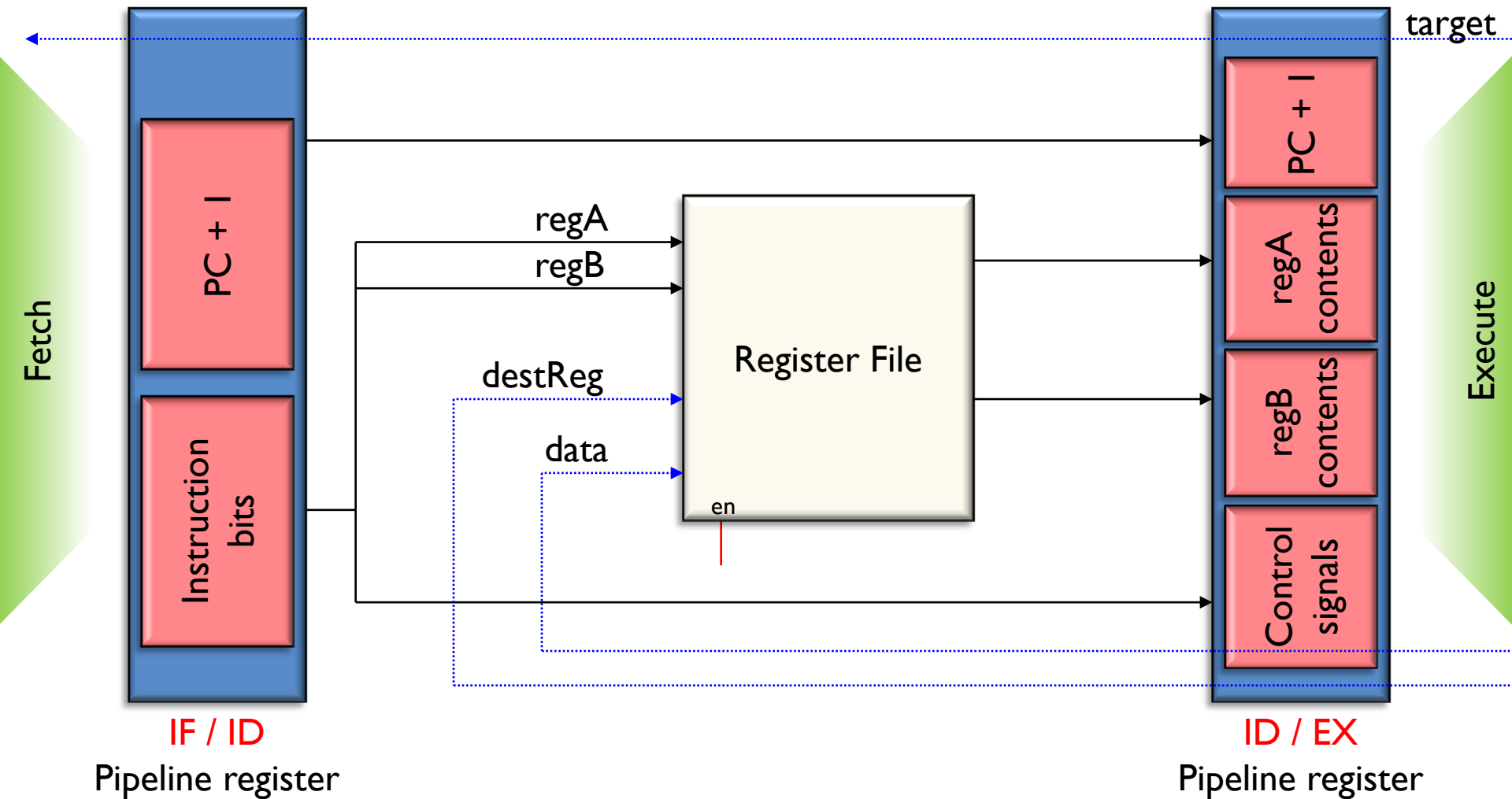
Stage 1: Fetch Diagram



Stage 2: Decode

- Decodes opcode bits
 - Set up Control signals for later stages
- Read input operands from register file
 - Specified by decoded instruction bits
- Write state to the pipeline register (ID/EX)
 - Opcode
 - Register contents
 - PC+1 (even though decode didn't use it)
 - Control signals (from insn) for opcode and destReg

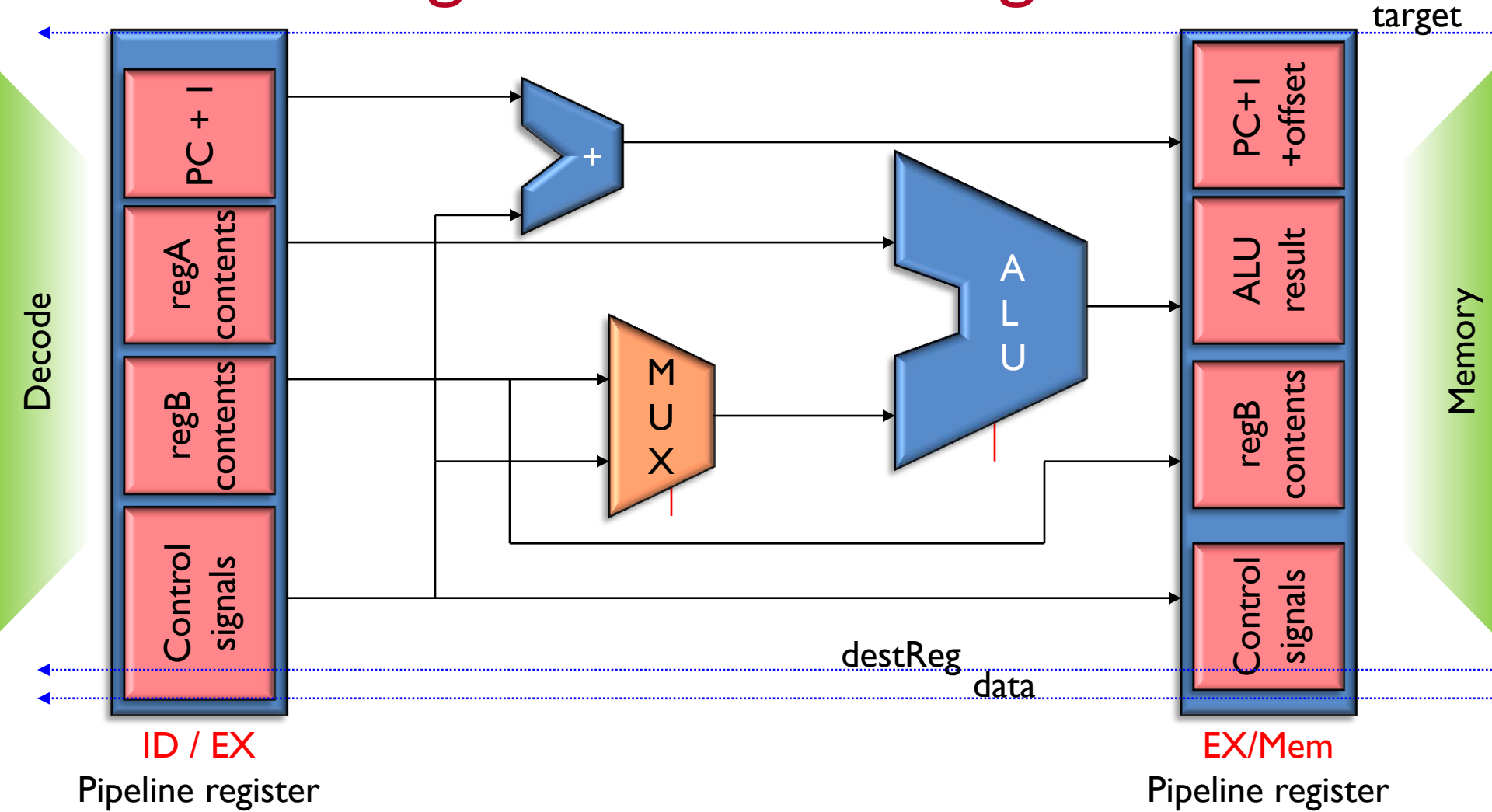
Stage 2: Decode Diagram



Stage 3: Execute

- Perform ALU operations
 - Calculate result of instruction
 - Control signals select operation
 - Contents of regA used as one input
 - Either regB or constant offset (from insn) used as second input
 - Calculate PC-relative branch target
 - $PC+1+(\text{constant offset})$
- Write state to the pipeline register (EX/Mem)
 - ALU result, contents of regB, and $PC+1+\text{offset}$
 - Control signals (from insn) for opcode and destReg

