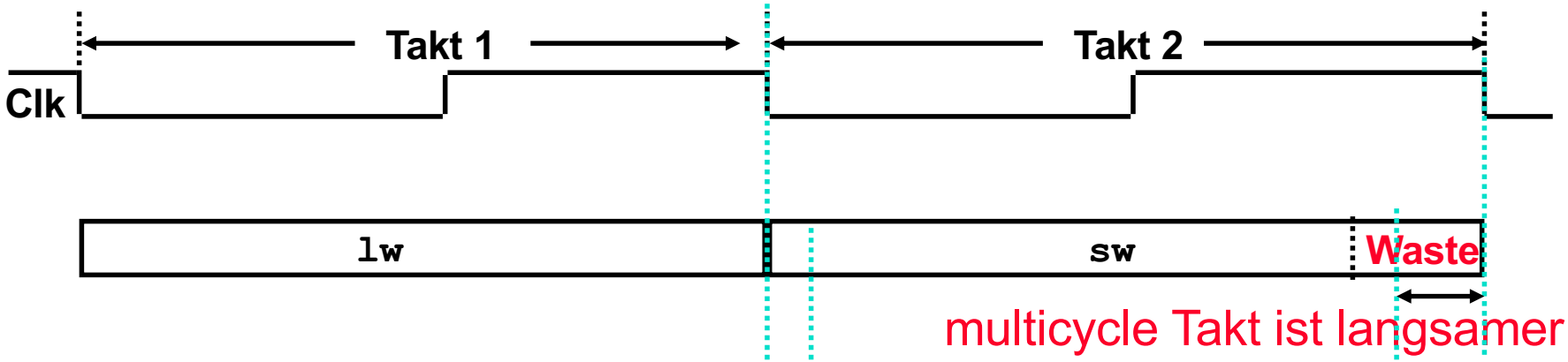
Basic MIPS Pipelining Review

[Adapted from Mary Jane Irwin for
Computer Organization and Design,
Patterson & Hennessy, © 2005, UCB]

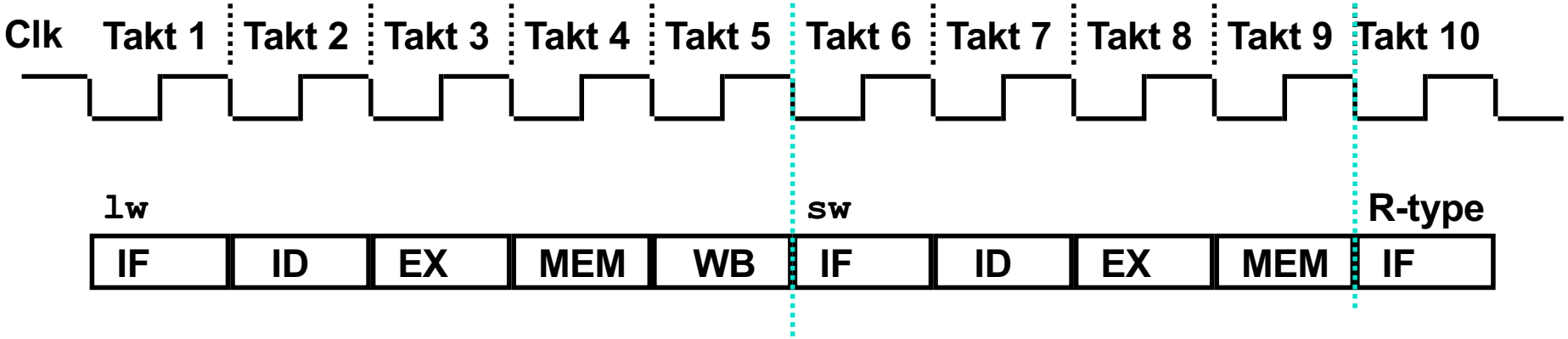
Review: Single Cycle vs. Multiple Cycle Timing

Single Cycle Implementation:



multicycle Takt ist langsamer als 1/5 des single cycle Takt aufgrund des overhead durch temporäre Register

Multiple Cycle Implementation:



Wie können wir noch schneller werden?

- Den Befehl in noch keine Schritte aufteilen
  - An einem Punkt wird gleichviel Zeit mit dem Ausführen der Befehle wie für das laden der Zustandsregister benötigt

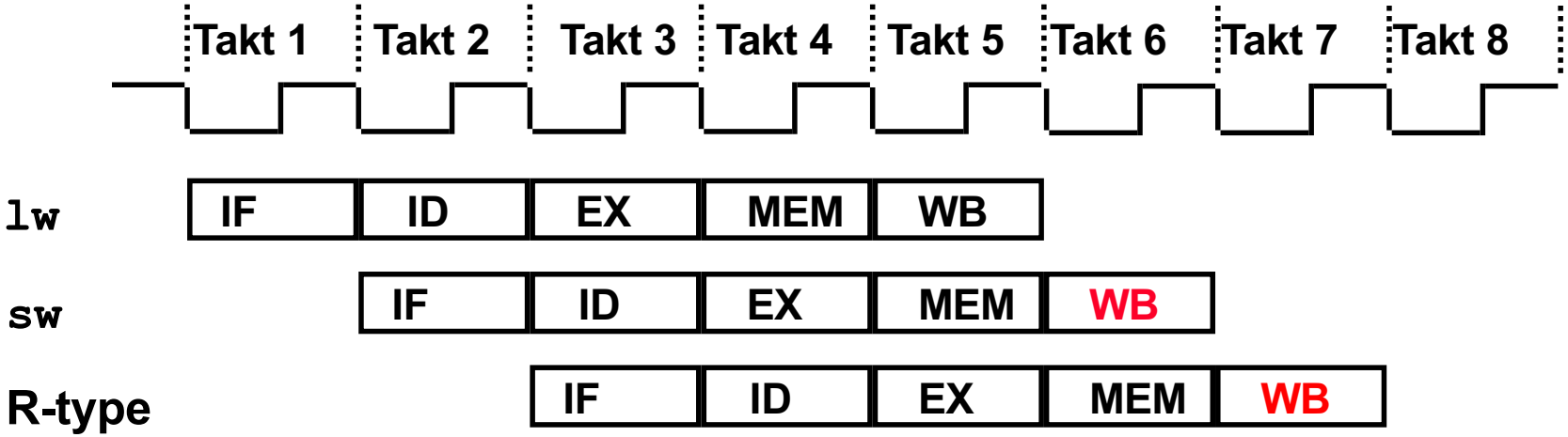
- Wir laden und führen den nächsten Befehl bereits aus, bevor der aktuelle Befehl komplettiert ist.
  - Pipelining – (alle?) moderne Prozessoren nutzen Pipelining für die Performance
  - Erinnerung an **DIE** Performance Gleichung:

    CPU time = CPI * Taktperiode * IC

- Fetch (und Execute) mehr als einen Befehl zur selben Zeit
  - Superskalare Prozessoren - (später in der Vorlesung)