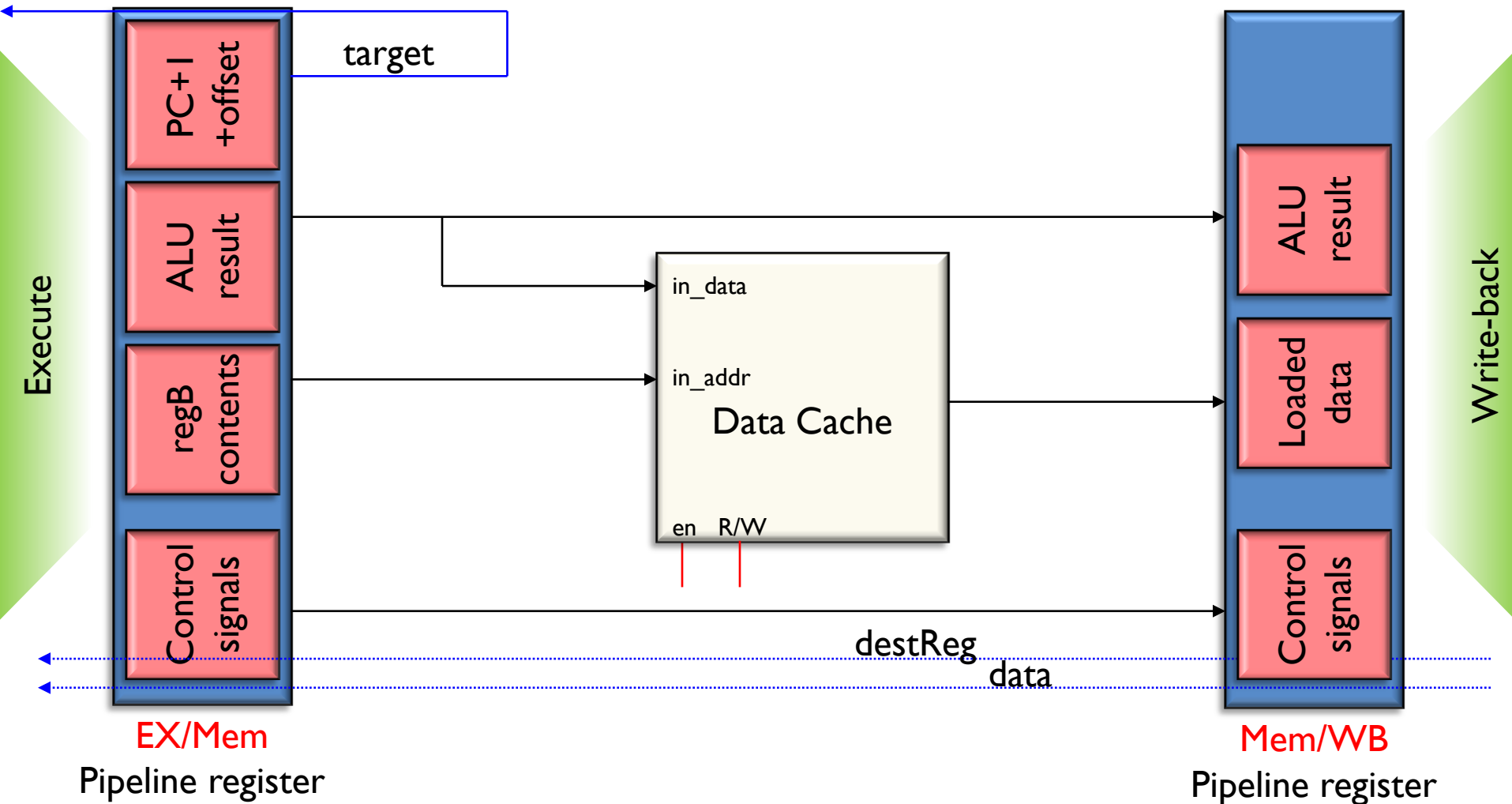
Stage 3: Execute Diagram



Stage 4: Memory

- Perform data cache access
 - ALU result contains address for LD or ST
 - Opcode bits control R/W and enable signals
- Write state to the pipeline register (Mem/WB)
 - ALU result and Loaded data
 - Control signals (from insn) for opcode and destReg

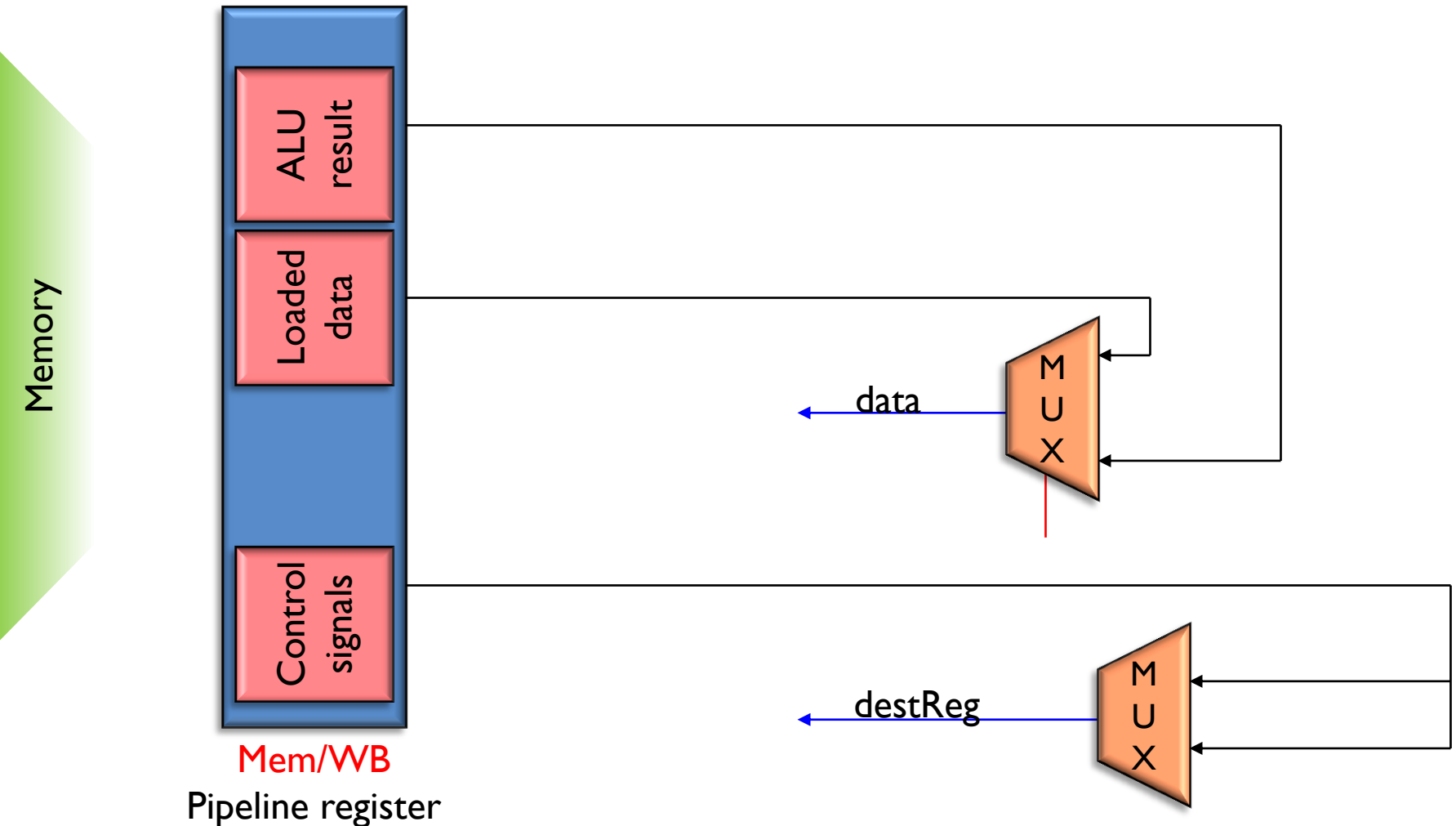
Stage 4: Memory Diagram



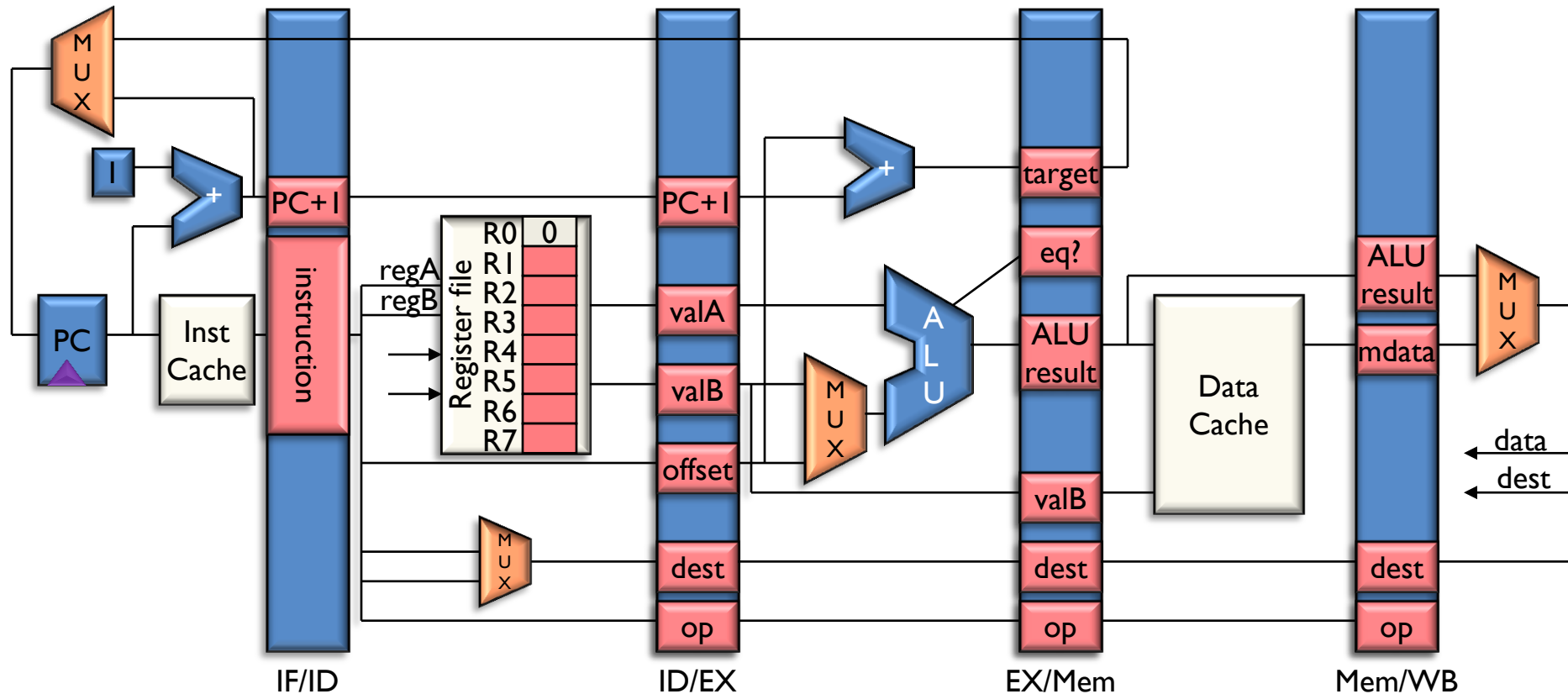
Stage 5: Write-back

- Writing result to register file (if required)
 - Write Loaded data to destReg for LD
 - Write ALU result to destReg for arithmetic insn
 - Opcode bits control register write enable signal