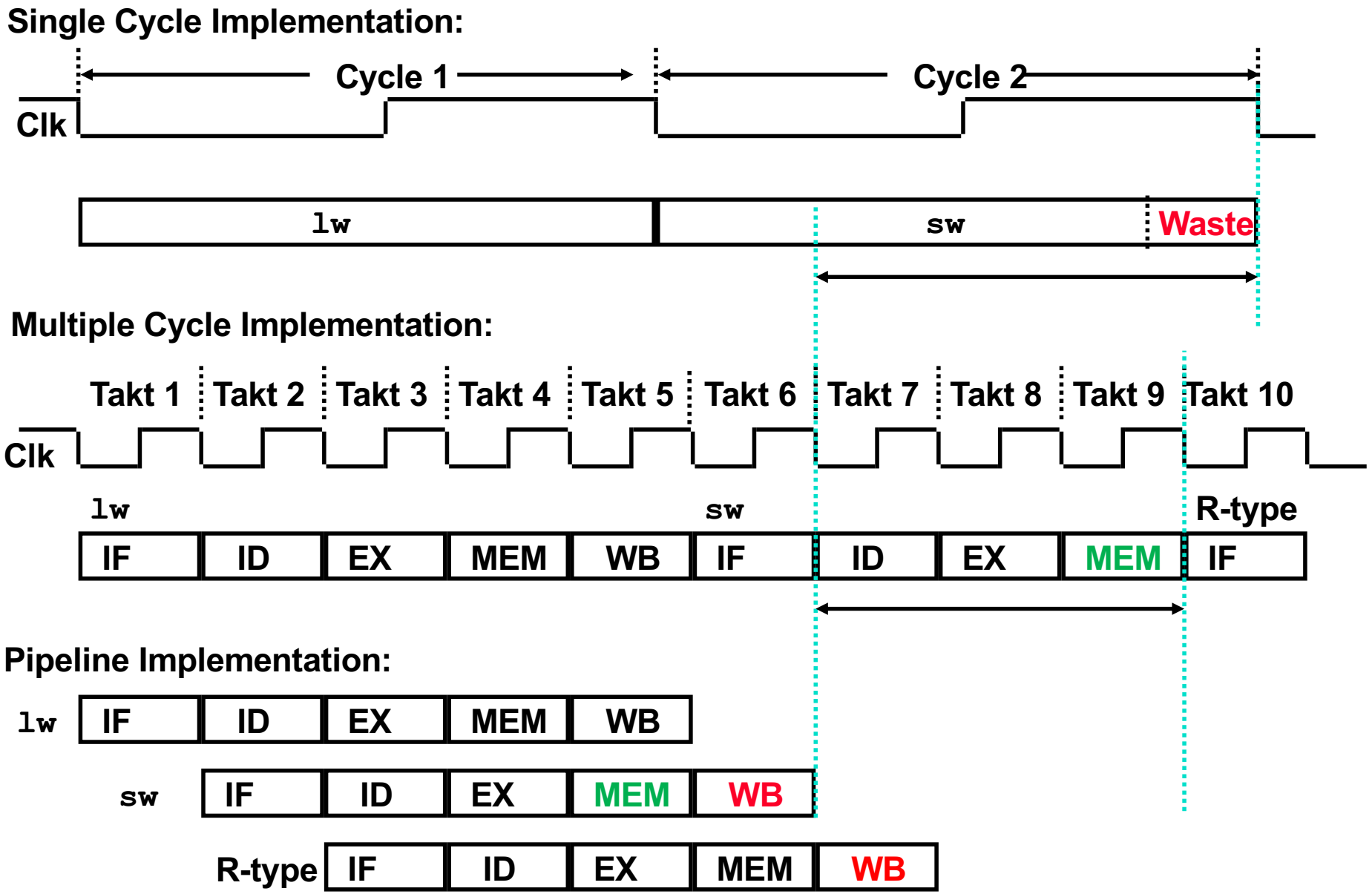
Ein Pipelined MIPS Prozessor

- Starte den nächsten Befehl bevor der aktuelle beendet ist
  - Verbessert Durchsatz - Menge Arbeit in gegebenem Zeitintervall
  - Befehls Latenzzeit (Ausführzeit, Delays, Antwortzeit – Zeit von Start bis zur Vervollständigung des Befehls) wird **nicht** reduziert



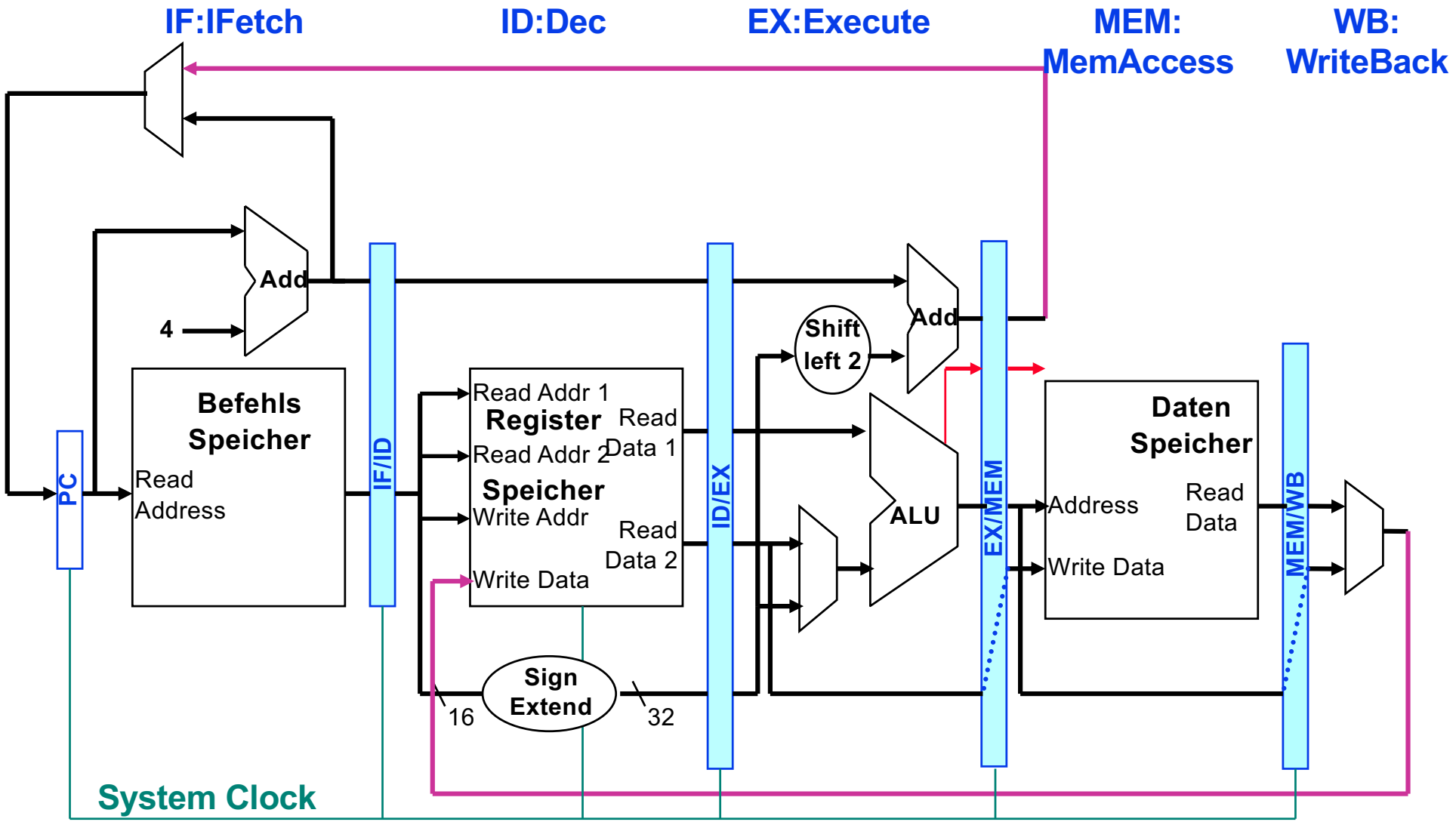
- Die Ausführzeit der Pipelinestufen werden durch die langsamste Ressource bestimmt. (ALU-Operation oder Speicherzugriffe)
- Für einige Befehle, werden einzelne Schritte verschwendet

Single Cycle, Multiple Cycle, vs. Pipeline



MIPS Pipeline Datenpfad Modifikationen

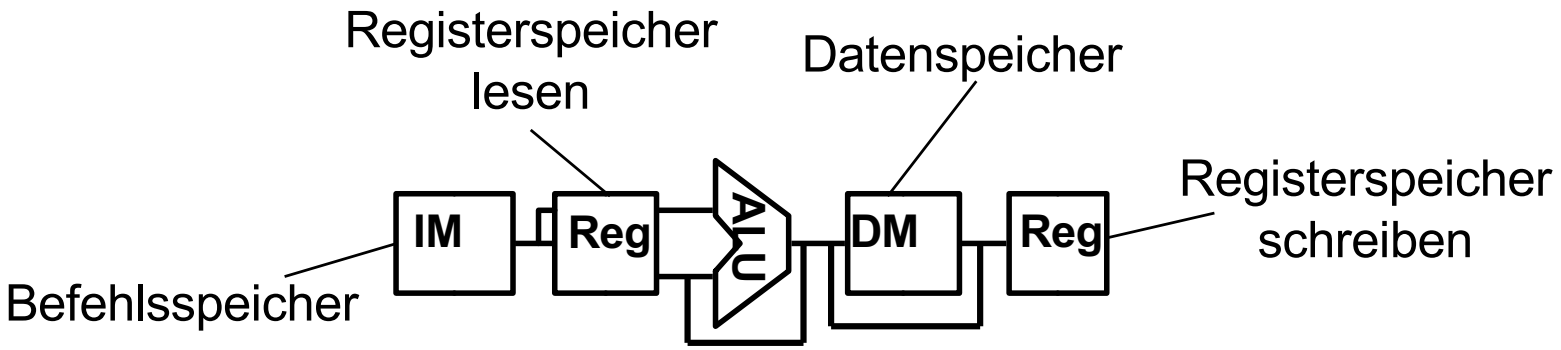
- Was müssen wir im MIPS Datenpfad hinzufügen/verändern?
  - Pipeline-Register trennen die Pipelinestufen um diese zu isolieren



Pipelining die MIPS Befehlssatzarchitektur (ISA)

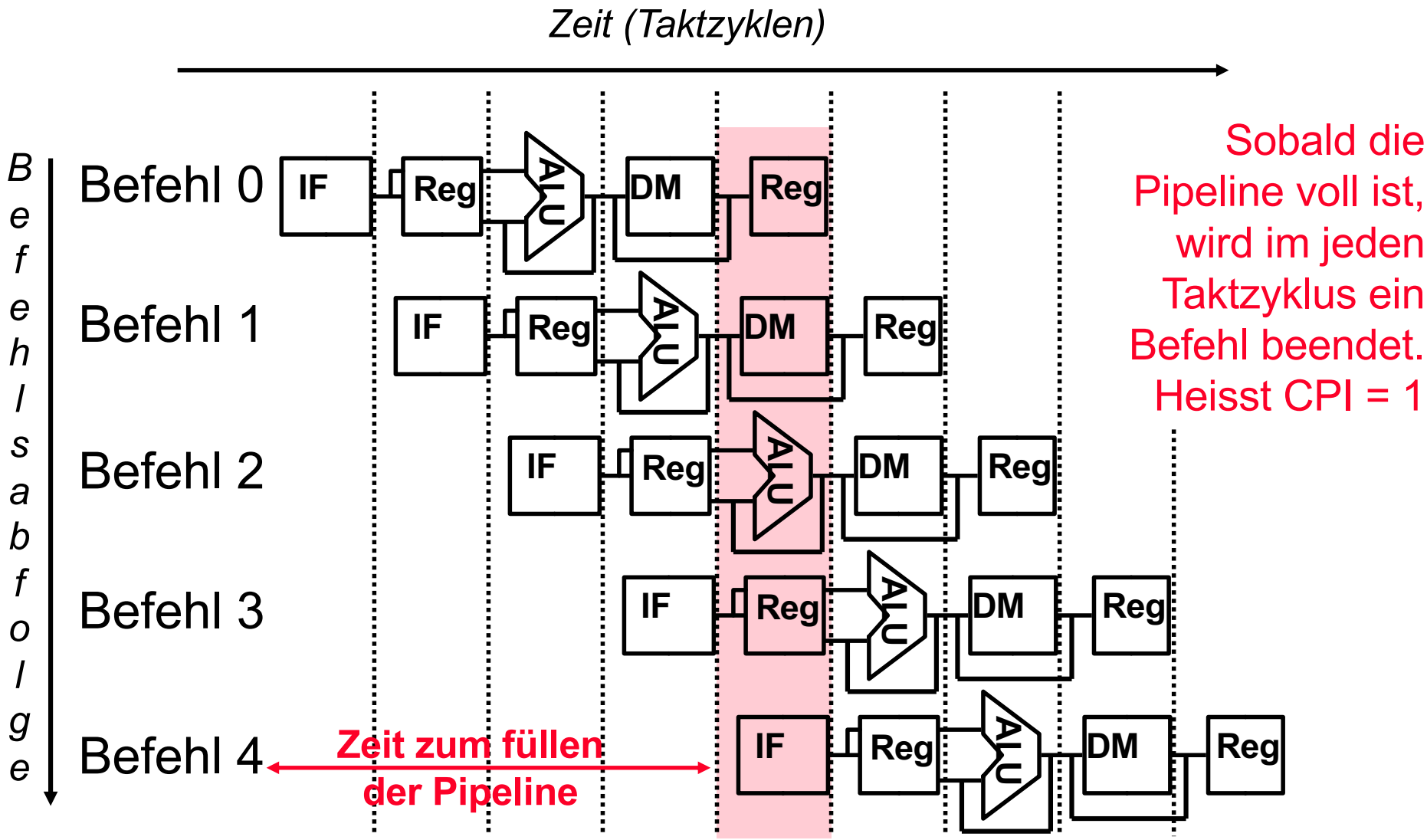
- Was ist einfach:
  - Alle Befehle haben die selbe Länge (32 Bits)
    - Lade in der 1. Stufe und decodieren in der 2. Stufe
  - Wenige Befehlsformate (drei) mit Symmetrien zwischen den Formaten
    - Kann in der 2. Stufe aus dem Registerspeicher lesen
  - Speicheroperationen kommen nur bei load und store Befehlen vor
    - Können die Execute Stufe zum berechnen von Adressen nutzen
  - Jeder MIPS Befehl schreibt max. ein Resultat und das geschieht in der Nähe des Ende der Pipeline (MEM & WB)
- Was ist schwierig:
  - Strukturkonflikte (structural hazards): Was wenn wir nur einen Speicher hätten?
  - Steuerkonflikte (control hazards): Was ist mit Sprüngen?
  - Datenkonflikte (data hazards): Was wenn der Input Operand eines Befehls vom Output des letzten Befehls abhängt?

Graphische Darstellung der MIPS Pipeline



- Kann helfen beim beantworten von Fragen wie:
  - Wie viele Taktzyklen werden für ein Stück Code benötigt?
  - Was macht die ALU im Taktzyklus 4?
  - Ist da ein Konflikt (Hazard)? Wenn ja, Warum?
    Wie kann er behoben werden?

Warum Pipeline? Für Performance!

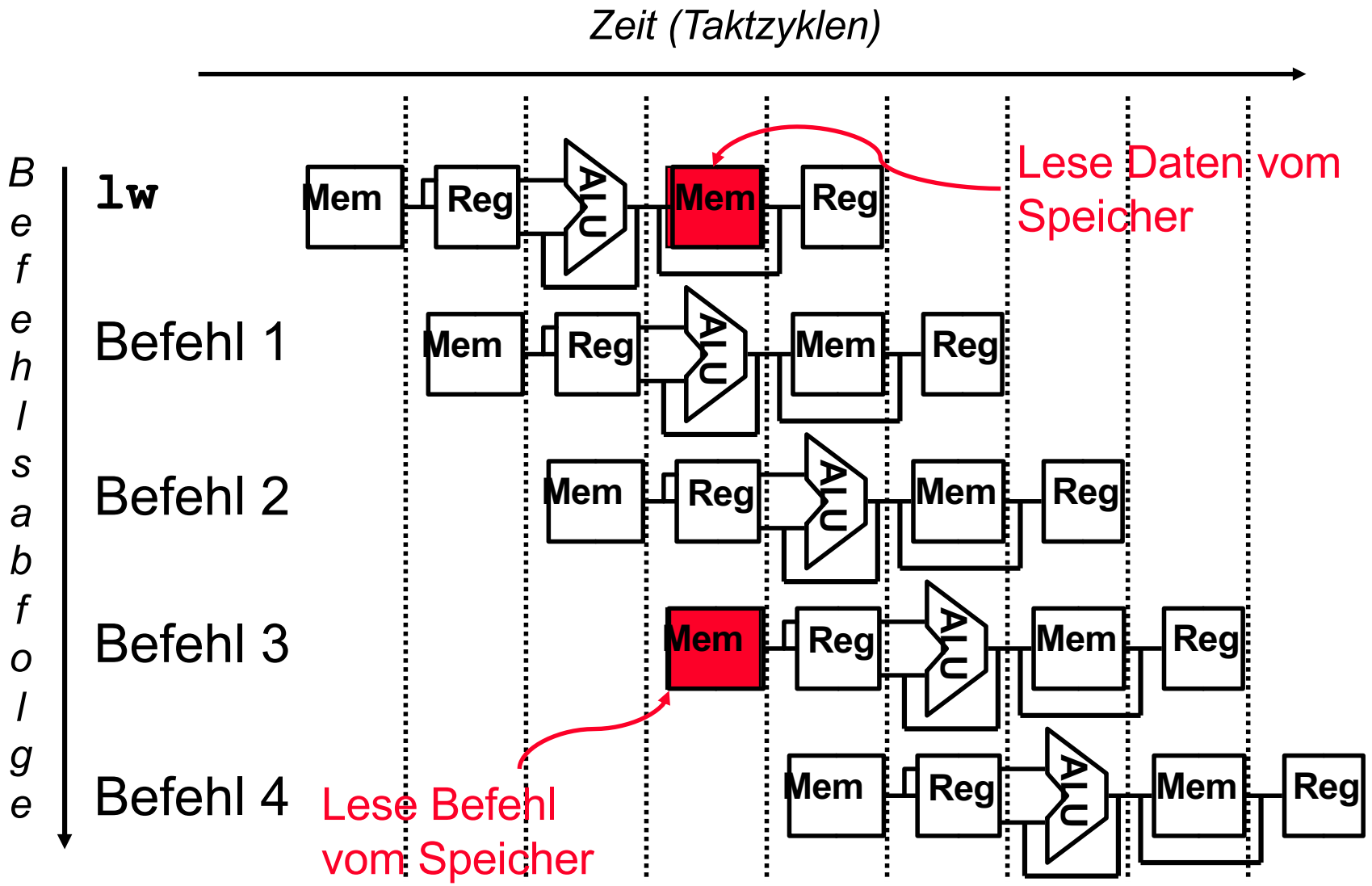


Kann uns Pipelining Probleme machen?