Stage 5: Write-back Diagram



Putting It All Together



Pipelining Idealism

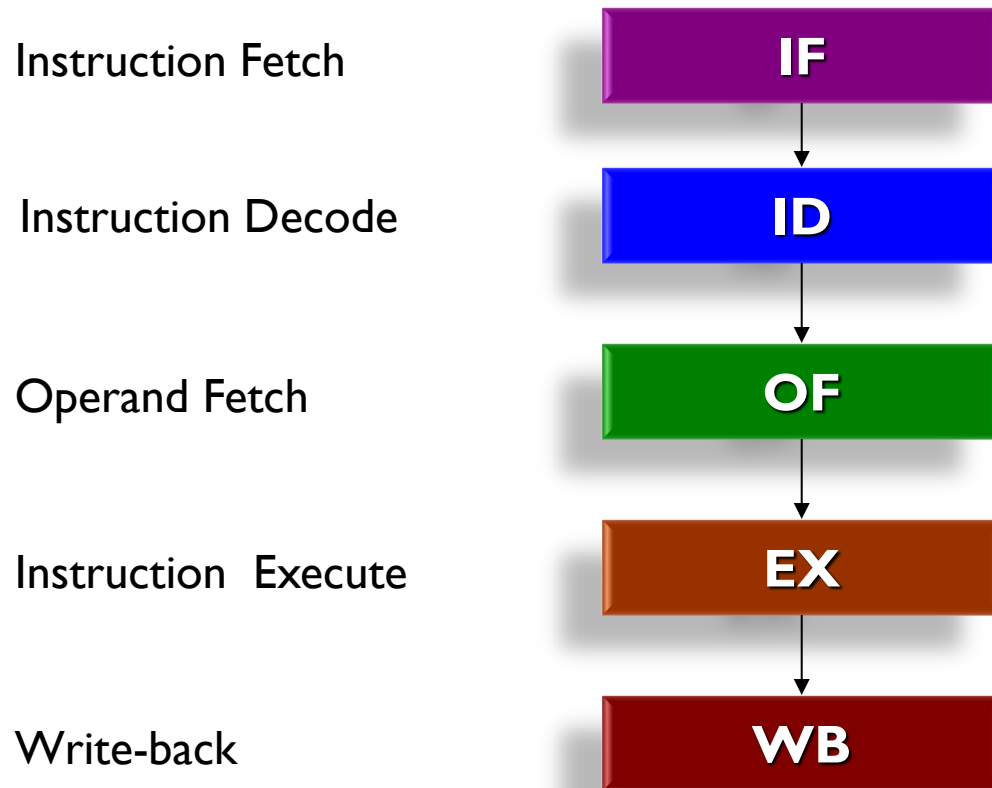
- Uniform Sub-operations
 - Operation can partitioned into uniform-latency sub-ops
- Repetition of Identical Operations
 - Same ops performed on many different inputs
- Repetition of Independent Operations
 - All repetitions of op are mutually independent

Pipeline Realism

- Uniform Sub-operations ... NOT!
 - Balance pipeline stages
 - Stage quantization to yield balanced stages
 - Minimize internal fragmentation (left-over time near end of cycle)
- Repetition of Identical Operations ... NOT!
 - Unifying instruction types
 - Coalescing instruction types into one “multi-function” pipe
 - Minimize external fragmentation (idle stages to match length)
- Repetition of Independent Operations ... NOT!
 - Resolve data and resource hazards
 - Inter-instruction dependency detection and resolution

Pipelining is expensive

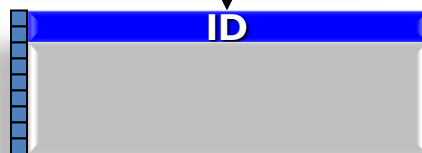
The Generic Instruction Pipeline



Balancing Pipeline Stages



$T_{IF} = 6$ units



$T_{ID} = 2$ units



$T_{OF} = 9$ units



$T_{EX} = 5$ units



$T_{OS} = 9$ units

Without pipelining

$$T_{cyc} \approx T_{IF} + T_{ID} + T_{OF} + T_{EX} + T_{OS} \\ = 31$$

Pipelined

$$T_{cyc} \approx \max\{T_{IF}, T_{ID}, T_{OF}, T_{EX}, T_{OS}\} \\ = 9$$

Speedup = $31 / 9$

Can we do better?

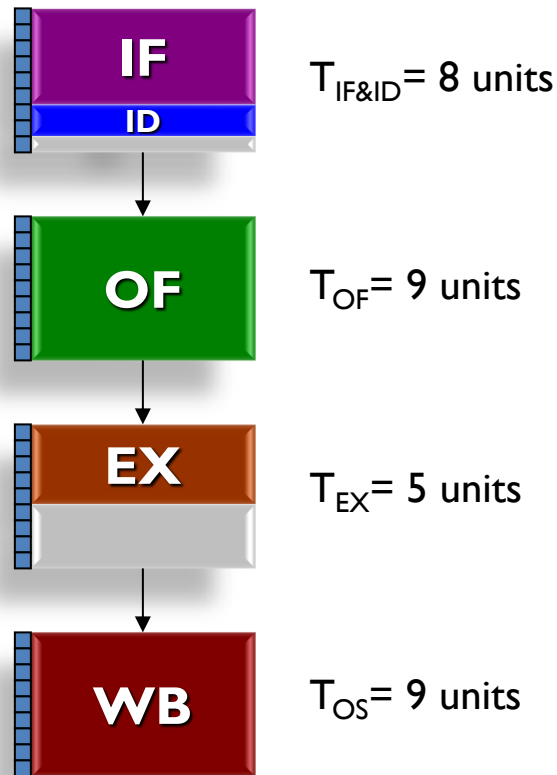
Balancing Pipeline Stages (1/2)

- Two methods for stage quantization
 - Merge multiple sub-ops into one
 - Divide sub-ops into smaller pieces
- Recent/Current trends
 - Deeper pipelines (more and more stages)
 - Multiple different pipelines/sub-pipelines
 - Pipelining of memory accesses

Balancing Pipeline Stages (2/2)

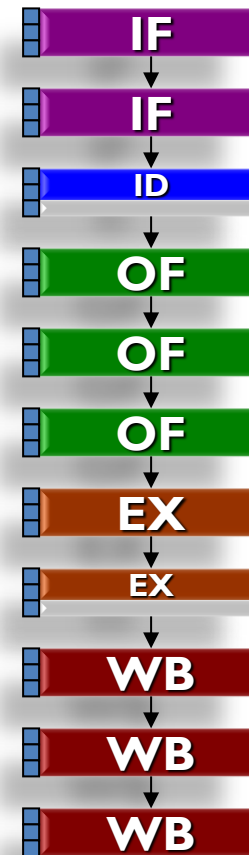
Coarser-Grained Machine Cycle:
4 machine cyc / instruction

stages = 4
 $T_{cyc} = 9$ units

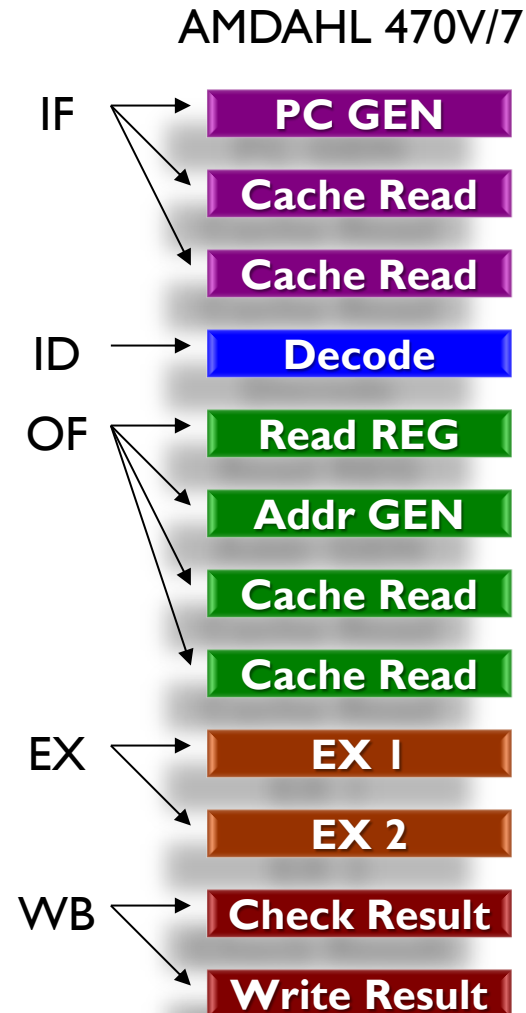
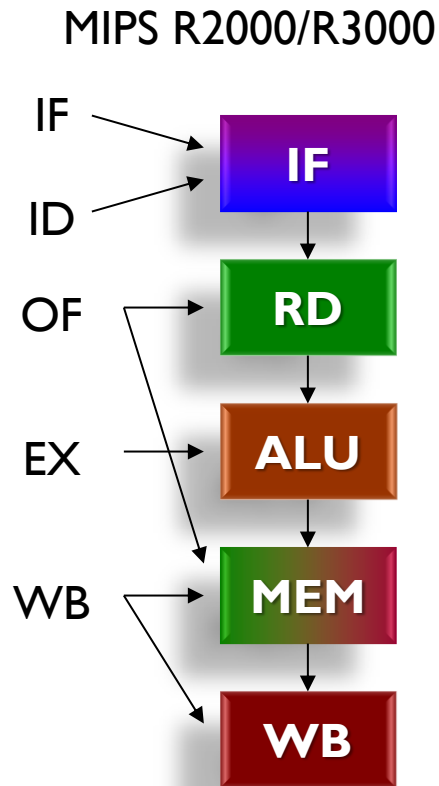


Finer-Grained Machine Cycle:
11 machine cyc / instruction

stages = 11
 $T_{cyc} = 3$ units



Pipeline Examples



Instruction Dependencies (1/2)

- Data Dependence
 - Read-After-Write (RAW) (only true dependence)
 - Read must wait until earlier write finishes
 - Anti-Dependence (WAR)
 - Write must wait until earlier read finishes (avoid clobbering)
 - Output Dependence (WAW)
 - Earlier write can't overwrite later write
- Control Dependence (a.k.a. Procedural Dependence)
 - Branch condition must execute before branch target
 - Instructions after branch cannot run before branch

Instruction Dependencies (1/2)

```
# for (;(j<high)&&(array[j]<array[low]);++j);
```

```

bge          j,  high, $36
mul          $15, j,    4
addu        $24, array, $15
lw          $25,  0($24)
mul          $13, low,  4
addu        $14, array, $13
lw          $15,  0($14)
bge          $25, $15, $36

```

\$35:

```

addu          j,  j,    1
...

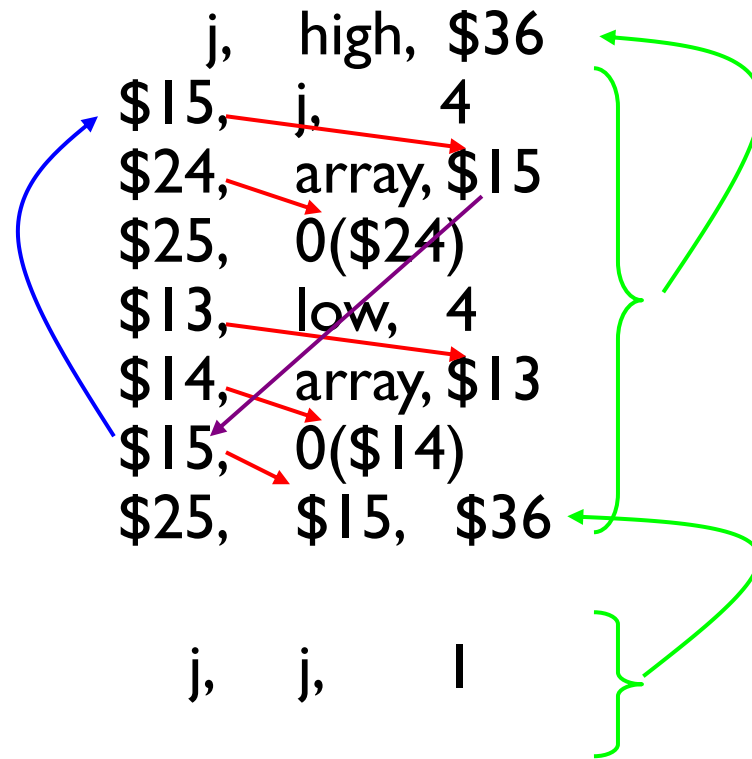
```

\$36:

```

addu          $11, $11, -1
...

```



Real code has lots of dependencies

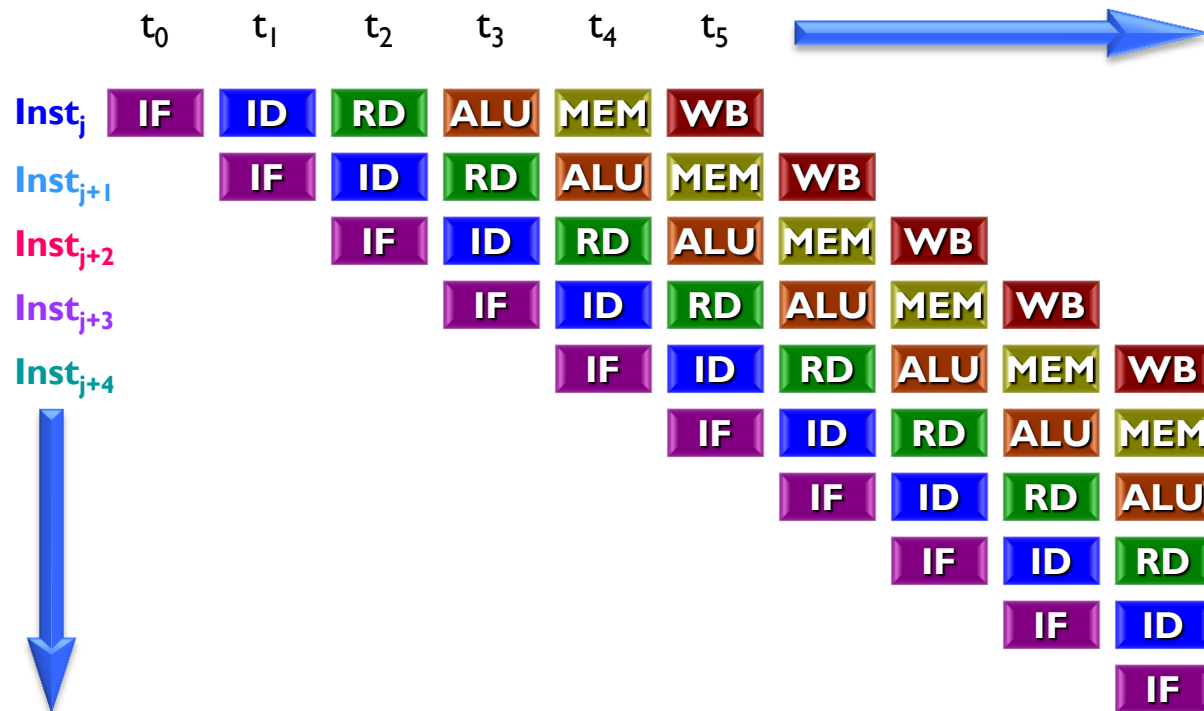
Hardware Dependency Analysis

- Processor must handle
 - Register Data Dependencies (same register)
 - RAW, WAW, WAR
 - Memory Data Dependencies (same address)
 - RAW, WAW, WAR
 - Control Dependencies

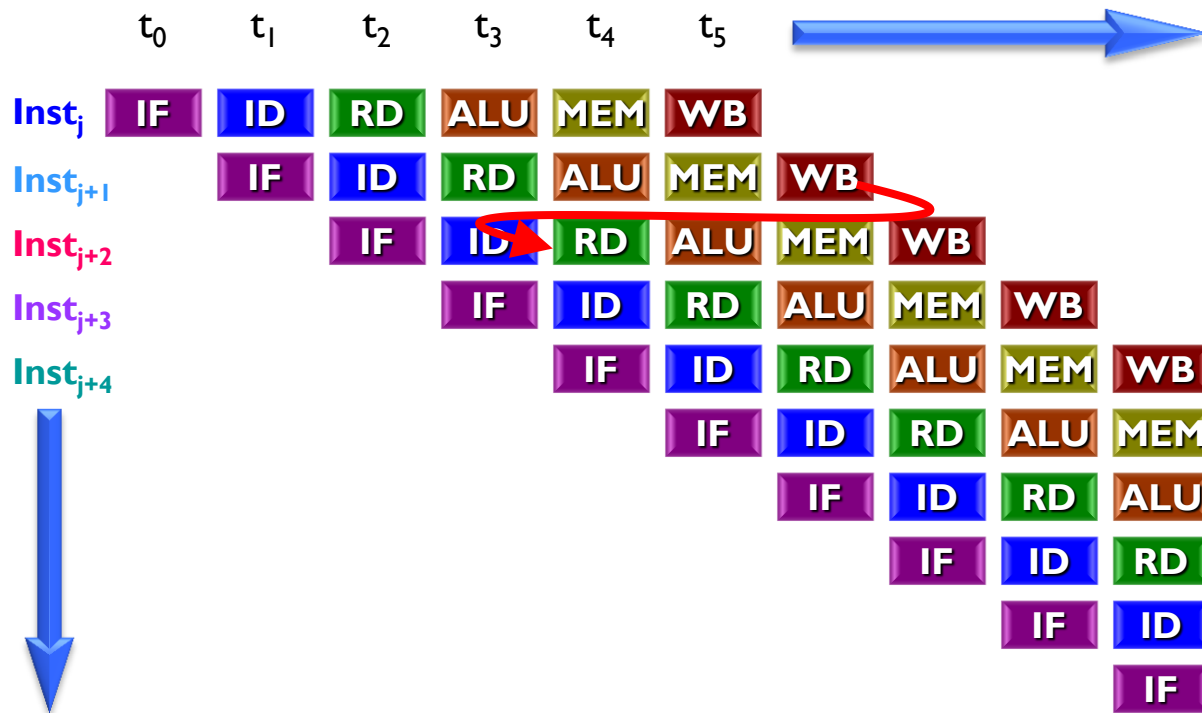
Pipeline Terminology

- Pipeline Hazards
 - Potential violations of program dependencies
 - Must ensure program dependencies are not violated
- Hazard Resolution
 - Static method: performed at compile time in software
 - Dynamic method: performed at runtime using hardware
 - Two options: Stall (costs perf.) or Forward (costs hw.)
- Pipeline Interlock
 - Hardware mechanism for dynamic hazard resolution
 - Must detect and enforce dependencies at runtime