- Ja: Pipeline Konflikte
  - Strukturkonflikte: Versuch von zwei Befehlen gleichzeitig dieselbe Ressource zu benutzen
  - Datenkonflikte: Versuch Datenwert zu nutzen bevor er bereitsteht
    - Das Resultat eines Befehls wird von einem nachfolgenden Befehl benötigt bevor dieser fertig bereit steht
  - Steuerkonflikte: Versuch eine Entscheidung über den Programm-fluss zu treffen bevor die Bedingung ausgewertet wurde und der PC die neue Adresse enthält
    - Sprungbefehle

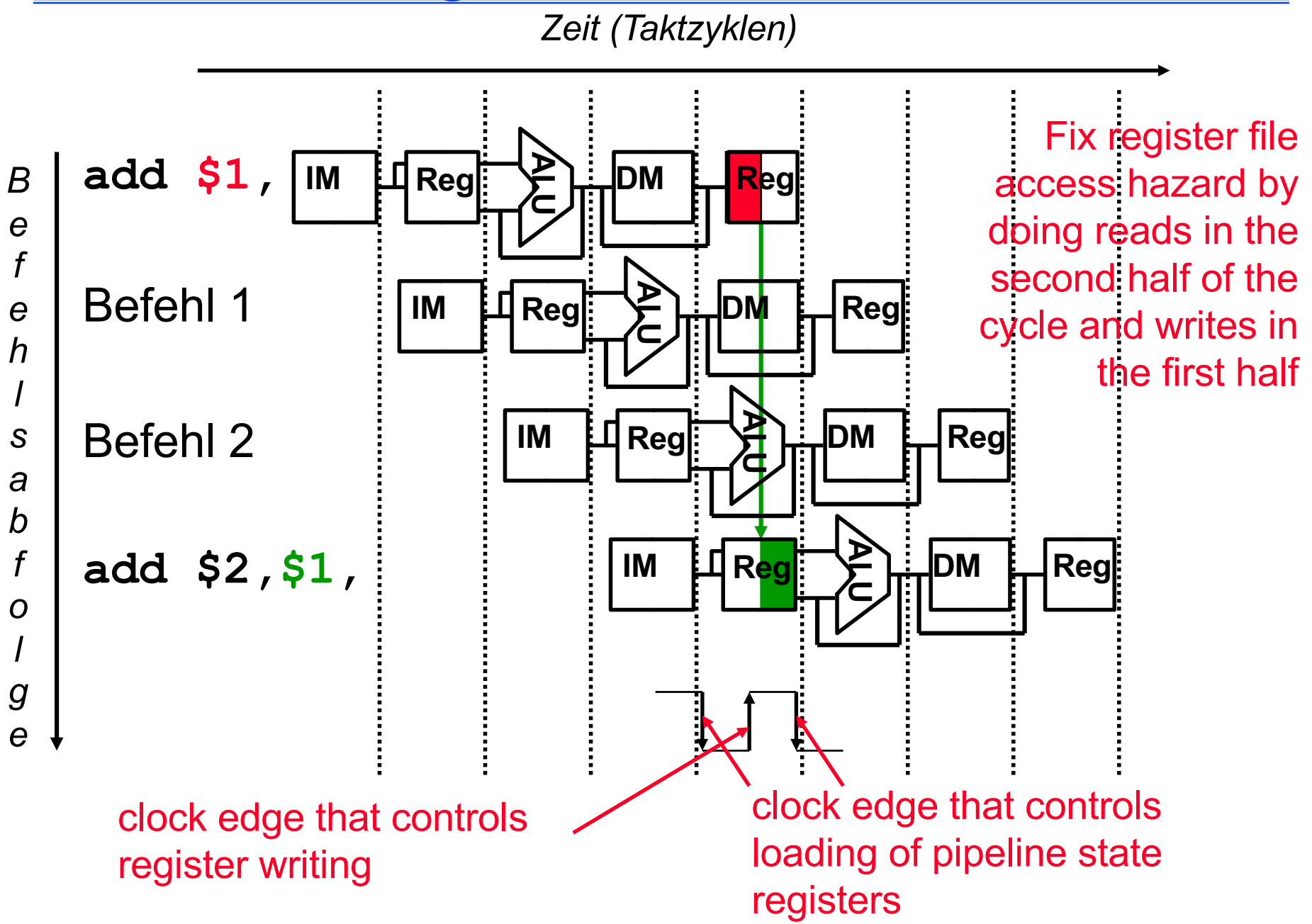
- Können Konflikte immer durch warten lösen
  - Pipeline Steuerung muss Konflikte erkennen
  - Pipeline Steuerung muss Konflikte auflösen

Ein einzelner Speicher wäre ein Strukturkonflikt



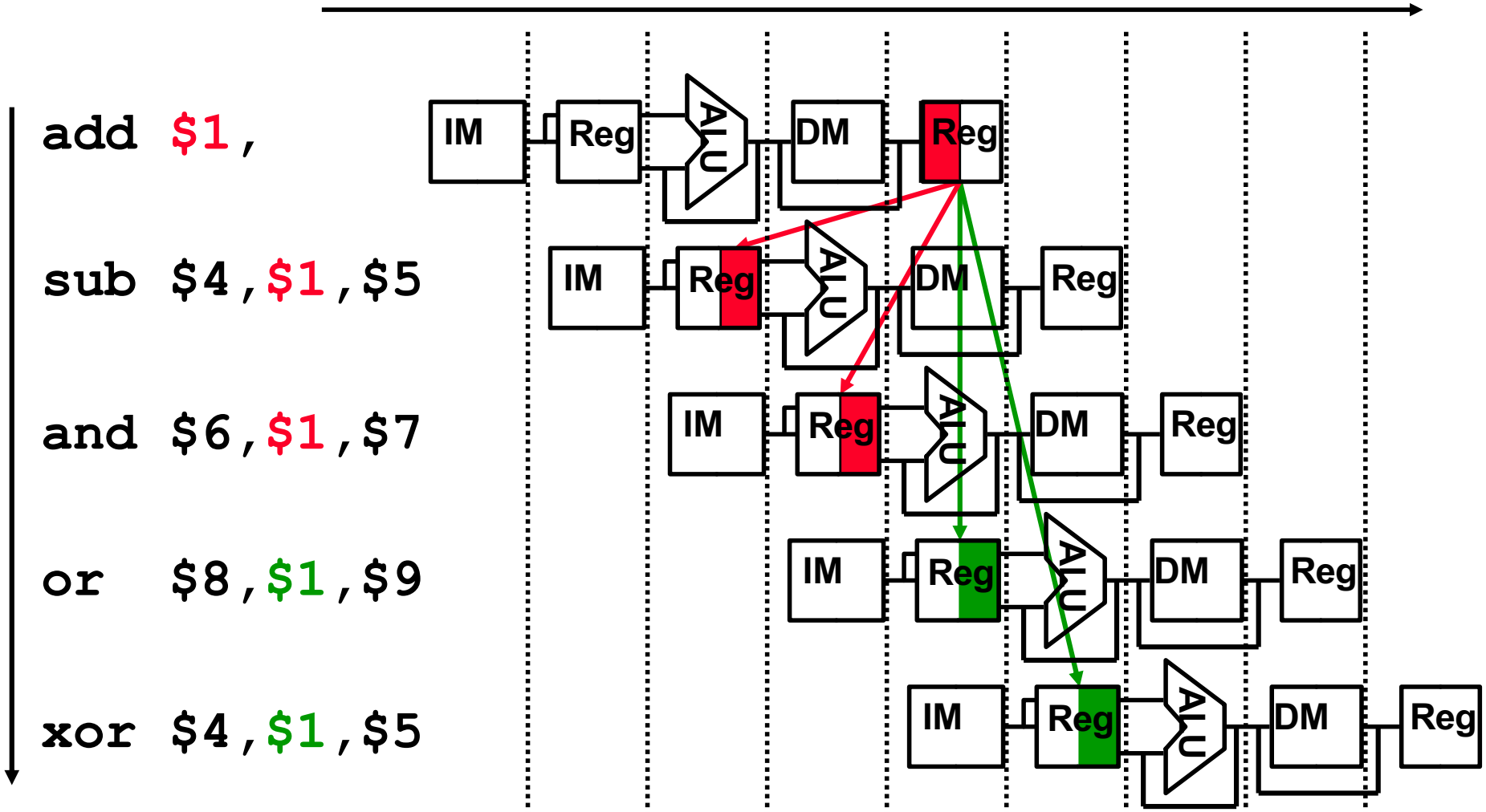
- Daher separate Speicher für Daten und Befehle (I$, D$)

How About Register File Access?



Register Usage Can Cause Data Hazards

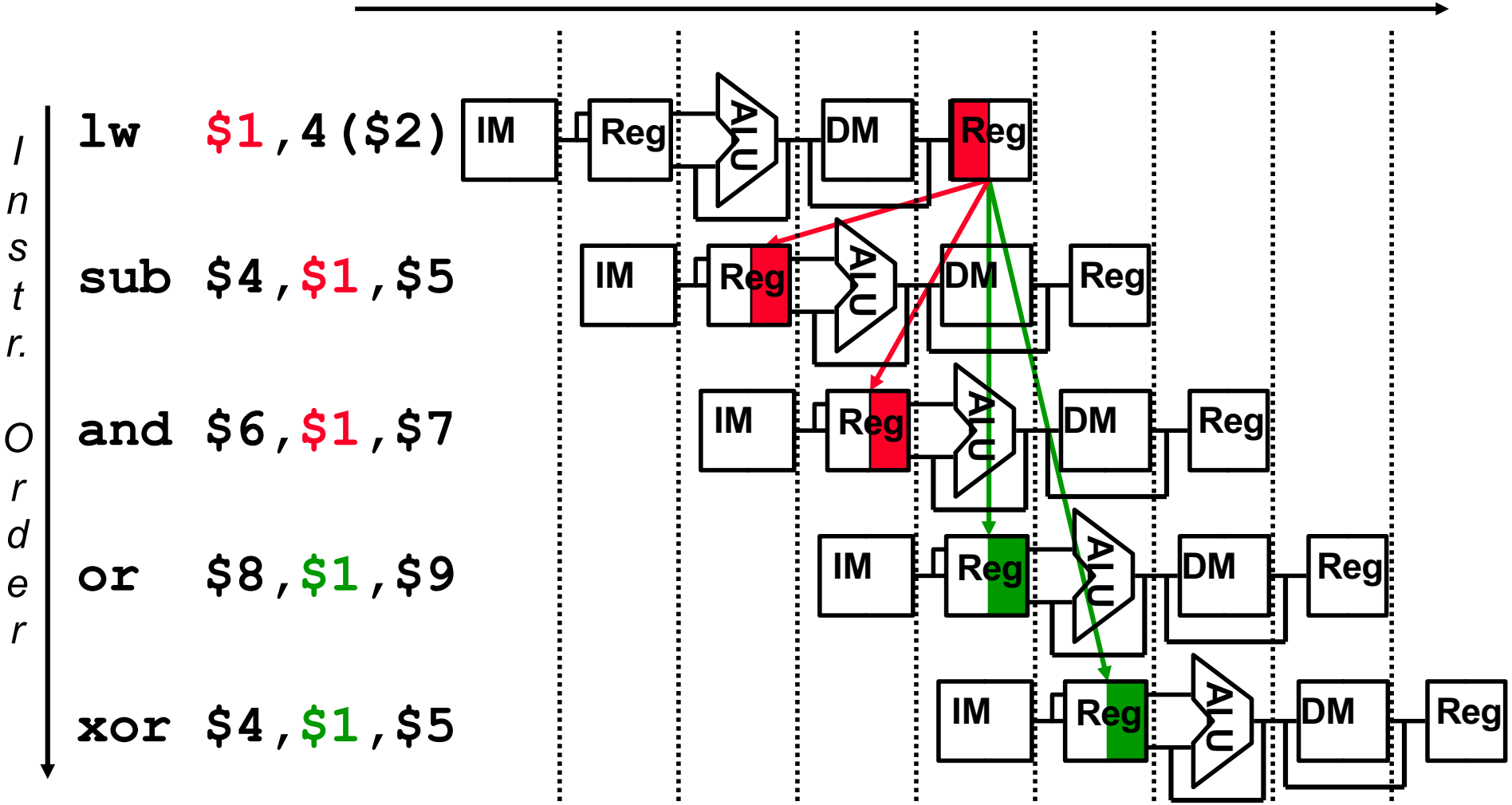
- Dependencies backward in time cause hazards