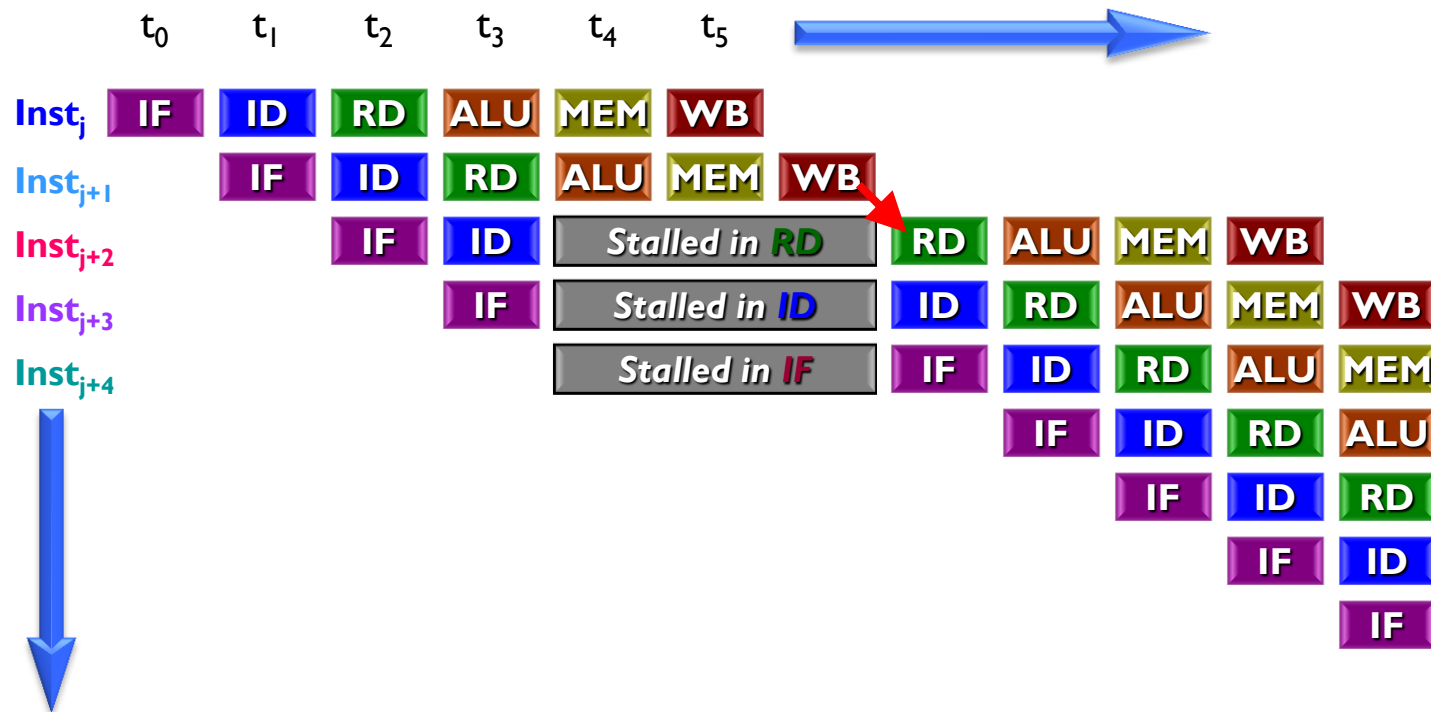
Pipeline: Steady State



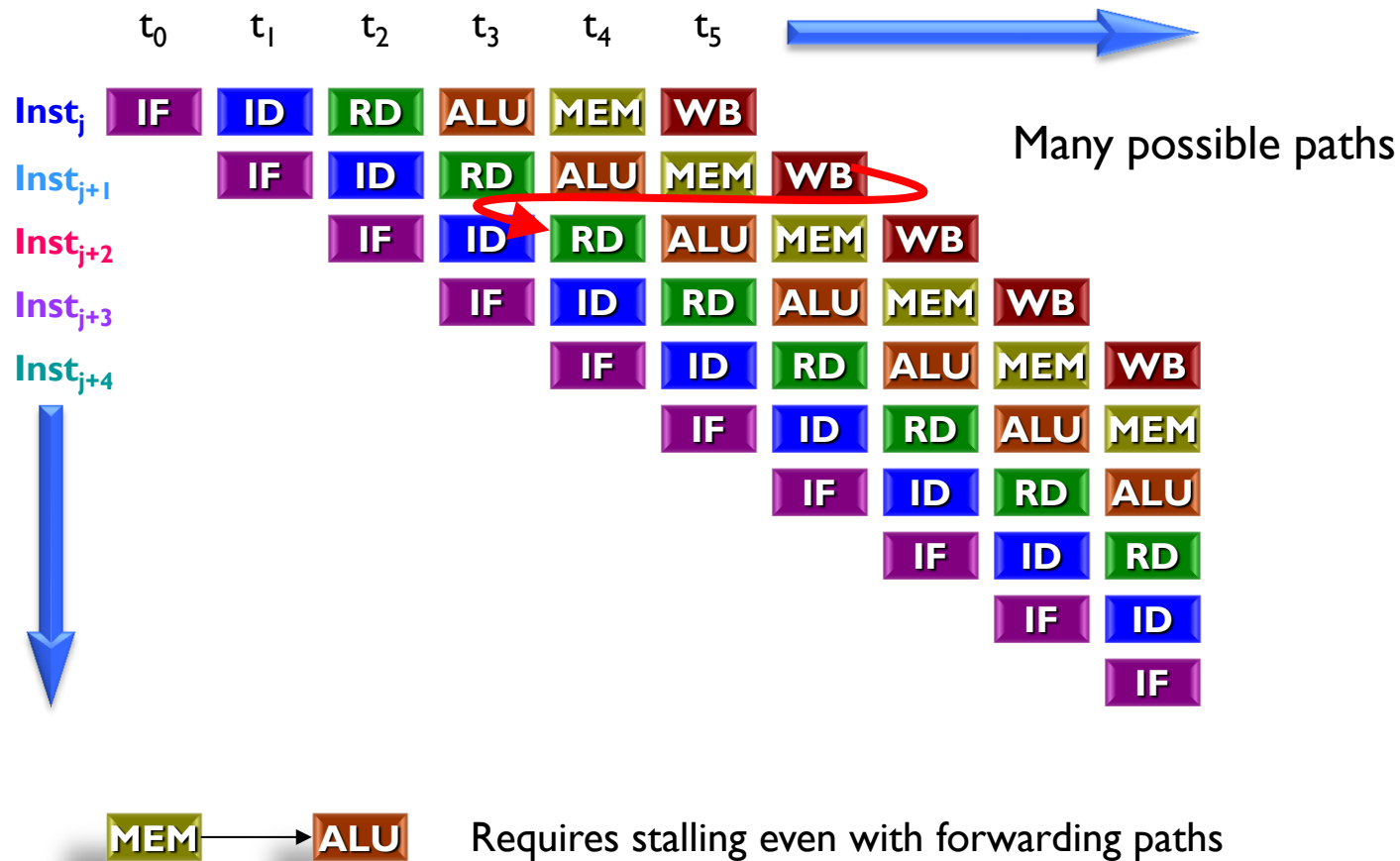
Pipeline: Data Hazard



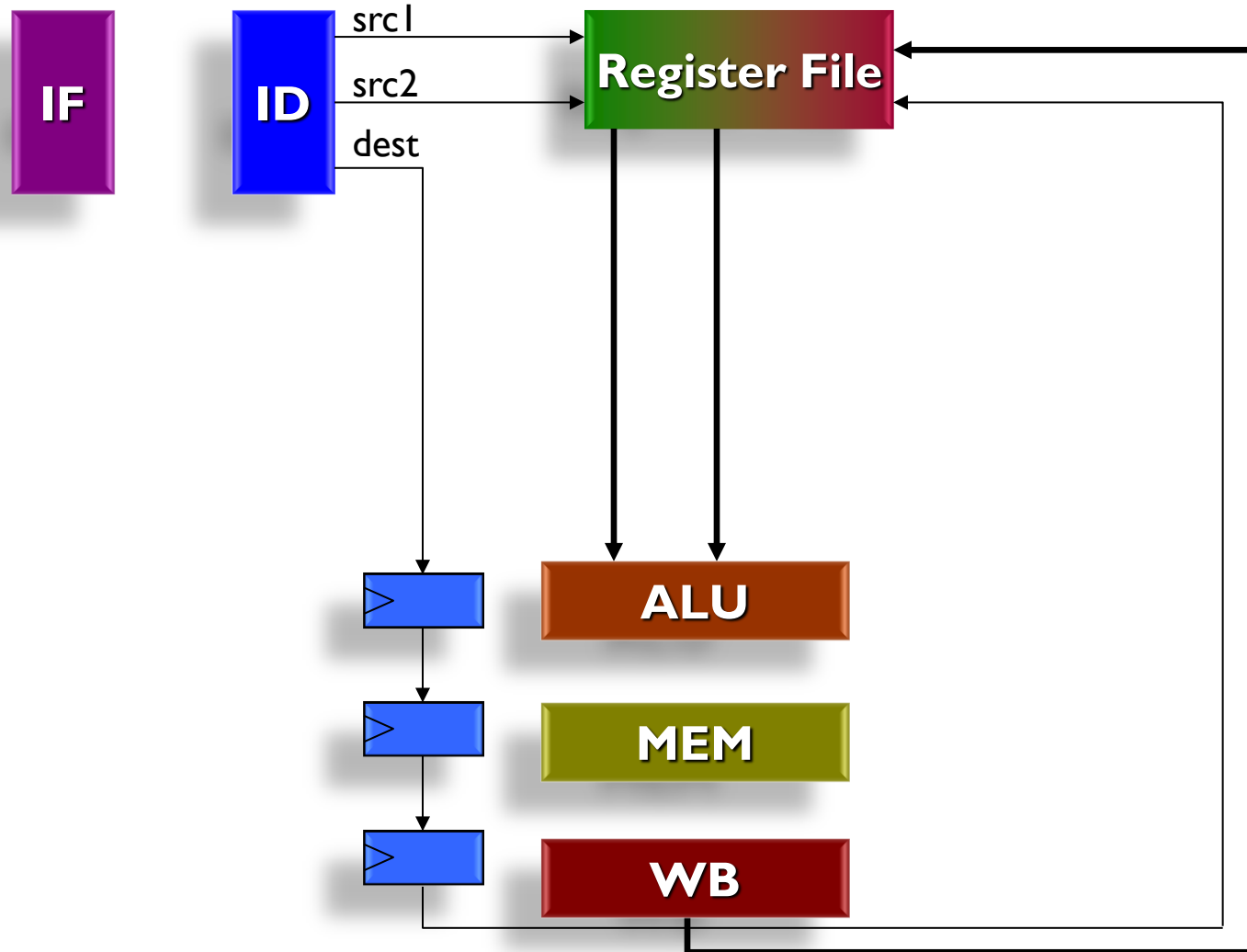