- Read before write data hazard

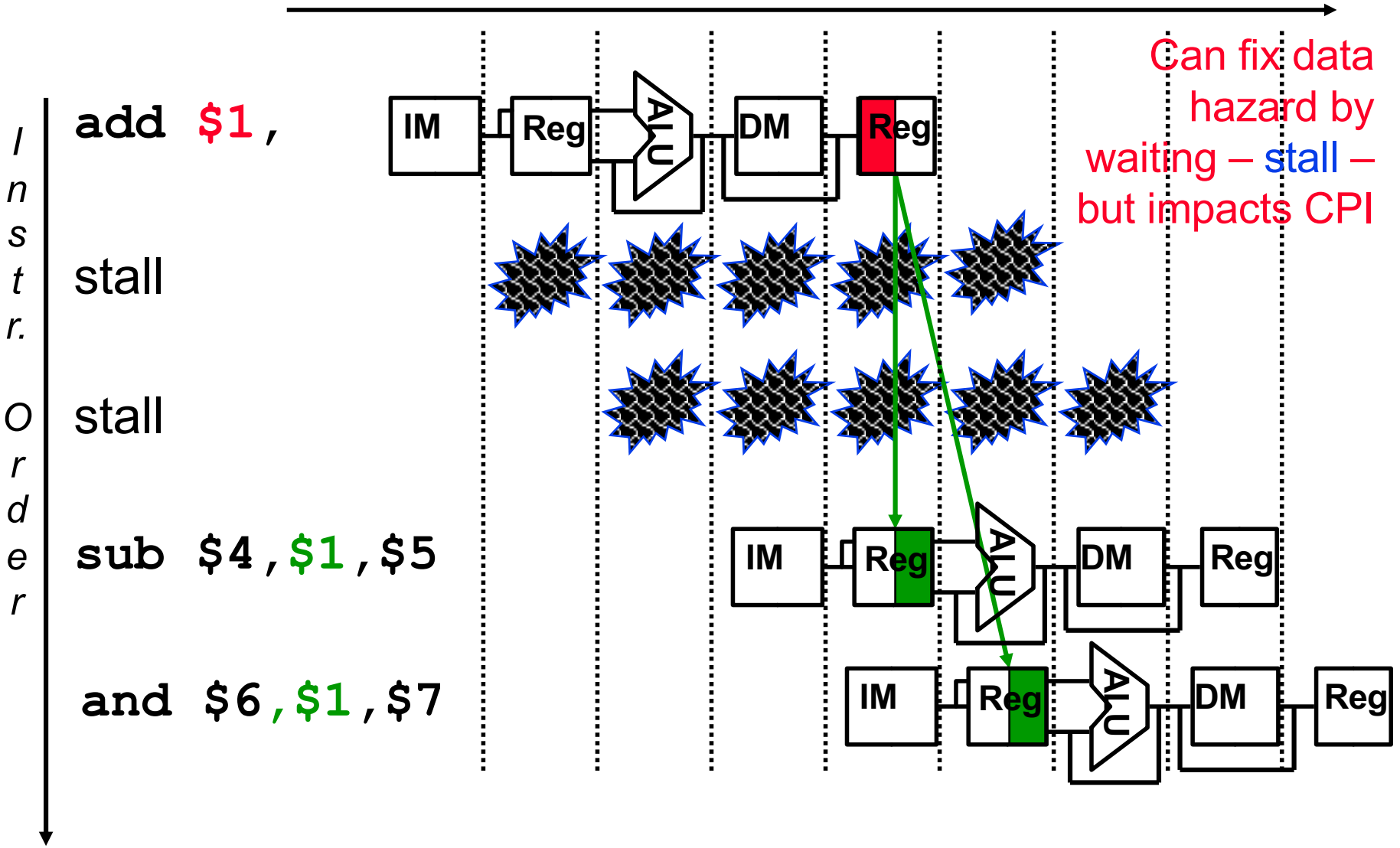
Loads Can Cause Data Hazards

- Dependencies backward in time cause hazards

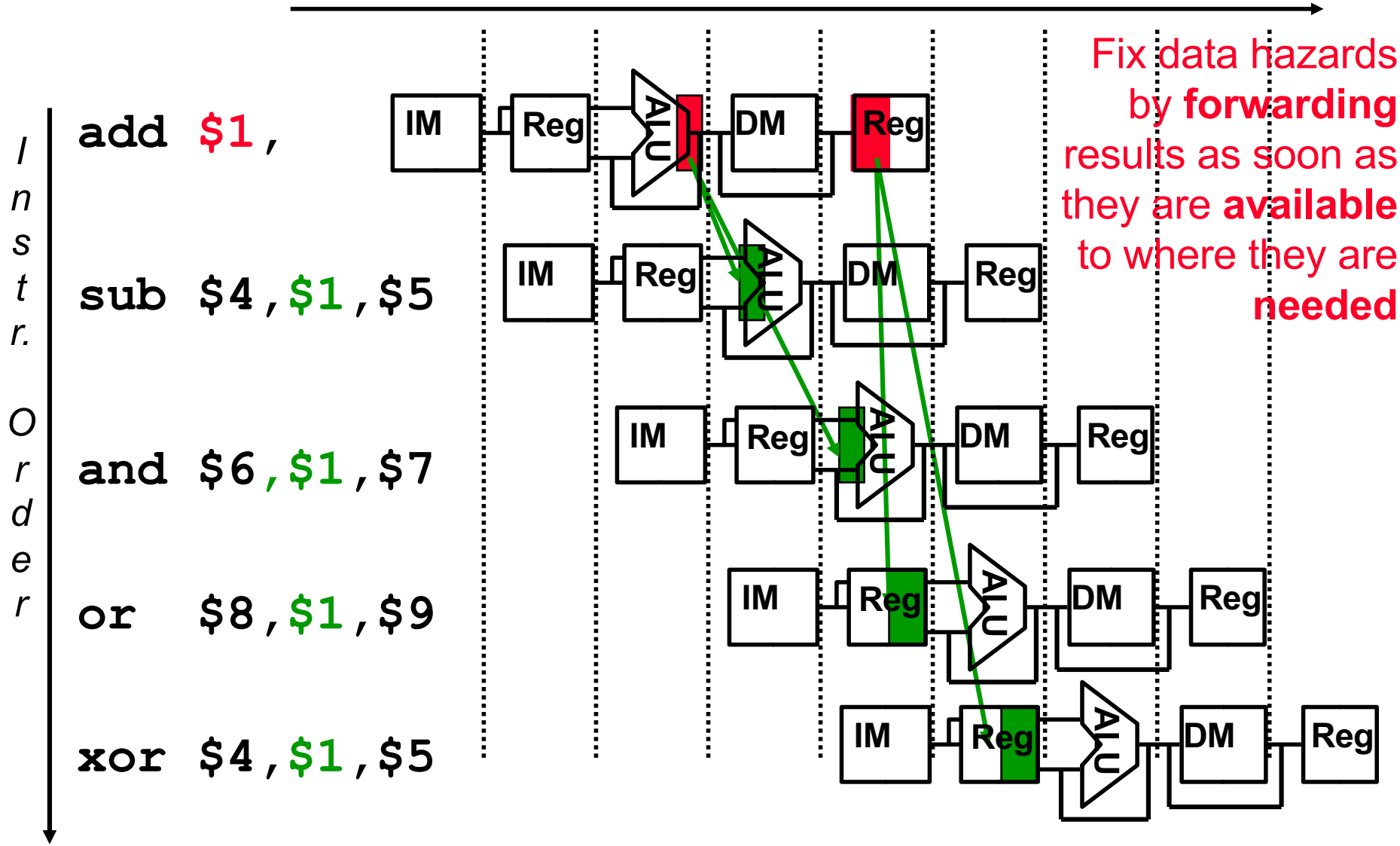


- Load-use data hazard

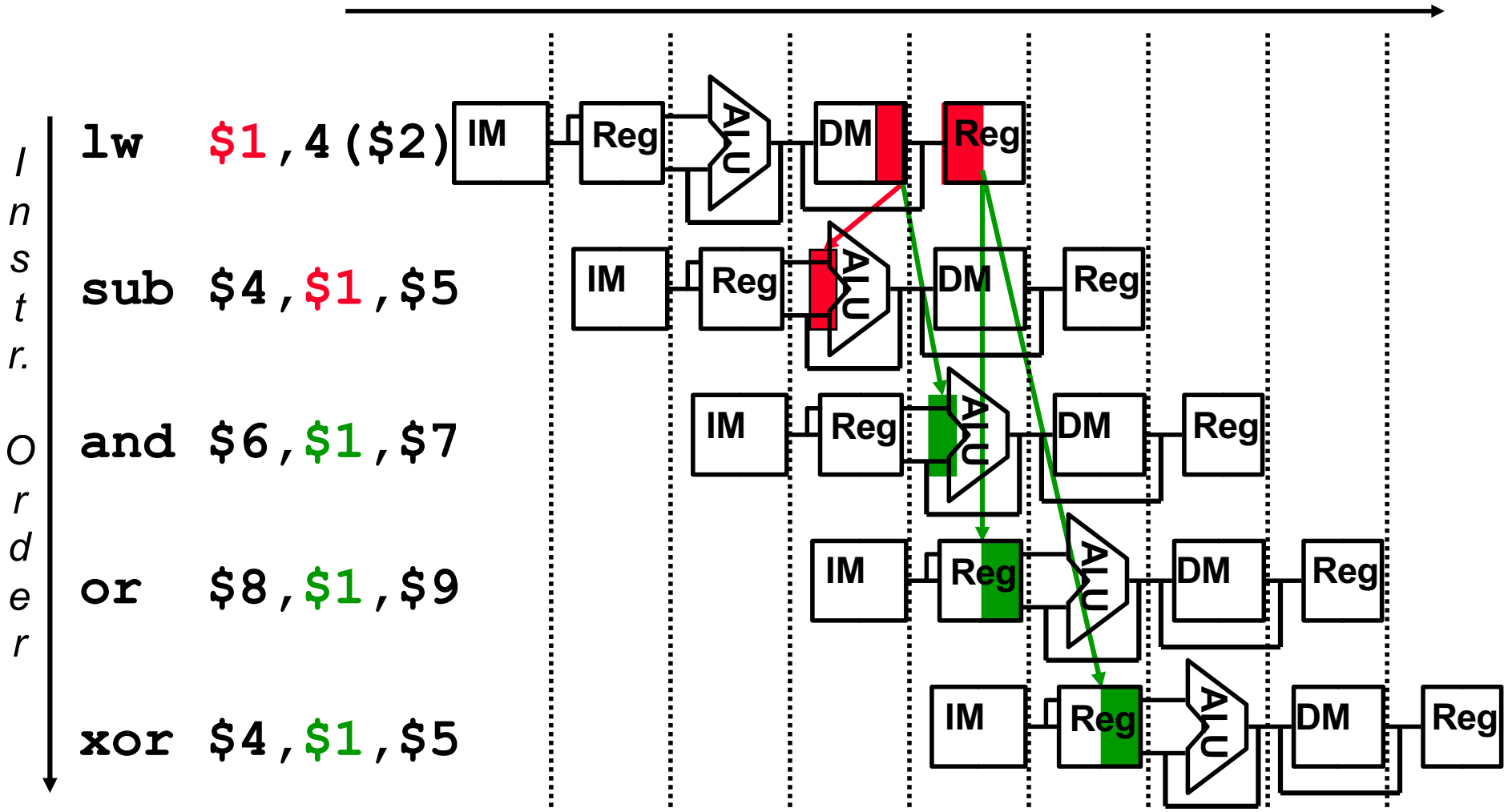
One Way to "Fix" a Data Hazard



Another Way to "Fix" a Data Hazard



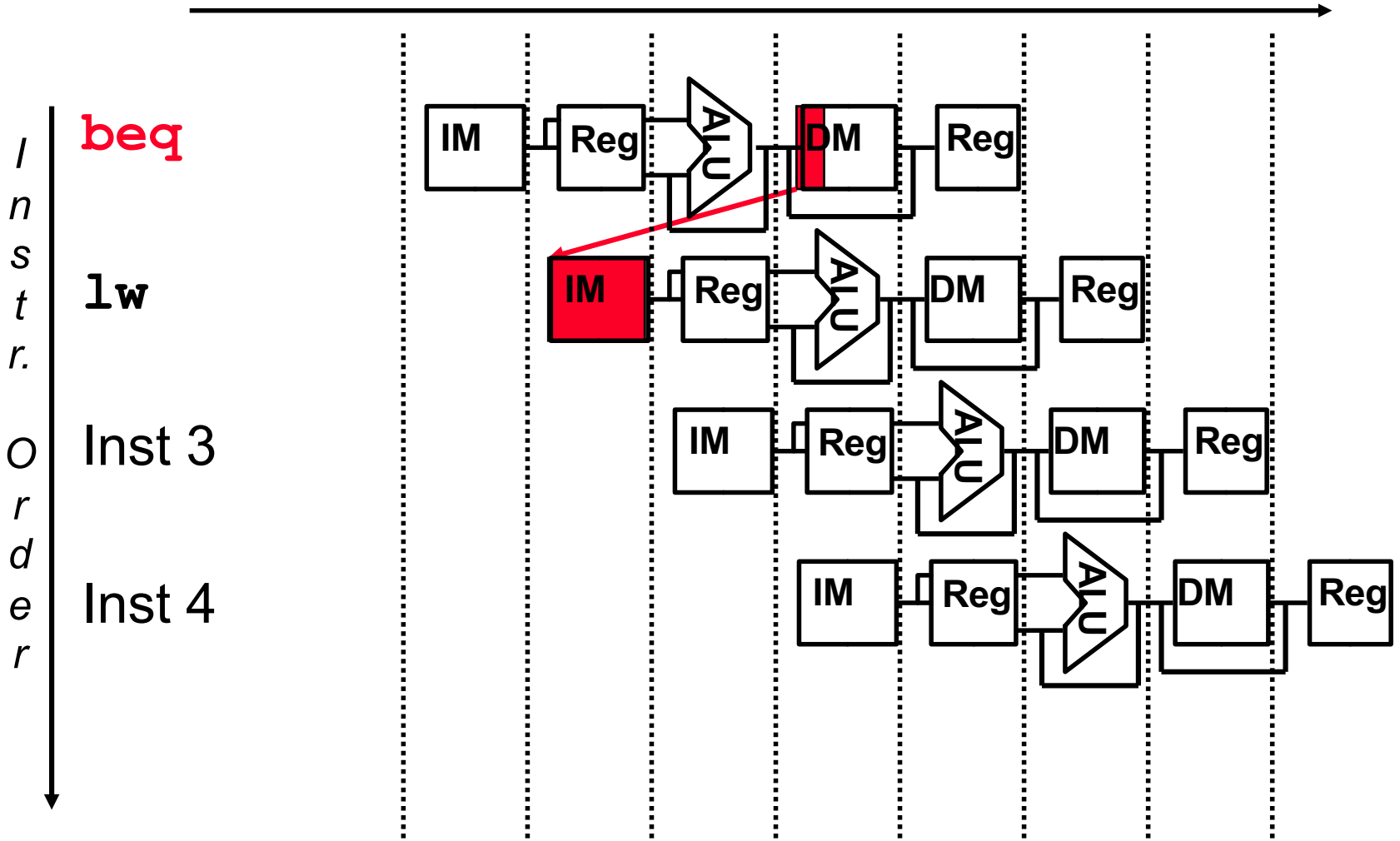
Forwarding with Load-use Data Hazards



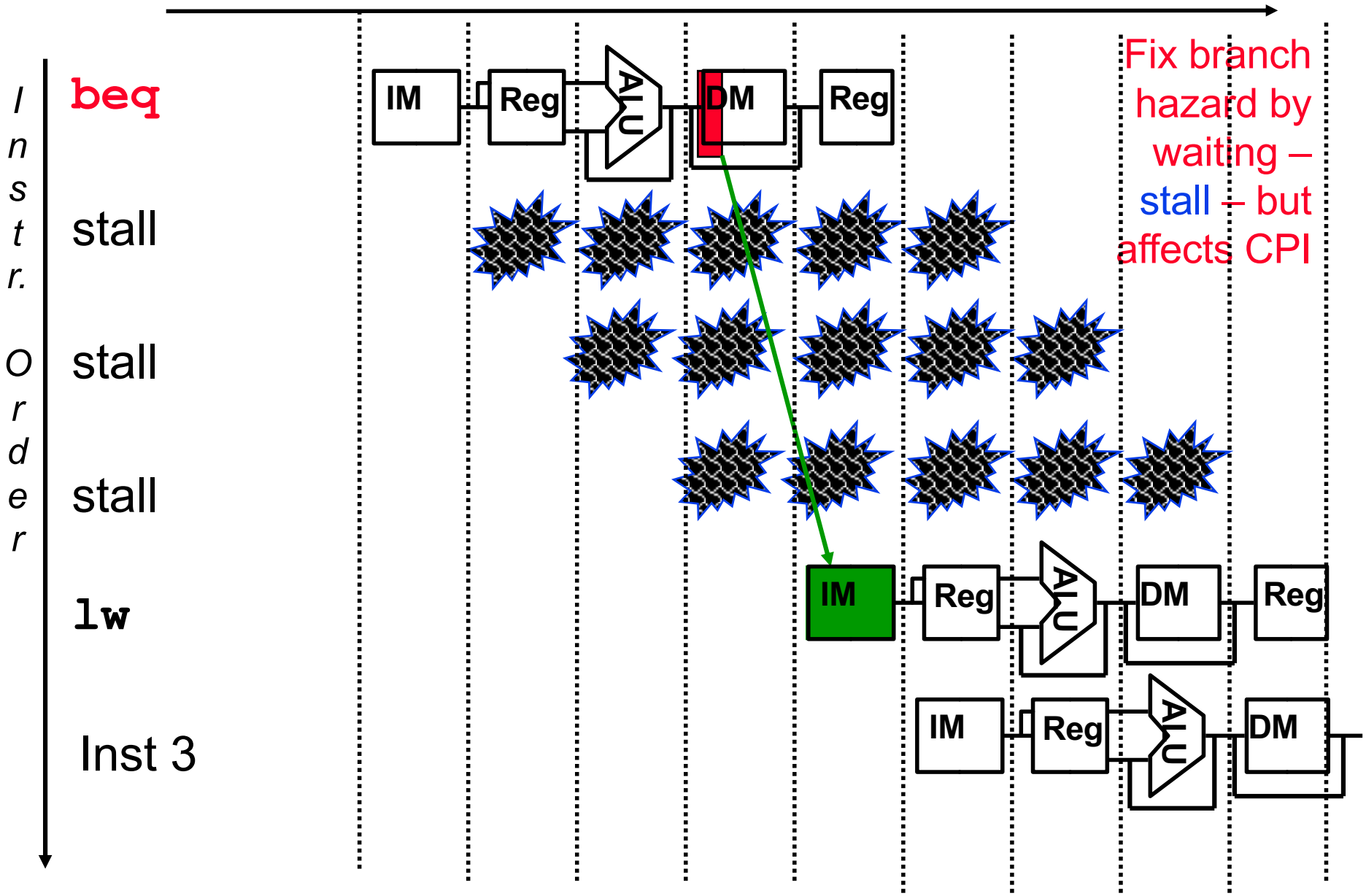
- Will still need one stall cycle even with forwarding