Option 1: Stall on Data Hazard



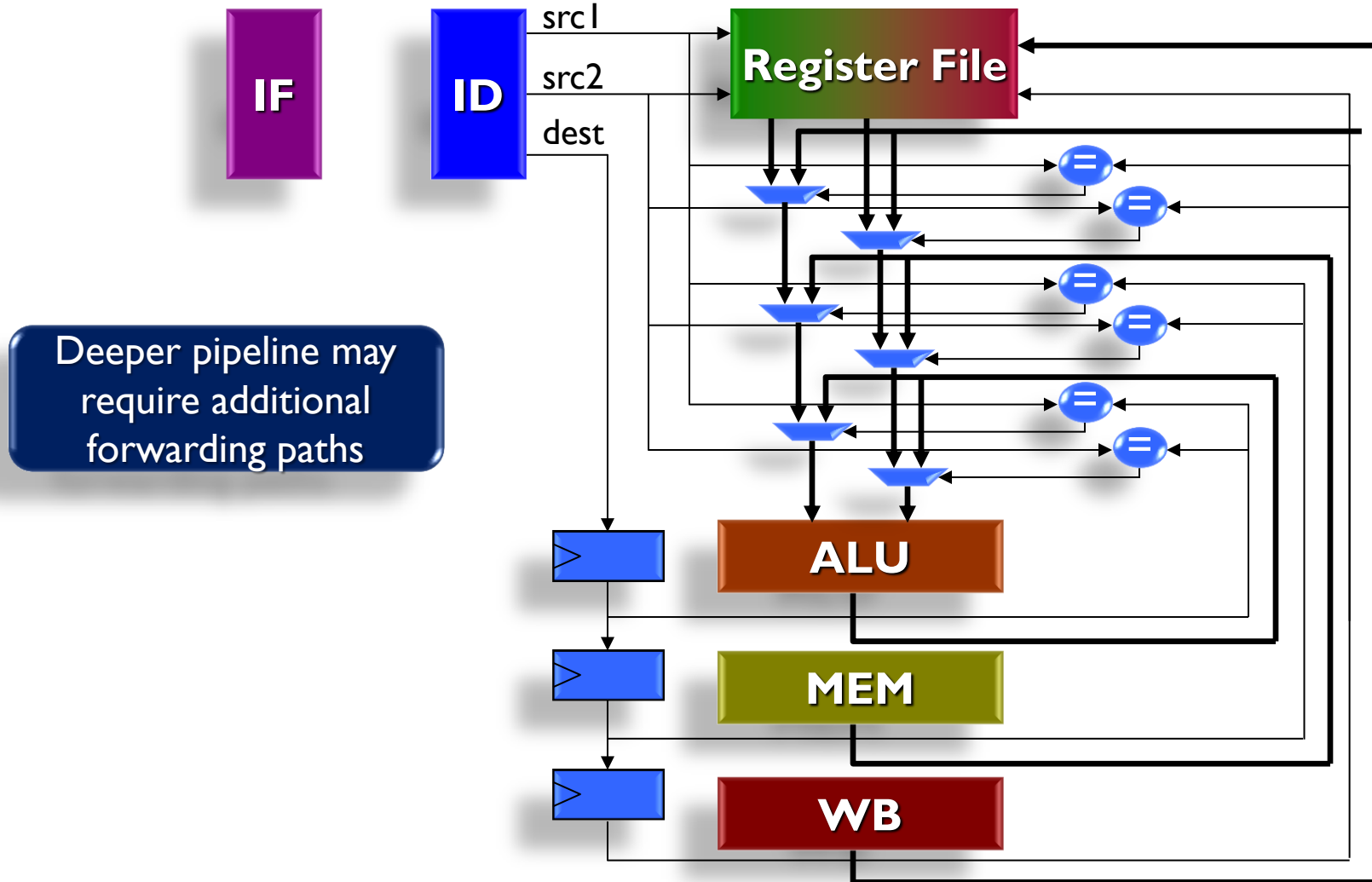
Option 2: Forwarding Paths (1/3)



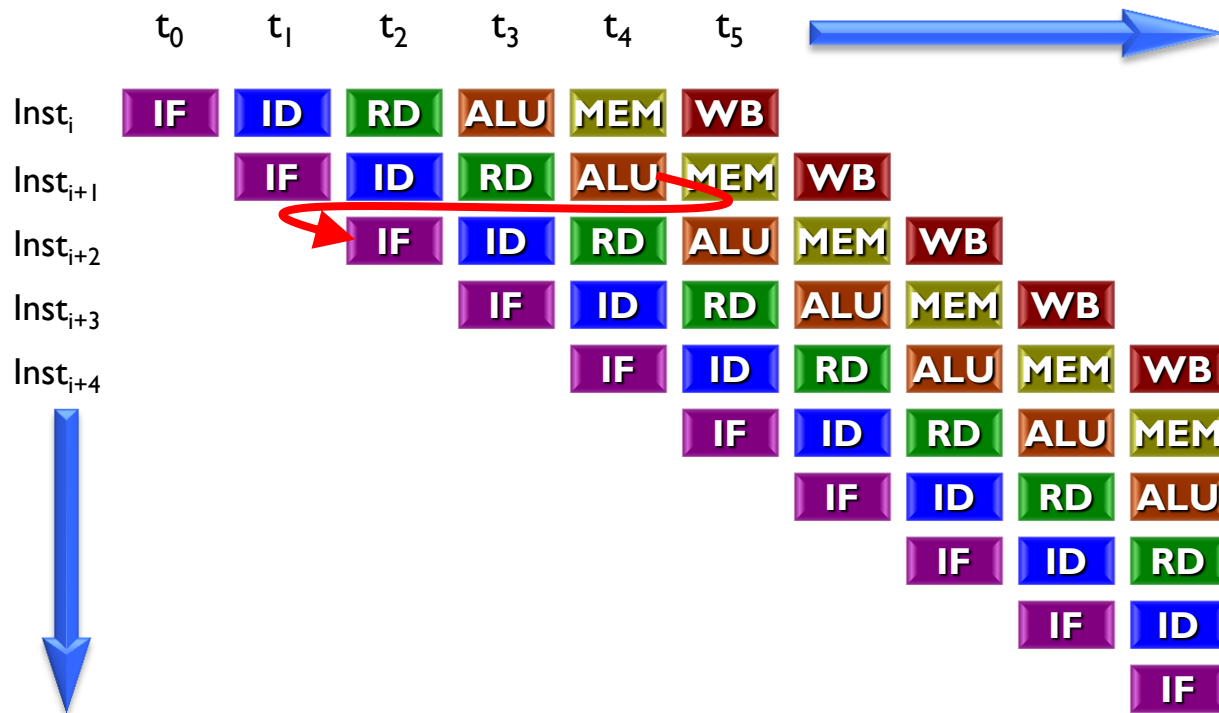
Option 2: Forwarding Paths (2/3)



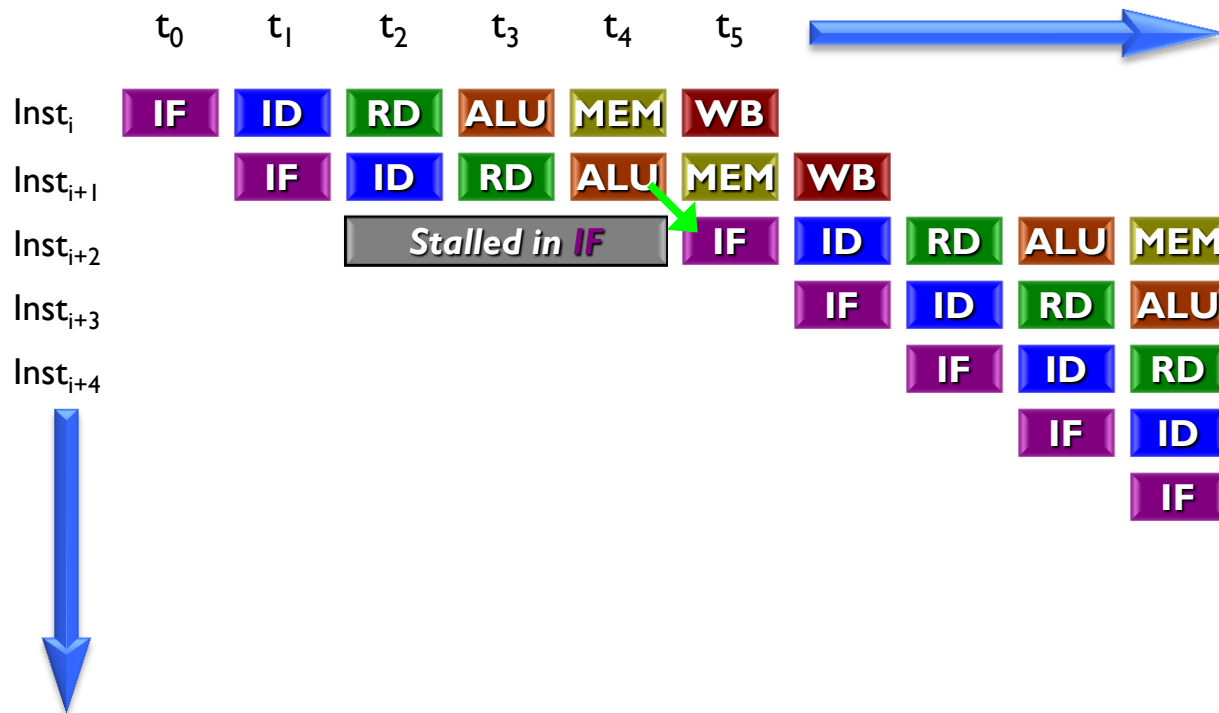
Option 2: Forwarding Paths (3/3)



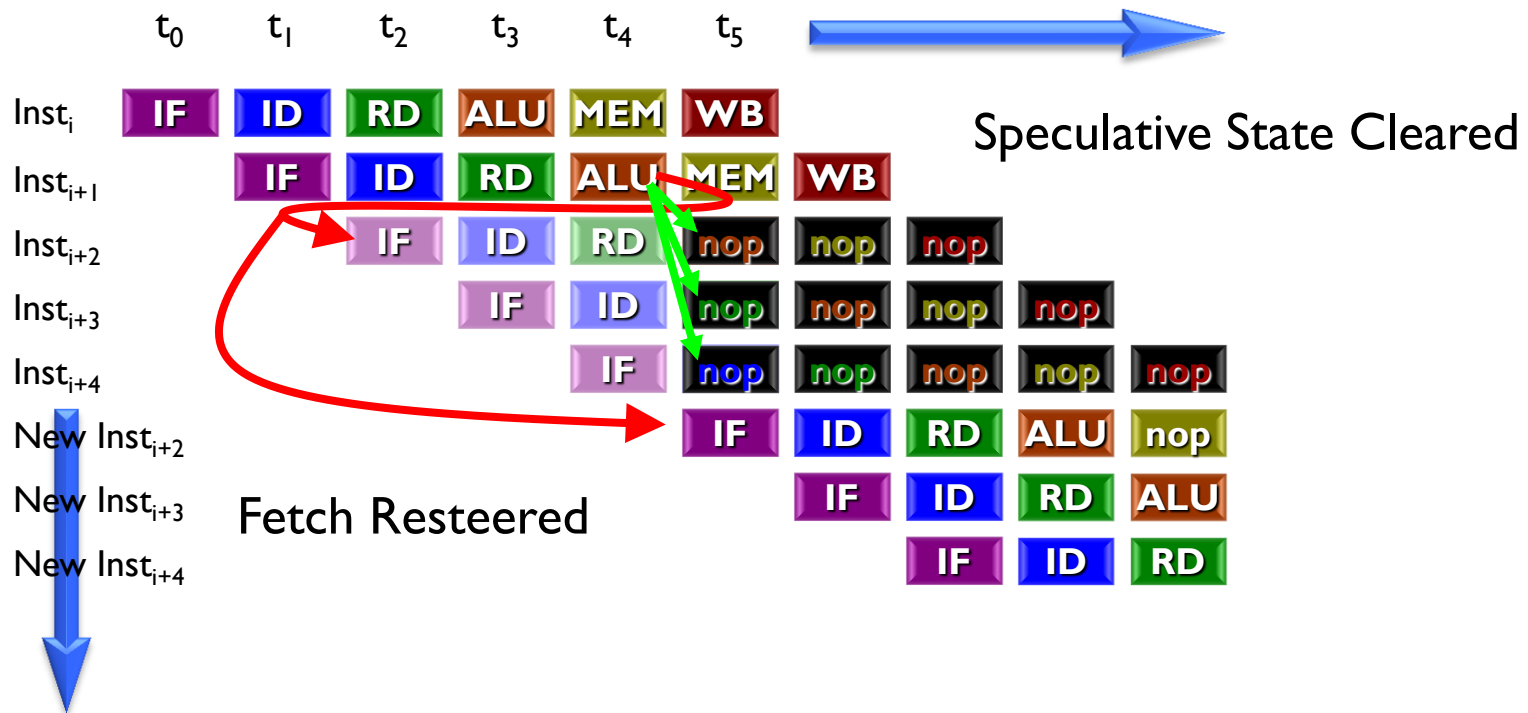
Pipeline: Control Hazard



Pipeline: Stall on Control Hazard




Pipeline: Prediction for Control Hazards

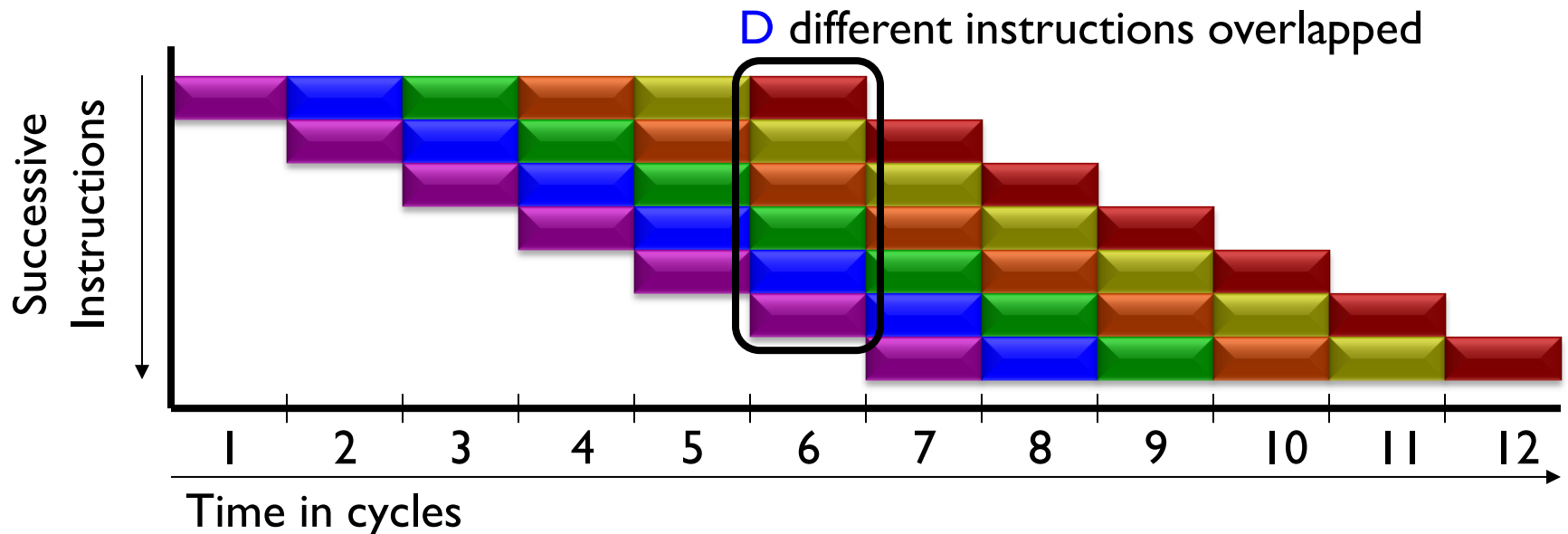


Going Beyond Scalar

- Scalar pipeline limited to $\text{CPI} \geq 1.0$
 - Can never run more than 1 insn. per cycle
- “Superscalar” can achieve $\text{CPI} \leq 1.0$ (i.e., $\text{IPC} \geq 1.0$)
 - Superscalar means executing multiple insns. in parallel