Branch Instructions Cause Control Hazards

- Dependencies backward in time cause hazards

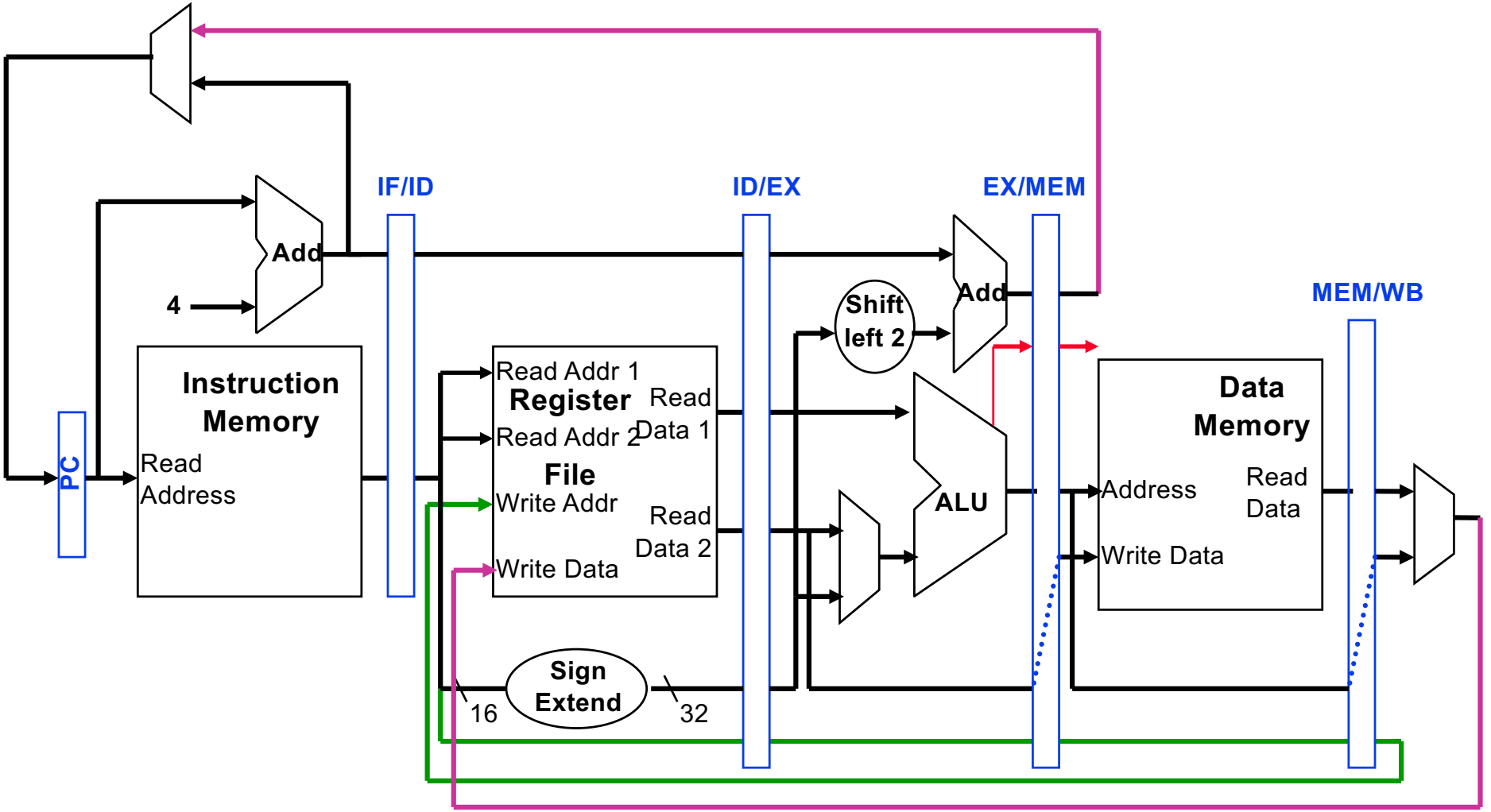


One Way to "Fix" a Control Hazard



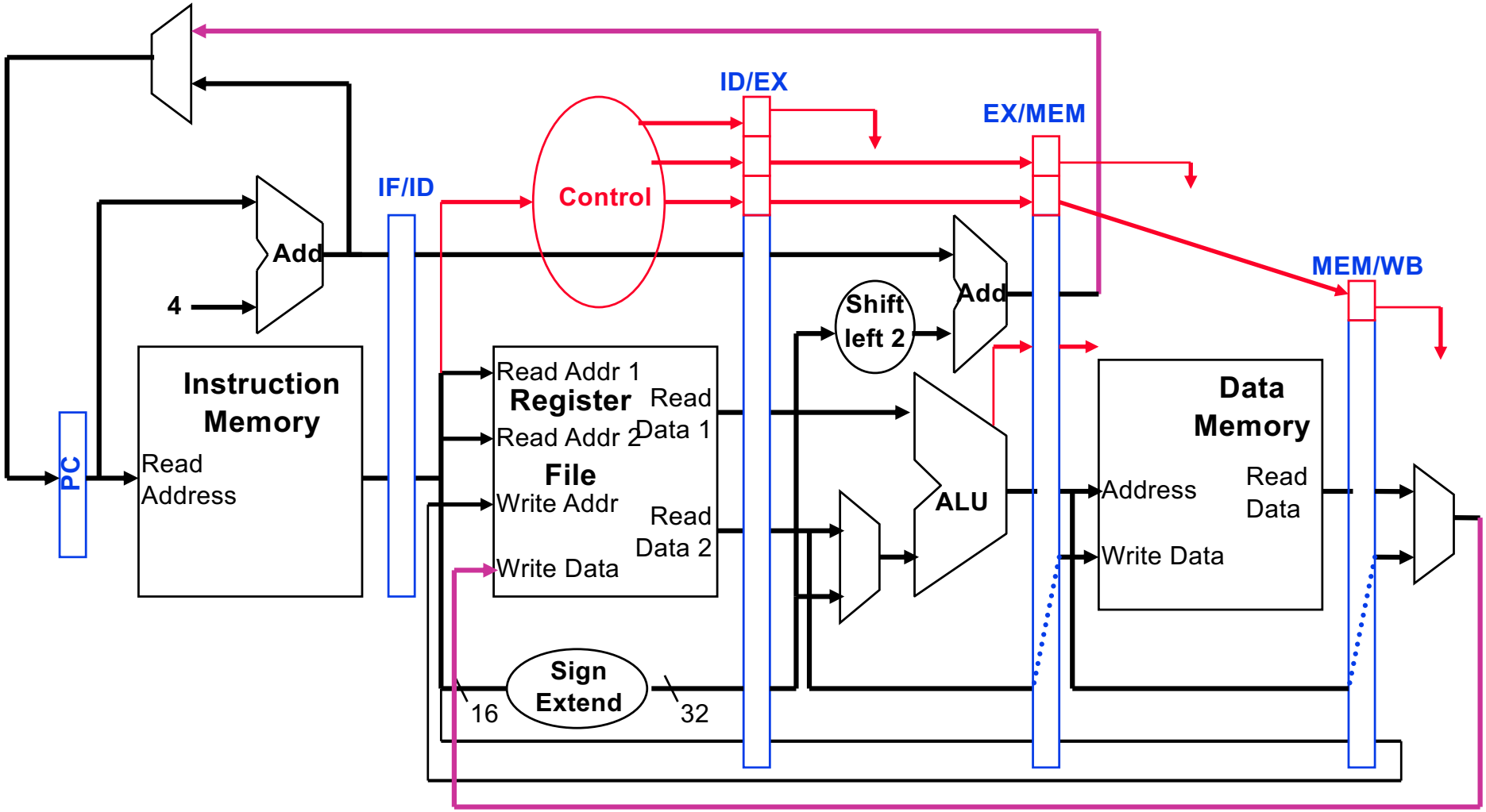
Corrected Datapath to Save RegWrite Addr

- Need to preserve the destination register address in the pipeline state registers



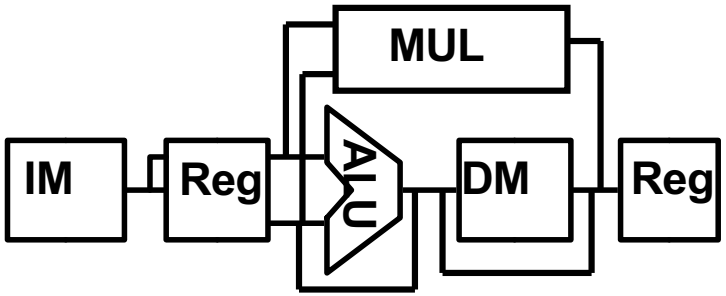
MIPS Pipeline Control Path Modifications

- All control signals can be determined during Decode
  - and held in the state registers between pipeline stages

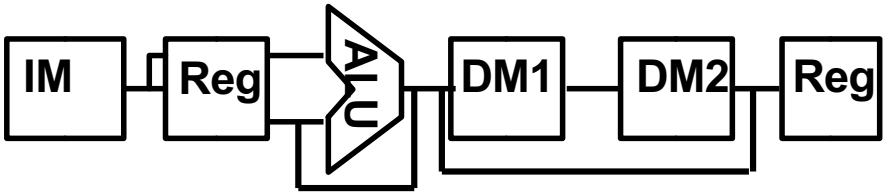


Other Pipeline Structures Are Possible

- What about the (slow) multiply operation?
  - Make the clock twice as slow or …
  - let it take two cycles (since it doesn't use the DM stage)



- What if the data memory access is twice as slow as the instruction memory?
  - make the clock twice as slow or …
  - let data memory access take two cycles (and keep the same clock rate)



Sample Pipeline Alternatives

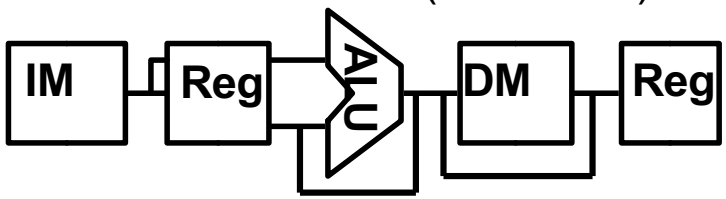
- ARM7

PC update
IM access

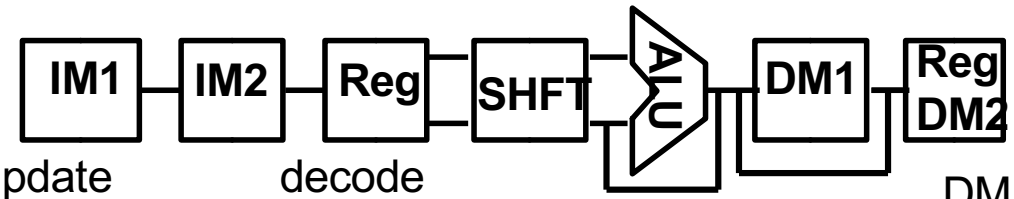
decode
reg access

ALU op
DM access
shift/rotate
commit result
(write back)

- StrongARM-1



- XScale



PC update
BTB access
start IM access

IM access

decode
reg 1 access

shift/rotate
reg 2 access

ALU op

start DM access
exception

DM write
reg write

Summary

- All modern day processors use pipelining
- Pipelining does not help latency of single task, it helps throughput of entire workload
- Potential speedup: a CPI of 1 and a fast CC
- Pipeline rate limited by slowest pipeline stage
  - Unbalanced pipe stages makes for inefficiencies
  - The time to "fill" pipeline and time to "drain" it can impact speedup for deep pipelines and short code runs
- Must detect and resolve hazards
  - Stalling negatively affects CPI (makes CPI less than the ideal of 1)