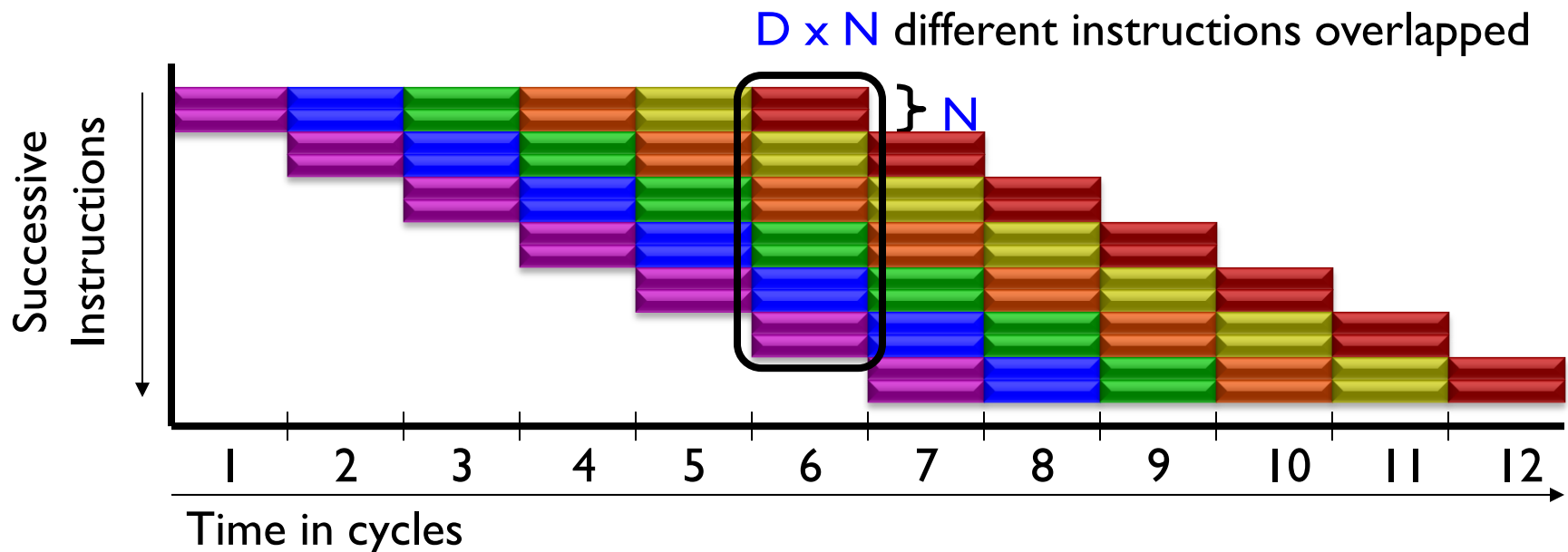
- Scalar pipeline (baseline)

- D
- 

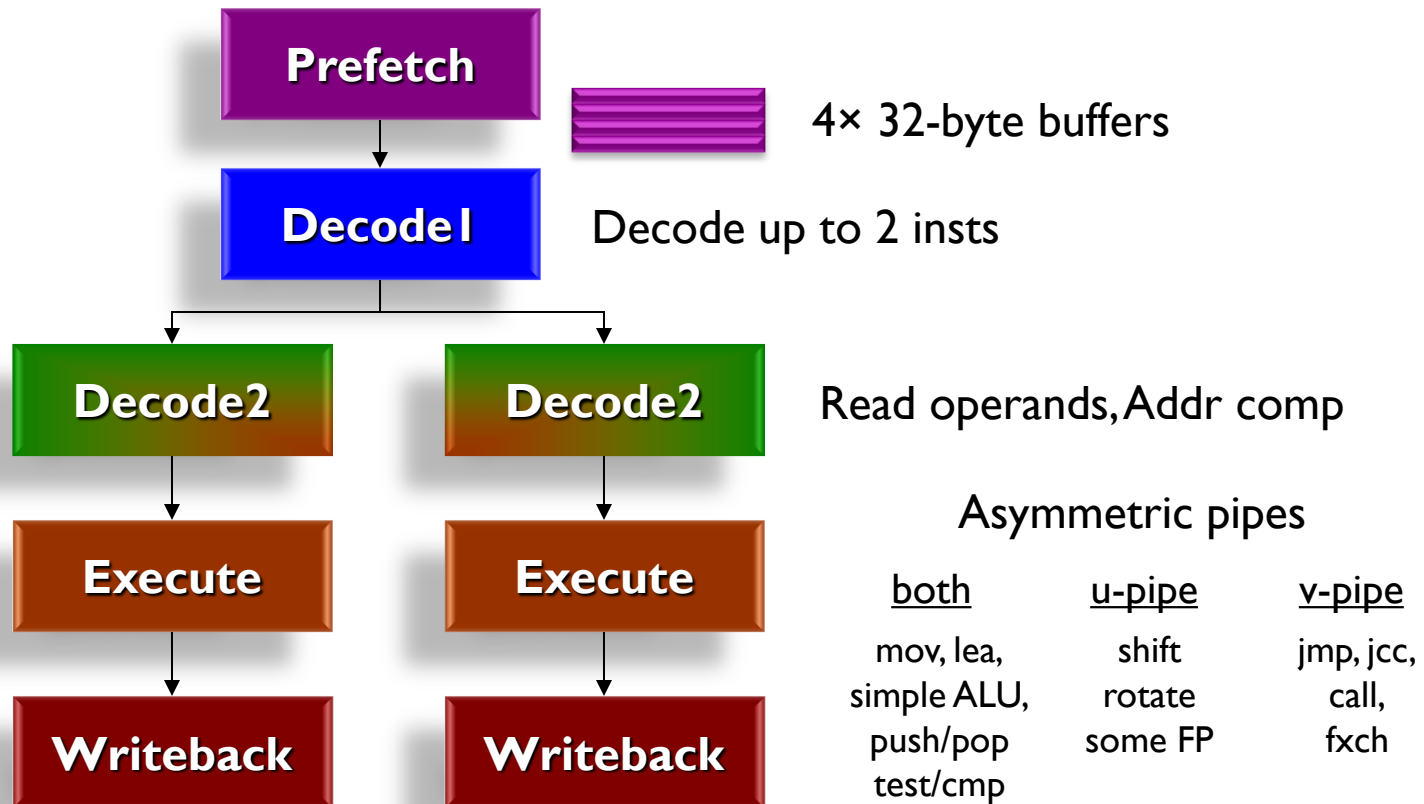


Superscalar Machine

- Superscalar (pipelined) Execution
 - Instruction parallelism = $D \times N$
 - Operation Latency = 1
 - Peak IPC = N per cycle



Superscalar Example: Pentium



Pentium Hazards & Stalls

- “Pairing Rules” (when can’t two insns exec?)
 - Read/flow dependence
 - `mov eax, 8`
 - `mov [ebp], eax`
 - Output dependence
 - `mov eax, 8`
 - `mov eax, [ebp]`
 - Partial register stalls
 - `mov al, 1`
 - `mov ah, 0`
 - Function unit rules
 - Some instructions can never be paired
 - `MUL`, `DIV`, `PUSHA`, `MOVS`, some FP

Limitations of In-Order Pipelines

- If the machine parallelism is increased
 - ... dependencies reduce performance
 - CPI of in-order pipelines degrades sharply
 - As N approaches avg. distance between dependent instructions
 - Forwarding is no longer effective
 - Must stall often

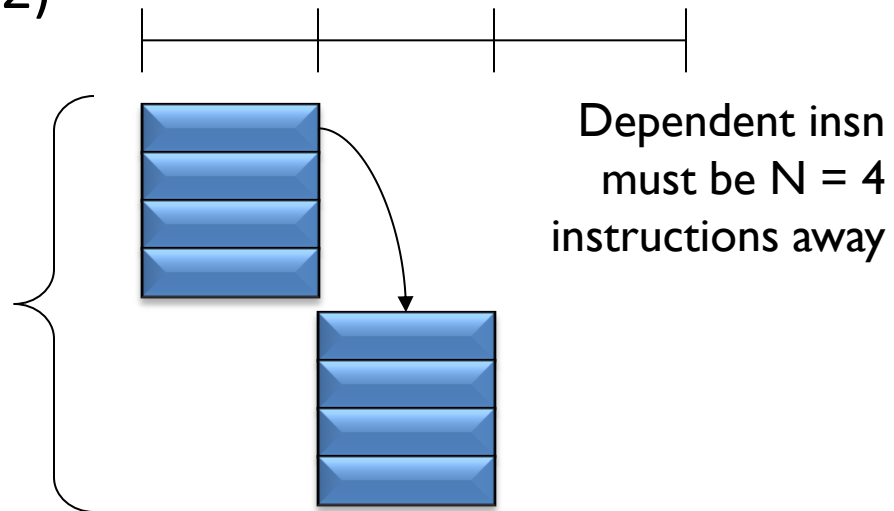
In-order pipelines are rarely full

The In-Order N-Instruction Limit

- On average, parent-child separation is about ± 5 insn.
– (Franklin and Sohi '92)

Ex. Superscalar degree $N = 4$

Any dependency
between these
instructions will
cause a stall



Average of 5 means there are many
cases when the separation is $< 4...$
each of these limits parallelism

Reasonable in-order superscalar is effectively $N=2$